

# A 3-Dimensional Modeling System Inspired by the Cognitive Process of Sketching

*Matthew Thomas Cook*

Submitted to the Department of Electrical Engineering &  
Computer Science and the Faculty of the Graduate School  
of the University of Kansas in partial fulfillment of  
the requirements for the degree of Master's of Science

## **Thesis Committee:**

---

Dr. Arvin Agah: Chairperson

---

Dr. Nancy Kinnersley

---

Dr. James R. Miller

---

Date Defended

Matthew Thomas Cook © 2007

The Thesis Committee for Matthew Thomas Cook certifies  
that this is the approved version of the following thesis:

**A 3-Dimensional Modeling System Inspired by the Cognitive Process  
of Sketching**

Committee:

---

Chairperson

---

---

---

Date Approved

# Abstract

Three-dimensional computer modeling software is becoming an increasingly important application to modern artists, designers, engineers, architects, and even laypersons. Where modeling applications were once extremely expensive and complex—available only to large businesses and institutions—recent advances in commodity computer hardware and the spread of consumer demand now places these applications within the reach of a much wider audience. Although commercial modeling applications are powerful, their interfaces are complex and unintuitive to artists and designers. Furthermore, most lack the ability to quickly create models in the early stages of design. The 3-D modeling and user interface research fields have begun to address these issues with sketch-based modeling interfaces, which purport to allow users to quickly sketch simple models, often combining modeling methods based on traditional physical artistic techniques, and utilizing drawing input from a digitizing tablet.

Sketch-based modeling research has made a great deal of progress. However a survey of current techniques reveals three reoccurring areas of oversight. (1) Most modeling interfaces attempt to mimic a specific artistic medium or technique—providing the user with an uneasy facsimile of the original, and the system with impoverished 3-dimensional input. (2) Many interfaces place too little consideration on the user’s necessarily 2-dimensional experience of the modeling process. And, (3) Projects that utilize a digitizing tablet focus only on positional input and ignore the rich source of additional dynamic physical pressure, hover, and tilt information provided by the tablet hardware.

To address these shortcomings, a sketch-based modeling prototype application and modeling interface were developed that focus on the *cognitive* aspect of the traditional sketching process, rather than on a specific technique. The system uses sketching strokes drawn onto user-adjustable 2-dimensional drawing surfaces

within a 3-dimensional environment to quickly generate 3-D models through flexible and intuitive sweep and extrusion methods. The interface also features a number of novel *tablet gestures* that utilize dynamic physical information from a digitizing tablet to control the application with natural and intuitive gesticulations.

Preliminary assessment of the modeling methods suggests that the interface is effective for quick or preliminary modeling activities, and that some natural media related techniques may be required to properly address all areas of sketching technique. Furthermore, the system of tablet gestures is shown to be an intuitive and compelling channel of application control, and an exciting prospect for further research.

# Acknowledgments

I would like to thank my academic advisor, professor, and committee chairperson, **Dr. Arvin Agah**. It is thanks to his advice, support, guidance, and patience that I was able to develop my research interests into this project, and it is thanks to his persistent help that I was able to finish it.

I would like to thank my professor and committee member, **Dr. Nancy Kinnersley**. Her infectious enthusiasm for mathematics and computer science theory have shaped the way I approach research problems, and her guidance in my undergraduate career encouraged me to pursue a graduate degree.

I would like to thank my professor and committee member, **Dr. James Miller**. It was his class that fueled my interest in computer graphics and taught me the skills utilized in this work. I will always remember fondly the graphics projects I created in his classes. I would also like to thank Dr. Miller for his invaluable help and advice on my project.

I would like to thank **Dr. Sanae Eda** for encouraging my research ethic.

I would like to thank **Dr. Albin Jones** for the calculus.

I would like to thank **Kuma Folmsbee** and **Nick Menefee** for keeping me sketching.

I would like to thank **Dr. Perry Alexander, Garrin Kimmell, Nick Frisby, Phil Weaver, Megan Lehnher, Jason Agron, Jennifer Streb, Mark**

**Snyder, Will Kritikos, Jim Stevens, Seth Warn, Ilya Tabakh, Joel Van Eenwyk, Zach Parr**, and anyone else in Lambda Group and the RC Reading Group for endless help and support.

I would like to thank **Dr. Costas Tsatsoulis** for graduate advice.

I would like to thank **Raleigh Ledet** for help with Wacom tablet internals.

Most of all, I would like to thank my parents, **Jan** and **Tom Cook**. Without their help and support I could never have developed my research and completed this project. I am endlessly grateful for the opportunities they have provided me.

*Thank You*

# Contents

<b>Acceptance Page</b>	<b>i</b>
<b>Abstract</b>	<b>ii</b>
<b>Acknowledgments</b>	<b>iv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Problem Statement and Approach . . . . .	5
1.3 Thesis Structure . . . . .	8
<b>2 Sketching Background</b>	<b>11</b>
2.1 The What and Why of Sketching . . . . .	12
2.1.1 Sketching as a Design Tool . . . . .	17
2.1.2 Sketching from the Observer’s Point of View . . . . .	19
2.1.3 Sketching as a Thinking Process . . . . .	22
2.2 The Link Between Sketching and Creative Thought . . . . .	24
2.2.1 Sketching as a Mental Process— <i>Feedback</i> . . . . .	25
2.2.2 The Activity of Sketching— <i>Incremental Refinement</i> . . . . .	28
2.2.3 Sketching Technique— <i>Overdrawing</i> . . . . .	29
2.3 Sketching by Hand . . . . .	32
2.3.1 The Sketching Process . . . . .	34
2.3.1.1 Kinds of Sketching Lines . . . . .	37
2.3.2 Limitations of Sketching by Hand . . . . .	47
2.4 Sketching with the Computer . . . . .	50
2.4.1 Computer Sketching Tools . . . . .	52

2.4.2	User Interfaces . . . . .	57
2.4.2.1	Gestures . . . . .	59
2.4.2.2	Pressure . . . . .	73
2.4.3	Computer Sketching Applications . . . . .	76
2.4.3.1	Commercial Software . . . . .	78
2.4.3.2	Research Applications . . . . .	87
2.4.4	Limitations of Sketching with the Computer . . . . .	95
<b>3</b>	<b>Modeling Background</b>	<b>99</b>
3.1	What Is Modeling? . . . . .	100
3.2	Uses of Modeling . . . . .	101
3.3	Limitations of Physical Modeling . . . . .	107
3.4	Modeling With the Computer . . . . .	109
3.4.1	Current Computer Modeling Techniques . . . . .	111
3.4.1.1	Construction from Primitives . . . . .	111
3.4.1.2	Curves, Surfaces, and Control Point Manipulation	114
3.4.1.3	Procedural Modeling . . . . .	117
3.4.1.4	Constructive Solid Geometry . . . . .	121
3.4.1.5	Implicit Modeling . . . . .	123
3.4.2	Current Modeling Applications . . . . .	125
3.4.3	Limitations of Computer Based Modeling . . . . .	129
<b>4</b>	<b>Bridging the Gap Between Sketching and Modeling</b>	<b>132</b>
4.1	Computerized Animation Systems . . . . .	133
4.2	Model Annotation . . . . .	142
4.3	Sketch-Based Modeling . . . . .	147
<b>5</b>	<b>Related Work in Sketch-Based Modeling</b>	<b>152</b>
5.1	Sketch Input . . . . .	153
5.2	Gesture Created Primitives . . . . .	162
5.3	Artificial Intelligence and Machine Learning . . . . .	175
5.4	Optimization and Line Labeling . . . . .	182
5.5	Blobby Inflation . . . . .	193
5.6	Height-Fields and Shading . . . . .	201
5.7	Deformation . . . . .	217



5.8	Contour Curves and Drawing Surfaces . . . . .	230
5.9	Stroke-Based Construction from Sweeps and Extrusions . . . . .	238
5.10	Summary . . . . .	245
<b>6</b>	<b>Requirements and Design</b>	<b>247</b>
6.1	Aims and Objectives . . . . .	247
6.2	Design Principles . . . . .	249
6.2.1	Principle I: Sketch is the Basis of Input . . . . .	249
6.2.2	Principle II: Observe a Hierarchy of Sketching Actions . . . . .	250
6.2.3	Principle III: The Goal is 3-D . . . . .	251
6.2.4	Principle IV: All Lines are Useful . . . . .	252
6.2.5	Principle V: Prefer Consistency Over Interactivity . . . . .	252
6.2.6	Principle VI: Prefer Tablet Input Whenever Possible . . . . .	253
6.2.7	Principle VII: Sketching Should Be Fun . . . . .	254
6.3	Application Requirements . . . . .	254
6.3.1	Tablet Input . . . . .	254
6.3.2	Stroke Input . . . . .	255
6.3.3	Environment and Drawing Surfaces . . . . .	255
6.3.4	View Adjustment . . . . .	256
6.3.5	2-D to 3-D Conversion . . . . .	256
6.3.6	Modeling System . . . . .	256
6.3.7	Graphical User Interface . . . . .	257
6.4	Application Design . . . . .	258
6.4.1	Target System . . . . .	258
6.4.2	Graphical User Interface . . . . .	260
6.4.3	Tablet Input . . . . .	267
6.4.4	Tablet Gestures . . . . .	272
6.4.4.1	Brush-Off / Rehearsal . . . . .	273
6.4.4.2	Pounce . . . . .	274
6.4.4.3	Joystick . . . . .	277
6.4.4.4	Flick . . . . .	279
6.4.4.5	Low-Angle Push . . . . .	281
6.4.5	Stroke Format . . . . .	284
6.4.6	Drawing Feedback and Stroke Conversion . . . . .	286

6.4.7	Curve Correction . . . . .	291
6.4.8	Drawing Interface . . . . .	298
6.4.9	Environment and Drawing Surfaces . . . . .	300
6.4.10	View Adjustment . . . . .	308
6.4.11	2-D to 3-D Conversion . . . . .	317
6.4.11.1	Conversion System Design . . . . .	317
6.4.11.2	Artistic Basis . . . . .	320
6.4.11.3	Conversion System Interface . . . . .	323
<b>7</b>	<b>Implementation</b>	<b>337</b>
7.1	System Architecture and Codebase . . . . .	338
7.2	Strokes . . . . .	344
7.2.1	Filtration . . . . .	344
7.2.2	Classification . . . . .	346
7.2.3	Conversion . . . . .	347
7.2.3.1	Representation . . . . .	347
7.2.3.2	Stroke Fitting . . . . .	353
7.2.4	Bézier Curve Length . . . . .	359
7.3	3-D Construction Foundations . . . . .	361
7.3.1	Curve Conversion . . . . .	362
7.3.2	Model Representation . . . . .	366
7.4	Sweep . . . . .	373
7.4.1	Frenet Frame . . . . .	374
7.4.2	Sweep Construction . . . . .	379
7.5	Generalized Cylinder . . . . .	385
7.5.1	Stroke Averaging . . . . .	386
7.5.2	Alignment Frame and Scaling . . . . .	390
7.5.3	Generalized Cylinder Construction . . . . .	393
7.6	Visual Environment . . . . .	397
7.6.1	Environmental Lighting Model . . . . .	398
7.6.2	Surface Properties . . . . .	400
<b>8</b>	<b>Conclusions</b>	<b>406</b>
8.1	Discussion . . . . .	409

8.1.1	Sketching Interface . . . . .	409
8.1.2	Tablet Gestures . . . . .	416
8.1.3	User Interface . . . . .	423
8.1.4	Modeling Methods . . . . .	426
8.1.5	Summary . . . . .	430
8.2	Contributions . . . . .	431
8.3	Limitations . . . . .	432
8.4	Future Work . . . . .	434
	<b>References</b>	<b>437</b>
	<b>A Tablet Feature Examination</b>	<b>462</b>
	<b>B Modeling Examples</b>	<b>466</b>

# List of Figures

1.1	Crockett Johnson’s Harold and the Purple Crayon [Johnson, 1981]	4
2.1	Examples of Sketching . . . . .	14
2.2	Infrared Reflectogram of Van Gogh’s <i>The Zouave</i> [Ives & Stein, 2005]	20
2.3	Examples of Overdrawing . . . . .	31
2.4	Animation Sketches from Disney’s Snow White and the Seven Dwarfs [Johnston & Thomas, 1995] . . . . .	33
2.5	Progression of a Sketch . . . . .	38
2.6	Gesture Correction Interface in Quick-Sketch [Egglı <i>et al.</i> , 1995]	65
2.7	Marking Menus . . . . .	72
	(a) Hierarchical Marking Menu Concept [Kurtenbach <i>et al.</i> , 1994]	72
	(b) Marking Menu in SketchBook Pro . . . . .	72
2.8	MacPaint Interface . . . . .	80
3.1	Parametric Surface Examples . . . . .	115
	(a) Bézier Surface . . . . .	115
	(b) NURBS Surface . . . . .	115
3.2	Subdivision Surface Example [DeRose <i>et al.</i> , 2000] . . . . .	119
4.1	Mickey Mouse Ears [Johnston & Thomas, 1995] . . . . .	150
5.1	Expectation List [Pereira <i>et al.</i> , 2001] . . . . .	168
5.2	Suggestive Interface [Igarashi & Hughes, 2001] . . . . .	173
5.3	Three Views of a Circle . . . . .	178
5.4	Automatic Photo PopUp [Hoiem <i>et al.</i> , 2005] . . . . .	180
5.5	The Necker Cube Illusion . . . . .	182
5.6	Angular Distribution Graph [Masry <i>et al.</i> , 2005] . . . . .	189

5.7	Stilton [Turner <i>et al.</i> , 1999]	192
5.8	Teddy [Igarashi <i>et al.</i> , 1999]	194
5.9	Smooth Sketch [Karpenko & Hughes, 2006]	198
5.10	Digital Elevation Model of Martian Surface [USGS, 2003]	206
5.11	3DPaint [Williams, 1990]	207
5.12	Comparison of Height-Field and Shading [Rushmeier <i>et al.</i> , 2003]	209
5.13	Concave vs. Convex Shading Illusion	211
5.14	2½-D Model from Comic Book Image [Kerautret <i>et al.</i> , 2005]	215
5.15	Early Volumetric Sculpting [Galyean & Hughes, 1991]	221
5.16	Sculpting a Chair with VolVis [Wang & Kaufman, 1995]	221
5.17	Stamp Deformation with User Created Tool [Ferley <i>et al.</i> , 1999]	223
5.18	Creation of a Muffin with Velocity Paint [Lawrence & Funkhouser, 2003]	226
5.19	Orthogonal Deformation Strokes [Cherlin <i>et al.</i> , 2005]	227
5.20	Stroke-Based Mesh Deformation [Kho & Garland, 2005]	229
5.21	Tape Drawing [Grossman, 2004]	232
5.22	Bounding Box Modeling Stage [Grossman <i>et al.</i> , 2001]	234
5.23	Construction with Rotational and Cross Sectional Blending Surfaces [Cherlin <i>et al.</i> , 2005]	243
6.1	Testing Platform	259
6.2	Application Windows	261
6.3	Tool Palette	262
6.4	Plane Palette	263
6.5	Tablet Info Panel	263
6.6	Application Mode Cursors	265
6.7	Wacom Intuos2 Digitizing Tablet and Accessories	268
6.8	Wacom Stylus Transducers	269
6.9	Raster vs. Vector Graphics	284
6.10	Overdrawing Correction [Pereira <i>et al.</i> , 2000]	292
6.11	The Cleanup Artist	296
6.12	Sketch Strokes	297
6.13	Application Environment	301
6.14	Orthogonal and Perspective Projection	311

6.15	Deconstruction of a Subject into Basic Shapes . . . . .	321
6.16	Polygon Construction . . . . .	325
6.17	Drawing a Box By Hand . . . . .	328
6.18	Sweep Construction . . . . .	329
6.19	Drawing a Vase By Hand . . . . .	333
6.20	Generalized Cylinder Construction . . . . .	335
7.1	Application Architecture Diagram . . . . .	341
7.2	Curve Continuity . . . . .	350
7.3	Stroke Continuity Examples . . . . .	359
7.4	Mesh Structure of Two Squares . . . . .	368
7.5	Halfedge Query Diagram . . . . .	369
7.6	Frenet Frame . . . . .	375
7.7	Frenet Frame Twist [Davison, 2003] . . . . .	377
7.8	Sweep Stroke Conversion and Frames . . . . .	383
7.9	Sweep Contour Curve Evaluation . . . . .	384
7.10	Final Sweep Construction . . . . .	385
7.11	Step-wise Polygon Mosaic Creation [Christiansen & Sederberg, 1978]	388
7.12	Stroke Average by Step . . . . .	389
7.13	Stroke Average by Length . . . . .	391
7.14	Generalized Cylinder Stroke Conversion, Frames, and Scale Spans	395
7.15	Sweep Contour Curve Evaluation . . . . .	396
7.16	Final Generalized Cylinder Construction . . . . .	397
7.17	Three Point Lighting . . . . .	401
B.1	Three Simple Forms . . . . .	467
B.2	Monuments . . . . .	468
B.3	Flat Sweep . . . . .	469
B.4	Distant Sweep . . . . .	470
B.5	Torus . . . . .	471

# List of Tables

A.1	Summary of Tablet Input: Hovering . . . . .	463
A.2	Summary of Tablet Input: Pen Down . . . . .	463
A.3	Summary of Tablet Input: Pressure . . . . .	464
A.4	Summary of Tablet Input: Tilt . . . . .	465

# List of Notes

1	Non-Photorealistic Rendering . . . . .	141
2	Hammerspace . . . . .	248
3	Objective-C++ . . . . .	340



# Chapter 1

## Introduction

### 1.1 Motivation

If you had to write a letter to a colleague, a report for class, or a flyer for the office party, how would you do it? What would you use? Thirty years ago you would probably have gone to your desk drawer, taken out a pad of paper and a pencil and set to work. Then, if you were lucky enough to have one, after one or two drafts you might sit down at the typewriter and type up your composition. Today however, for most of us the composition process begins at our computer's keyboard. It's not that there's anything wrong with pad and pencil but we've all come to realize that the benefits offered by a word processing program not only make the writing process easier, but give us more power and more flexibility. Even for mundane writing tasks the convenience and versatility offered by the computer make it more attractive than the traditional alternatives.

The same is true of many of the common tasks we now use our computers for every day. Fifteen years ago the best photos from the family vacation—the ones that would go in the holiday cards—were the two or three shots among dozens of

rolls that happened to turn out just right. Today photo editors and paint programs let us pull aside the rough gems and burnish away the poor lighting, re-eye, and errant fingers in the frame. To professional photographers these actions are nothing new or magical, but because of the expertise, equipment, and above all time required to pursue them, before the introduction of image software they were out of the reach of the amateur.

Now consider another task. If you had to explain to the baker what the cake should look like for your daughter's wedding, or if you wanted to design a few set pieces for your nephew's school play, how would you do it? What would you use? No matter what kind of computer you have, for most of us the tool of choice is a blank piece of paper and something to sketch with. Sketching is an intuitive way to think, plan, and convey visual or spatial information, and the activity is so innate that even people who claim they can't draw won't hesitate to sketch out an idea in order to get their point across.

However the amount of information that can reliably be communicated through a sketch is limited. By its very nature, sketching is inherently a 2-dimensional medium and so the 3-dimensional features of a subject can only be suggested. Professional artists study to develop techniques like foreshortening and perspective that allow them to manipulate 2-dimensional drawings to give the illusion of depth. However these techniques are difficult to master, and like a professional photographer's darkroom skills, for the average person out of reach. Even for practiced and professional artists a realistic rendering could require hours of work, certainly more than is justified for a simple sketch. However, although artists or laypersons may only be able to produce a 2-dimensional rendering, their mental image of the subject contains a full—if under-developed—understanding of its 3-D

structure. In essence we are limited by the tools at our disposal.

Again computers can offer a way to address these issues. Computer aided design and 3-D modeling software allow a user to design 3-dimensional structures directly that, like a wooden model or clay statue, capture all of the information necessary to completely describe the subject's shape. However whereas the convenience and versatility of word processors and paint programs stem in part from how easy they are to use, most 3-D modeling software is anything but. Derived from their early roots as engineering tools and targeted primarily at the high-end professional market, most 3-D applications are highly complex and rely on a great deal of technical expertise. For the average computer user the sheer inertia of learning and using these applications to create even simple models far outweigh any benefits 3-D modeling may have over traditional sketch. Coupled with the extremely high cost of commercial 3-D modeling applications these factors have prevented computers from moving in on sketching in the same way they have for writing reports or fixing photos.

But consider for a moment if things were different. Imagine for example that you could sketch your ideas in three dimensions. Like Harold and his purple crayon or SpongeBob and his pencil (see figure 1.1), what if you could draw your designs in midair and watch as they pop into existence. Not only would this provide a more versatile and expressive means of communicating your ideas, but at the same time, freed from the confines of the 2-dimensional page, you would no longer need to worry about advanced artistic techniques like foreshortening and perspective. Instead you could draw the simple shapes you see in your mind's eye and presto, your model would take shape. If you think carefully about the process you've just imagined, is it all that different from the sketch you put down on a

piece of paper? Physically, of course, drawing in mid air is quite a departure from touching pencil to paper, but because your design is 3-dimensional in your head, mentally you're doing almost the same thing. In fact, because there's no need to translate the 3-dimensional view into a 2-dimensional representation, drawing your ideas in space may even be more direct.



**Figure 1.1.** A promotional poster for Crockett Johnson's classic children's book character Harold and his purple crayon. Harold uses his purple crayon to draw objects which his vivid imagination makes real. [Johnson, 1981]

While commercial computer modeling software has forged ahead, stacking new features and capabilities on top of the same basic interface foundations, the re-

search community has attempted for some time to improve computer modeling interfaces. Based on the same sentiments of the versatility and importance of traditional sketching a large segment of this community is now focused on the development of *sketch-based modeling*. These projects are an attempt to develop a more intuitive and usable method of 3-dimensional modeling. At the same time, much of this research has also focused on servicing the same rough, experimental, early planning stages of the design process in which sketching plays a key role—a segment of the modeling market that is underserved by the current crop of commercial modeling software. The overarching goal of these projects is to develop a system that can tap into a user’s artistic skills, both innate and learned, and allow them to use those abilities to create 3-dimensional models.

## 1.2 Problem Statement and Approach

Over nearly a decade of work researchers are now beginning to hone in on promising new modeling techniques and interface designs, some of which are already affecting high end computer modeling software. However the dream of a simple and effective 3-D modeling tool that could be used both by professionals for early design work, and by average users to develop their ideas, has yet to fully materialize. Despite the diversity of approaches in current sketch-based modeling research, there are three areas in which research efforts seem to be repeating the same mistakes.

First, in an attempt to draw out the user’s traditional artistic skills many researchers have developed systems that attempt to mimic specific artistic techniques—from shading and perspective drawing to carving and sculpture. Although this approach has met with some limited success, the methodology also breeds frus-

tration from users who are promised a familiar experience but provided with little more than an ersatz copy. No matter the quality of the interface or the correspondence of its features to ‘the real thing’, the computer will never be able to replicate the genuine experience, nor should it. The goal is to create 3-D modeling software—something that is beyond the scope of traditional 2-D drawing. Rather than focusing on the development of systems that mimic physical artistic techniques, the interface should instead cater to the mental processes that underlie those techniques. Targeting these components of the sketching process will help to bridge the gap between traditional skills and the new medium, even where traditional technique is incongruous.

Second, in an area where more attention to physicality should be applied, it is lacking. Although the product of the modeling process is three-dimensional, the user’s entire experience of that process is through the computer’s 2-dimensional display and input devices. Any modeling process that hopes to be intuitive must take this inherent inconsistency into account.

To address these first two issues, this thesis describes the design and development of a sketch-based modeling application interface and 3-D modeling process that derives its inspiration from the way we *think* about sketching. The design of the system draws upon aspects of other sketch-based modeling research efforts that reflect this design philosophy. Utilizing a digitizing tablet and a 2-dimensional sketching idiom, the foundations of the input system are based on basic physical attributes of traditional sketching. Using sketching as the jumping off point for 3-dimensional construction, the system features a modeling method based on generalized sweeps and extrusions to translate the user’s 2-dimensional drawing input into 3-D model components. These methods mimic the way tradi-

tional artists break down and reconstruct 3-dimensional forms in terms of basic shapes and structures, and how artists represent and render those structures in 2-dimensional drawings.

The third aspect that appears to be under appreciated in other sketch-based modeling research is the full utilization of the available interface hardware. Recognizing the importance of sketching, many systems utilize digitizing tablets and other pen-based input methods to help their users interact in a more natural sketching style. However beyond the physical use of a digitizing pen to evoke the feeling of drawing, few applications give any consideration to other physical information that could be provided by this input method. Along with basic positional information, digitizing tablets are capable of generating dynamic information about the pressure of the user's pen, the type of tool in use, the proximity of the tool to the tablet's surface, and the orientation of the user's grip and stroke.

In response to this untapped channel of information, this thesis also describes the design and implementation of a number of novel tablet gestures that allow the user to interact with the system by means of naturally inspired physical hand and drawing movements. In testing the system of gesticulations in the context of a sketch-based 3-D modeling application, the author's research goals are threefold: first, to demonstrate the utility of this underutilized source of data; second, to determine if the drawing inspired physical movements can successfully translate as intuitive channels for application command and control; and third, to see if physical gestures can provide a more natural means of navigating the 3-dimensional modeling environment via the 2-dimensional input system.

## 1.3 Thesis Structure

The remainder of this thesis is organized as follows. We begin in **Chapter 2** with an in-depth discussion of sketching. We will first examine the process of sketching and its links to the creative process. We will then explore the traditional use of sketching, both as an artistic practice and by laypersons as a tool for planning and design. Finally, we will explore how sketching practice has translated into the computer field, including the tools and applications used by computer sketching artists. This discussion will form the foundations of our use of sketch in the modeling arena.

In **Chapter 3**, we will explore the current practice of modeling, placing a greater focus on the growing field of 3-dimensional computer modeling. In this context, we will discuss the current state of computer modeling applications and interface techniques, highlighting some of the limitations of this field that sketch-based modeling hopes to address.

**Chapter 4** briefly addresses two related examples of a transition within an industry from a primarily paper-based traditional art or professional technique, to a digitized workflow. The first explores the animation process and the introduction of computerized animation systems, and the second centers on the annotation of existing 3-dimensional architectural models. Observing how these transitions were made, and how traditional techniques are involved is instructive as a means of gauging how best to bring about similar change for the field of modeling.

**Chapter 5** provides an in-depth examination of related work in the field of sketch-based modeling. Articles and projects by a number of authors have been categorized into general approaches to the modeling task. This chapter covers topics including methods of handling sketching input in a modeling system and



a number of alternative construction methods and model representations. The chapter concludes with an examination of several projects that had the greatest influence on the designs of the current project.

In **Chapter 6** we will discuss the design of the prototype modeling application developed for this project. The chapter begins with an examination of the underlying design principles that guided the process and the requirements for the application. The chapter then describes in detail the design of each aspect of the system, including the motivations and justifications for each design decision.

**Chapter 7** continues on from Chapter 6 with a thorough description of the implementation of each feature of the application. This chapter contains descriptions of the underlying representations of various application components, describes the mathematical foundations for many of the construction methods utilized by the software, and gives descriptions of the algorithms derived from those foundations. This chapter also addresses aspects of the visual environment of the application, and describes the techniques and algorithms used to provide the user's visual experience.

Finally, **Chapter 8** presents a discussion of the general impressions of the application gleaned from preliminary testing and development, and the author's brief concluding remarks. In the first section the author's opinions of the efficacy of each feature, as well possible courses of further research and improvement are presented to provide the reader with a preliminary assessment of the methods described in this research effort. Next, the contributions of this research effort are enumerated and limitations of the current work are highlighted. This chapter concludes with an examination of several possible avenues of future research.

This thesis concludes with a complete listing of **References** cited in this work,

and two appendices. **Appendix A** contains tables describing various features of digitizing tablet hardware, and their utility and suitability as methods of application control. **Appendix B** contains a number of example models created with the modeling system.

## Chapter 2

# Sketching Background

The term ‘sketch’ or the practice of ‘sketching’ entered the English lexicon in the mid to late 17th century to refer to the practice of making rough, preliminarily drawings [Simpson & Weiner, 1989]. Although many modern artists produce final works that could be considered sketches, sketching is usually done as preparatory work in the early design process of some larger or more involved project. Despite the term’s close association with the fine arts, the idea of ‘sketching out a plan’ or ‘making a sketch of one’s idea’, even when no actual drawing is involved, is a widely used and particularly vivid metaphor, and hints at the fact that what we call sketching is really an amalgamation of artistic techniques and very innate skills.

Although the term sketching was not in use until relatively late in human history, artists, designers, engineers, and thinkers of all stripe have probably used something similar to sketching to help them develop their ideas from nebulous kernels of inspiration into concrete designs and images. The practice of sketching is so central to the creative process that it is almost instinctual, and examples of designer’s notes and artist’s doodlings exist from well before a word existed to

describe them. Perhaps the most famous examples of sketching come to us from the 15<sup>th</sup> century artist and inventor Leonardo da Vinci. Although you're likely to see his notebooks and writings on display in a fine museum, in his day his drawings of eerily prescient flying machines or groundbreaking anatomical studies were not created as works of art, but as the byproducts da Vinci's curious mind at work. By the same token, each of us creates countless scribbles in the margins of papers or thumbnail plans on pads or scraps of paper.

In this section we will take a closer look at the practice of sketching, both in a traditional sense, and how modern technology is beginning to make its way into the field. To understand why sketching is so important to the design process, let's examine just what the process of sketching entails and how designers, engineers, artists, and every day people use sketch.

## 2.1 The What and Why of Sketching

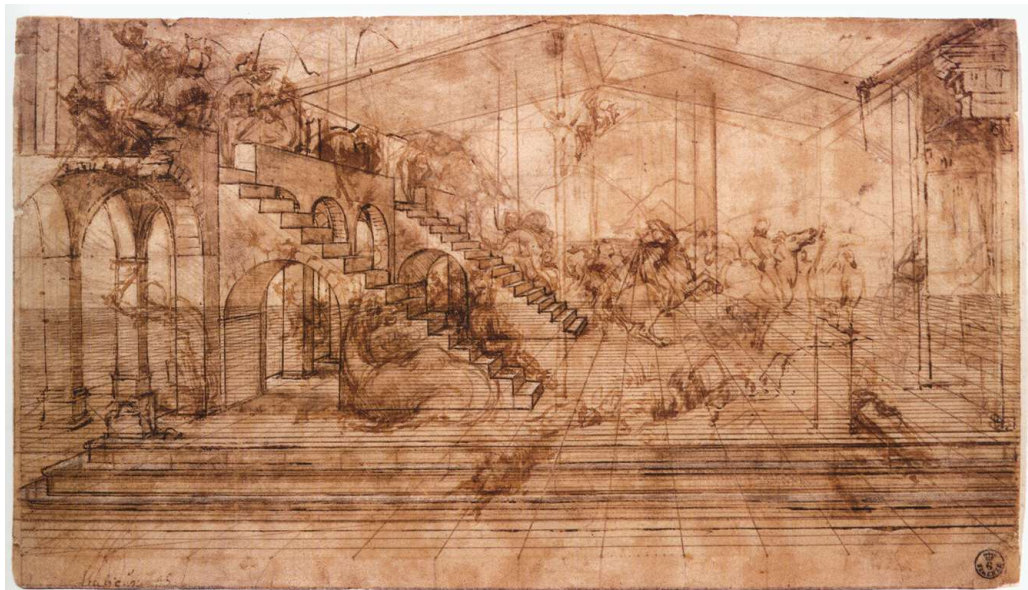
Let us begin by formally defining what we mean by sketching. *Sketching* is a form of drawing used for planning, preparation, or experimentation. Sketches can vary from a loose, messy, and even trifling doodles, all the way to a substantial and detailed work of art, however for the purposes of this discussion, we will limit ourselves to drawings created with the specific intent of representing some visual or spatial form. These can include the drawings you might make to plan something you will build, simple thumbnails to describe what something might look like, an initial drawing made before a more detailed work of art, fanciful characters drawn in the margins of a book, or any other time one uses drawings as a tool. Overall, the thing that differentiates a sketch from other drawings is not necessarily how it looks, but the intention behind its creation.

As we have said, sketching is not an activity relegated to professional artists and designers. It's true that these professions make heavy use of sketching, but people from all walks of life produce sketches, often for the very same reasons. To aid in our discussion of sketching, we will refer to the person who produces a sketch as the artist or designer. By this terminology we don't intend to limit our discussion to professional artists and designers, but instead in the same way that the person who writes a piece of text is the writer, no matter how many or few novels they have published, this is simply the most natural manner of addressing the subject whose designs or artistic thinking process are reflected in the sketch.

Sketching, whether by professional or amateur, is primarily used as initial or preparatory work for something else. The work derived from one or more sketches is usually more detailed, has a more finished style, and is more involved to create. It's because of the difficulty and time required to create that derived work that sketching is used to help the artist, designer, or whomever may be producing the sketch, to plan things out before major changes or modifications become difficult or impossible. By the same token, sketches are also widely used to help generate ideas. By laying plans out on paper, the artist is forced to solidify and confront each aspect of a design as it's depicted, a process that forces the brain to carefully consider each aspect of the design and how they work together.

Because they are used as preliminary work, sketches tend to look messy and disorganized. Although a sketch may take some time to produce, each stroke is usually created quickly and dynamically rather than as the result of a careful thought process. An artist simply lays down stroke after stroke, mark after mark, making decisions as he or she works rather than planning out each step before it's made. In essence, the sketch is a visual depiction of that planning process, and so

contains mistakes, revisions, experimentation, and usually a fair amount of noise. Most sketches are not designed or intended for public consumption, but are for the artist's benefit in the same way a writer makes a rough draft. This rough, *sketchy* nature is the primary characteristic of a sketch. Some typical examples of sketching are presented in Figure 2.1.



**Figure 2.1.** Several examples of sketching. This first is a sketch created by Leonardo da Vinci in 1481 in preparation for his later painting *The Adoration of the Magi*. This sketch is part of the permanent collection of the Uffizi Gallery, Florence Italy. Notice how the artist has included a grid of perspective lines to guide the placement of objects within the scene. Also notice how some elements, such as the stairs and arches on the left, are well defined with dark lines, whereas many figures appear only as light ghostly impressions. (continued on page 15...)



**Figure 2.1.** (continued from page 14) A series of sketches of heads and faces for character development. Notice that the artist begins with several basic head shapes into which a scaffolding of construction lines are drawn to help place the facial features. Some drawings are basic or incomplete, whereas others have been developed further. (continued on page 16...)



**Figure 2.1.** (continued from page 15) Rather than a cohesive sketch guided by a particular interest or composition, this work is a page of random doodles. We can see how the artist generates random shapes or groupings searching for something interesting. In some cases these doodles are dead ends, but in others the artist finds something to play with.



### 2.1.1 Sketching as a Design Tool

Sketching is in widespread use today by professional artists and designers, but also by people from any number of other backgrounds. However, sketching is most closely associated with the fine arts, and with good reason. Art and design instructors encourage their students to begin new pieces with one or more sketches to help the students visualize and develop an interesting or effective composition. Drawing classes from beginner to advanced levels will often spend days or weeks at a time producing nothing but sketches to give the students practice observing and exploring a model or still life. Art instructors exhort their students to carry this practice into their every day lives, and many if not most art students maintain sketchbooks into which they scratch, doodle, scribble, draw, plan, write, and otherwise record and develop their ideas. Art students not only sketch out plans and preparatory work for specific projects, but use their sketchbooks as a sort of ‘catch-all’ for interesting ideas, inspiring images, practice work, and anything else that sparks their artistic sensibilities. Professional and student artists alike will also sketch scenes, figures, images, or objects into their sketchbooks to serve as a general reference for later perusal. An artist might, for example, spend the afternoon sitting in a park, sketching the children at play. Then at a later date when the artist wishes to incorporate a child into a work, he or she can refer back to the sketchbook for ideas and inspirations. For many artists this practice becomes a habit and continues into their professional careers.

Similar behavior can be seen in other visual fields such as industrial design and architecture [Do, 2005]. Although the final designs created by these students or professionals are sometimes far removed from the 2-dimensional works produced by visual artists, they still begin their design process with sketches. In fact, even

though a large part of design work is now done at the computer terminal, sketches are still an integral part of the process. Diehl *et al.* cite one researcher who surveyed Computer Aided Design (CAD) designers, asking how many use sketches in their work [Diehl *et al.*, 2004]. More than half stated that they use sketches as preparation for their computer work, and 35% responded that they continued to use sketch in parallel with the computer system. In his 1993 book examining so-called ‘study drawings’ produced by architects, Daniel Herbert goes so far as to say that such drawings are “the designer’s principle means of thinking” [Herbert, 1993]. These sentiments are widely shared inside and outside the creative community.<sup>1</sup>

Similar practices carry over into other professions, especially those in which there is a focus on abstract or complex ideas that are difficult to visualize. Engineers, physicists, and scientists of almost any description are well acquainted with sketching out ideas or diagrams on whiteboards. It is common practice, for example, to work through a problem with a colleague by sketching diagrams and equations in tandem. Although the content may be different, the purpose of these sketches is the same.

Designers, architects, scientists, and artists are, at least in most cases, highly trained professionals. They spend many years practicing their craft, honing their skills, and learning techniques like sketching to help the develop their ideas. However the sketching skills they learn are not some mysterious alchemy but are simply a more formal and specialized incarnation of something many if not most people do instinctually. Back-of-the-envelope calculations and cocktail-napkin diagrams are just two colorful examples of sketching that are familiar to everyone.

---

<sup>1</sup>Ellen Do in her 2005 article presents a compendious overview of references for literature in this area [Do, 2005].

### 2.1.2 Sketching from the Observer's Point of View

Although sketching is primarily a personal activity undertaken solely for the artist's benefit, sketches can also have value to a limited audience of viewers apart from the artist. For example, the sketches produced by famous painters and other artists, or the drawings produced by designers or architects are of great value to the academic community. In her background research in developing an architectural assistant program, Ellen Do reviewed preliminary sketches by famous architects like I. M. Pei, tracing the evolution of Pei's building designs from their earliest concepts to the finished structures [Do, 2005]. Similar work has been done to examine the preliminary sketches of master painters. Art historians can use infrared reflectograms and x-ray photography to reveal the sketch lines underneath layers of oil paint or other media—see Figure 2.2. Indeed in the current art market, simple sketches and early drawings by master painters command prices commensurate with completed works. As we will see, sketching is an important part of the creative process, and so the sketches produced as part of that process shed light on the way the artist, designer, or architect is thinking.

Whereas the sketches produced by painters were probably not intended for public viewing, there are cases when an artist will create sketches with the intention of sharing them with others. This is often the case in the design and graphic arts community. Creating and rendering an entire image to a finished product is a time consuming and expensive process, time and energy that could be wasted in a commercial environment if the image is produced for a client and then the client rejects the work for some reason. To avoid this problem, and to foster communications between the designers and a client, graphic artists will often produce



**Figure 2.2.** (left) Vincent van Gogh’s *The Zouave*, 1888—reed pen, pen, and ink over graphite on wove paper from the Solomon R. Guggenheim Museum, New York. (right) An infrared reflectogram of the same work, showing van Gogh’s original hidden graphite sketch underneath the ink. [Ives & Stein, 2005]

preliminary sketches that are presented to other design team members and eventually the client for review and critique. This process is also a familiar one to an architectural firm, which must produce many preliminary drawings and designs before a full plan is complete. Once a design is approved and finalized, only then will the artist or designer invest the time and effort to produce a final quality version.

Aside from the obvious speed with which the sketches can be produced, the visual style of the sketches function as a sort of non-verbal communication, above and beyond the visual depiction in the image. Because sketches have a rough and unfinished look, they communicate to the viewer—in this case the client—

that the work is open to further revision, something that a clean and professional drawing does not necessarily communicate. As Strothotte *et al.* point out, a clean, well-defined drawing also has a sense of completion that might give the client the mistaken impression that a project has progressed further than may be the case [Strothotte *et al.*, 1994]. The article goes on to mention that sketched images have a certain dynamic and lively quality that can make them more appealing. The authors offers the example of architects, who will go so far as to print hard copies of their preliminary designs out of the CAD systems used to create them and then trace over the designs in pencil to give them a sketchy look before presentation to clients.

Strothotte *et al.* describe this additional feeling—one communicated by the quality of a rendering above and beyond the image it depicts—as *transputed information*, because it is transmitted as visual information, but it must be computed by the viewer through some reasoning process. Not only does the overall gestalt of a sketch communicate this transputed information, but the artist can also make local adjustments to convey even more information. By varying the quality of the sketch marks, or the detail of different portions of the image, the artist can lead the viewer’s eyes and emphasize some elements of the image while de-emphasizing others. This property is not unique to sketching, but because the spectrum of detail and mark making in a single sketch can be so wide, sketching is uniquely suited to express information in this way.

The use of sketching in the early stages of a design, as in the example above, has another benefit. Pereira *et al.* in discussing industrial design point out that early changes in a design can have large effects on later stages of the design process [Pereira *et al.*, 2001]. In the same way that sketching can function as a planning

tool for the individual, sketches can be used by a group of designers, artists, scientist, or laypeople to coordinate their efforts and avoid headaches later. If the old adage goes “*a stitch in time saves nine*”, then we might remark “*a sketch before a stitch means a job without a hitch*”.

### **2.1.3 Sketching as a Thinking Process**

We have made reference several times to the deep connection between sketching and the creative process. Many people have the impression that truly creative people are somehow struck from the blue with their brilliant insights, but in actuality inspiration is only a tiny step in the direction of creating new ideas.

How many times has this happened to you? You’re going about your daily business, maybe sitting in your car listening to the radio, or flipping through a magazine in a waiting room and you happen to catch a glimpse of something, or maybe focus on a conversation for just an instant. Whatever it was, it enters your head and starts to mingle with something else, combining and tuning over and mixing and morphing, and all of a sudden you think, “*Hey, that’s a pretty good idea!*” Sometimes you simply dismiss it and move on with your day, or worse yet commit yourself to jotting it down only to forget. If you’re lucky it may come back to you, or even begin nagging at the back of your mind until you finally do something about it. On the other hand you may frantically dive into your bag, digging for a pen and scrap of paper so you can record it before it slips away.

For the most part, these brief moments of mental clarity may be invigorating or even inspirational, but probably won’t pan out. We may get lucky on occasion and have one in our moment of need—say while staring at that blank screen waiting for our History report to spill forth—but very often these ephemeral sparks of

knowledge are just that, mere seeds that must be cultivated. The ideas may seem brilliant and innovative at the time, but upon further inspection they are in their most nascent form: not thought out, rough around the edges, incomplete, mere notions, concepts, or inklings of something larger.

The trick is to convert these basic concepts into a working model. The conversion process means working out the details: laying out a framework, fixing some specifics, thinking about tradeoffs and consequences, and fleshing out a working model until you have something you could explain to another person clearly. For a large project this could often entail working out many ideas; fleshing out one concept by creating and exploring others until you can form a web of connected concepts that hold together. The point is that, although the original idea may have gotten the ball rolling, much more work—both *creative* work and *work* work—is focused on fleshing out that initial concept.

This is where sketch steps gallantly into the picture. As Garner puts it “... [sketches] impose both order and tangibility on the one hand, while on the other hand their ambiguity stimulates reinterpretation” [Garner, 2001]. The sketching process, by its very nature, facilitates the mental process necessary to flesh out an idea into something bigger and more concrete. Strangely enough, we do this by utilizing our poor drawing skills. When you grab for some scrap paper and a pencil, your intention may be to somehow transcribe your brilliant insights directly from your brain to the page, but no matter your skills as an artist, this translation from mental image to paper sketch is necessarily a lossy one. A picture may be worth a thousand words, but a mental picture speaks volumes and there’s simply no way to accurately capture that image in full fidelity. It may seem like a losing battle between the artist and his or her tools but in fact it’s an

integral part of the creative process. As you try to express your ideas on paper you're forced to experiment, try things, make mistakes, or simply include a good deal of ambiguity. As you work you are constantly comparing your drawing with the mental image to check that things are coming out as planned. However, at the same time the perceptual part of your brain begins butting-in, reinterpreting the drawing and creating new ideas from that raw input. It's this back and forth between the thinking and drawing process that makes sketching such an effective tool. We'll explore this process in greater detail in the next section.

## 2.2 The Link Between Sketching and Creative Thought

Given the important role sketching plays in many creative fields, and the close association between sketching and the thinking process, it's not surprising that the practice of sketching has been subjected to its fair share of scientific scrutiny.<sup>2</sup> Although this research points to the links between sketching and creativity, on some level understanding the basis of the connection is as challenging as understanding the thinking process itself. Research in fields like cognitive science, neuropsychology, computing, and design continue to develop the problem [Garner, 2001], but a true understanding of why sketch is related to creativity remains beyond the horizon. Still, through observational and parallel studies—examining how sketching is used, observing sketches being made, and following the genesis of an idea or concept through sketches—researchers can at least theorize about *how* sketching feeds the creative process, even though they may not understand the *why*. One such theory was outlined by Do *et al.* in their 1996 article. In their publication

---

<sup>2</sup>A full account of current research is beyond the scope of this work; for the interested reader, Do *et al.* give a good sampling of literature on sketching in general [Do & Gross, 1996] and within the architecture and design domain [Do, 2005]. Garner also lists several papers in the area [Garner, 2001].



the authors attempt to lay out the basic activates of the design process, and to show how the act of sketching supports them [Do & Gross, 1996].

Although Do *et al.*'s work is focused on the design domain, we can probably think of analogous activity in any creative field and generalize the concept of the creative mental process reinforced by the physical drawing activity. We can think of the activity of sketching as composed of three related components. The first is the mental component of sketching, the cognitive process that is going on as you create and refine new ideas. The second aspect is your physical response to that thought process. In other words, how you change and adapt your drawing in response to the mental activity. We will call the mental component of sketching *feedback*, because it resembles a feedback cycle of interpretation and re-interpretation. The process of feedback is then driven by an activity we will call *incremental refinement*, in which the artist makes a series of adjustments to the sketch in response to the feedback cycle, a term we borrow from related literature on the subject [Gross & Do, 1996] [Michalik *et al.*, 2002]. The physical technique used to make these adjustments we call *overdrawing*.

### **2.2.1 Sketching as a Mental Process—*Feedback***

Feedback relies on the system of perception that our brains use to extract patterns and meaning out of what we see. Exactly how perception works, and even what perception is delves into deep questions of neurobiology, philosophy, epistemology, and metaphysics that have been topics of intensive debate for centuries. Even the physical processes behind vision and thought are mostly a mystery. As John R. Searle famously remarked, it's neuroscience's "dirty little secret" that the mechanisms for computation and representation in the brain are still unknown

[Searle, 1997]. However scientists are beginning to work out some of the processes involved, even if they do not fully understand how or why they work that way.

Probably the most popular model of perception is based on the *neuron doctrine* [Sabbatini, 2003], which holds that neurons are the basic units of the nervous system and function as connected but independent units, like a network of processing nodes. Based on this concept of neuroscience Bertrand Russell used what he called the causal chain of vision to explain perception and the connection between what we see with our eyes and what we perceive in our minds [Russell, 1927]. The theory works something like this: When you look at a picture of something, the visual image projected onto your retina is translated into nerve impulses by structures in your eye and passed through your optic nerve on to the visual cortex, a region in the back of your brain in what is called the occipital lobe. The stimulus causes a pattern of activation within the network of neurons in your brain, and it's this activation that is the physical manifestation of perception.

Once your brain has a mental image of what you are looking at, the next task is to assign that image some meaning. As with perception, exactly how this is achieved is also a matter of debate, but most theories in this area center around an idea of pattern matching and emergence. The neurons in your brain act as feature detectors, which construct a mental model of what you see by comparing the visual stimulus with patterns of stimuli it has encountered before, trying to make connections between what it sees and what it knows [Lehar, 2004]. The brain latches on to specific, characteristic features of what it sees and uses the combination of those features along with basic knowledge of the world to make a tentative identification of the object and its characteristics. Thus, the more information your brain can receive, the more definite this identification can be.

The interesting thing is that, even in situations where there is a lack of stimulus, your brain attempts to make these connections between what it sees and what it knows. This is the reason why we can look up on a cloudy day and see shapes in the clouds, or why small children<sup>3</sup> believe they see monsters in a dark room. The brain is taking in far too little information to make a positive identification, so our imaginations kick in and fill in the rest. Although this process can make for a fun activity on a cloudy day, it also serves as one of the basic operations behind human creativity. As we've discussed, using this same system people infer connections between things that may not at first seem to be related, and these connections form the basis of those ephemeral ideas that pop into our heads.

Do *et al.* in describing the activities used by a designer, discuss several similar activities [Do & Gross, 1996]. The authors for example explain that designers often use analogy to solve design problems, drawing on concepts from other areas and refitting them to the present situation. They also highlight abstraction, where by the designer steps back from the details of something to a simpler view in order to see broader patterns or configurations in the larger design. These are both examples of reinterpreting a work to fuel the feedback process.

This same process of interpretation occurs as you are sketching. Because sketching is largely an incremental process, for much of the time you work, the picture does not fully describe the concept it's meant to. Especially in the early stages, the sketch may bear little or no resemblance to anything in particular. Just as the mind seeks to make connection in the random swirls of clouds, the same process is going on as you look at your sketch. Your mind is in a continual loop of picturing an idea, trying to render the idea with strokes on the page,

---

<sup>3</sup>And some not so small.

seeing the strokes, reinterpreting them, and comparing the strokes to the mental image. When portions of the sketch are ill-defined, your mind begins to make those same imaginative connections out of the existing lines. You begin to recognize interesting shapes or ideas in the drawing, and perhaps use those to augment or solidify their mental image. You then in turn enhance those elements that suggested the idea in the first place. This process continues until you are satisfied that the sketch contains enough detail to convey the desired message or concept, at which point the sketch is finished.

This cyclical pattern of interpretation and re-interpretation is not unique to sketching, Durand makes reference to a similar optimization process in the context of 3-dimensional modeling [Durand, 2002], however sketching seems to be particularly well suited to the task.

So, by harnessing the opposing activities of creating visual stimulus, and interpreting visual stimulus, sketching creates a feedback loop that drives the creative process. Most of the time our brains interpret our scratches and doodles for what they are, mistakes or distractions, and we erase or ignore them, but sometimes these happy accidents<sup>4</sup> become the seeds of new ideas or interpretations that the designer not only leaves in the drawing, but builds upon to help flesh out part of the design that is still fuzzy or ill-conceived.

### **2.2.2 The Activity of Sketching—*Incremental Refinement***

Whereas feedback accounts for the cognitive component of sketching, the activity of sketching is characterized by incremental refinement. Incremental refine-

---

<sup>4</sup>The term “happy accident” in relation to art was popularized in part by Bob Ross, host of a long running oil painting instruction program that aired on the American Public Broadcasting System in the mid 1990’s.

ment can be summed up by a maxim familiar to any art instructor: “*work from the macroscopic to the microscopic*”. This apophthegm simply states that the artist works from the general shape and form of their subject—its macroscopic properties—down towards the microscopic details.

During sketching incremental refinement occurs on multiple levels. The first, and most basic level is tightly bound to the feedback process. This is where the artist makes incremental changes to the drawing to reinforce emerging patterns identified by feedback. As Do *et al.* point out, this kind of refinement is characterized by repetitive redrawing and over-sketching [Do & Gross, 1996]. The authors observed that this practice serves to mentally select portions of the drawing in the artist’s mind, highlighting the shapes that he or she wishes to bring forward and de-emphasizing others.

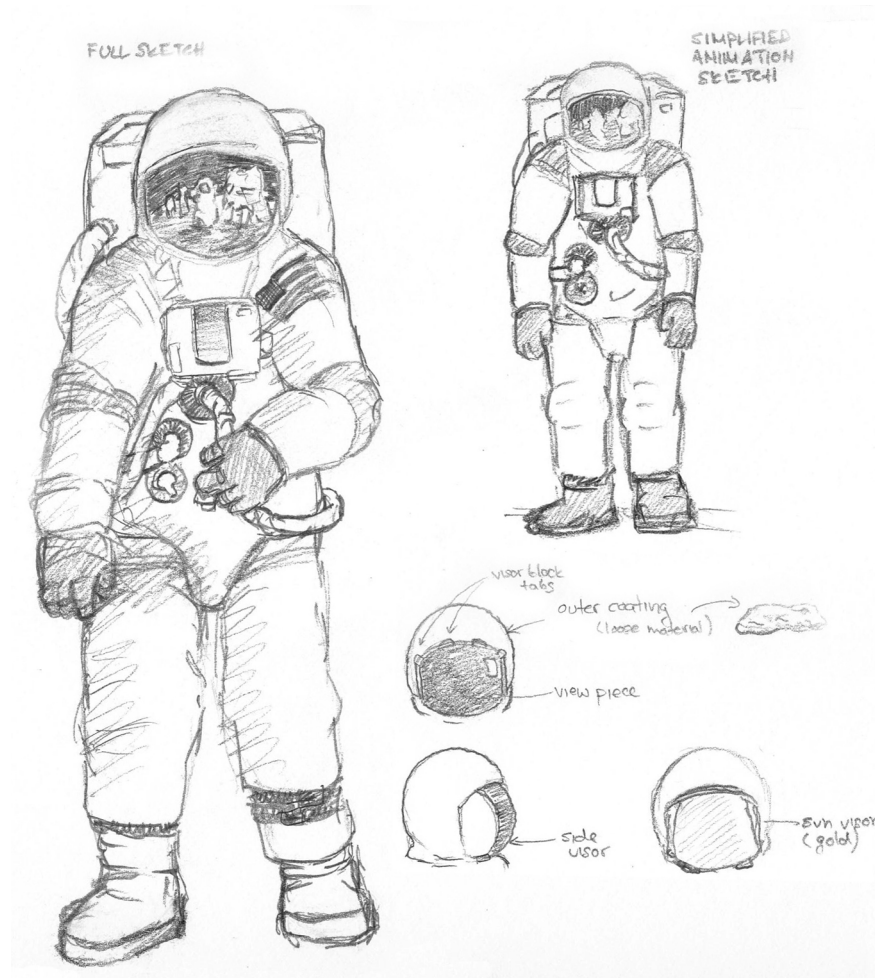
At a higher level, incremental refinement is tied to the process of producing a sketch from start to finish. Here the artist begins by defining large, abstract shapes, and then through recognition, replacement, refinement, and transformation [Do & Gross, 1996], the finished form emerges as necessary details are developed.

### **2.2.3 Sketching Technique—*Overdrawing***

As the artists—be they professional artist or designer designers, or scientist, or bored calculus student for that matter—works on a sketch, they may use their drawing tool in many different ways to produce a wide variety of different marks. Appropriately enough, artists refer to the variety and character of strokes in an artwork as *mark-making*. Although the artist may use many different kinds of mark-making, sketching is characterized by a specific technique called *overdrawing*.

Overdrawing goes by many names in art, design, and modeling literature including oversketching [Karpenko *et al.*, 2002] [Zelevnik *et al.*, 1996] [Pereira *et al.*, 2003], re-sketching, re-drawing [Do & Gross, 1996], overtracing [Do & Gross, 1996], scribbling, ‘nervous’ hand [Henzen *et al.*, 2005], and even the vaguely alluring pen-timenti [Alex, 2005]. No matter what it’s called, all of these terms at least allude to the same basic practice in which the artist places marks directly over existing strokes in the drawing. As the artist works these repeated lines begin to build in some areas of the drawing, making the lines they reiterate darker and more defined than others. Although the artist is redrawing some lines not all of this mark-making is to repeat the same paths verbatim. More often than not, the artist also makes minor adjustments in each stroke’s path and character. As subsequent overdrawing strokes are added these embellishments are further defined and darkened, emphasizing them over the historic path of the line. An example of overdrawing can be seen in Figure 2.3.

Do *et al.* describe overdrawing as having three functions to the artist [Do & Gross, 1996]. The first is to serve as a means of selecting a portion of the drawing in the artist’s mind. By continually insisting upon certain lines or shapes, the artist forces him or herself to consider those elements they feel are important, and to see them in relation to other parts of the drawing. In this case, the artist’s overdrawing strokes follow the path of the original lines, only making them darker. Artists also use overdrawing to adjust and refine the shapes in the image. In this case, the artist’s overdrawing strokes follow most of the underlying line, but may veer off in places or add fine details and embellishments to what was before a simple path. In this way the overdrawing technique is used to perform incremental refinement. Finally, artists use overdrawing to extract and make concrete emergent patterns



**Figure 2.3.** Preparatory sketches for an animated astronaut. In this sketch we can see several examples where the artist has created a general shape, and then honed that shape down using repeated overdrawing strokes. Looking at the far left suit for example, the astronaut's boots are composed of messy, over extended lines that blend together to create a boot shape. Similarly, we can see a great deal of variation in the astronaut's silhouette. In some areas the outline is faint and composed of only a few lines, whereas in others the line is very dark and over-defined. This is evidence that these darker areas of the outline have been corrected and overdrawn to a greater degree than the lighter areas.

they perceive in the tangle of lines making up an unfinished work. The shapes the artist chooses to emphasize are recognized by the feedback process, selected, extracted, adjusted, and refined, all by overdrawing.

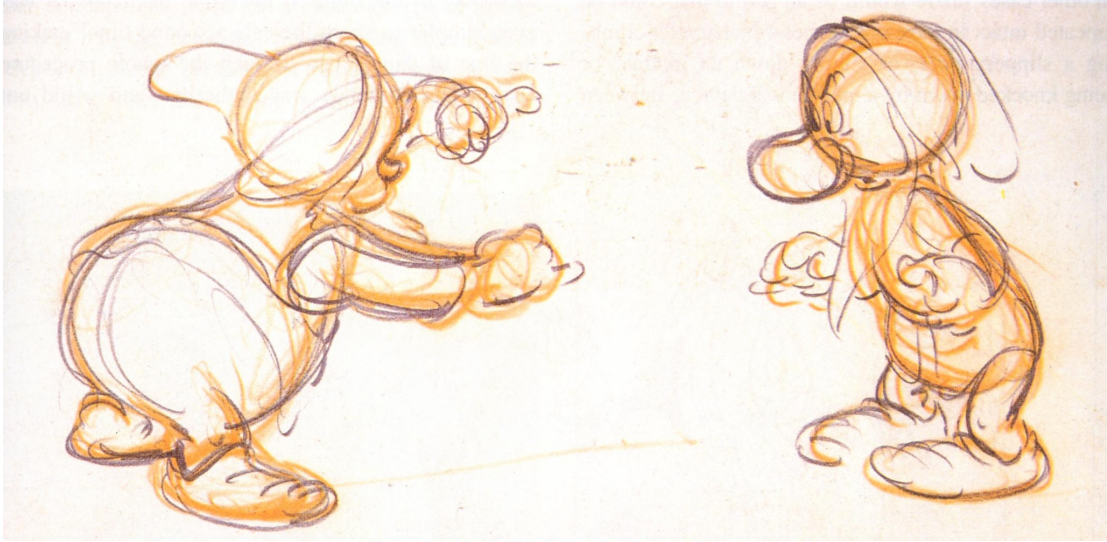
Looking at a completed sketch it's often easy to see the messy character that overdrawing created. A close examination of the built up lines and their lighter and darker components reveals a history of the creation and adjustment of that line. Overall, the heaviest aggregation of lines along certain paths forms a *line hypothesis* [Henzen *et al.*, 2005] that suggests the path of a single, smooth line that follows the dark core of the other lines. Depending on the purpose of the sketch, the artist may simply leave the lines the way they are. However, when used as a rough prototype for more exacting work the artist may wish to erase some or all of the overdrawing noise, or simply trace clean lines onto another paper. In animation for example, a designated clean-up artist performs this task. It is his or her job to interpret the path of this clean line hypothesis from the original animator's overdrawn sketches [Henzen *et al.*, 2005]—see Figure 2.4.

These three concepts: *feedback*, *incremental refinement*, and *overdrawing*, are emblematic of all sketching.

## 2.3 Sketching by Hand

Although we have been speaking in general terms, most of the discussion of sketching up to this point has focused on what we might call 'traditional sketching' or 'sketching by hand' in which the sketch is created with paper and drawing implement. This term is not a formal classification, but a convenient invention. We use it here to refer to any and all sketching activity that is not





**Figure 2.4.** Rough sketches of two of the seven dwarfs from Walt Disney's Snow White and the Seven Dwarfs (1937) by animator Fred Moore [Johnston & Thomas, 1995]. These rough drawings allowed the animator to feel out more vibrant and lifelike motions. The drawings would later be cleaned up and refined by a clean-up artist.

associated with a computer or other electronic means. This accommodating dichotomy will make it easier to compare the two activities.

As we have already seen, sketching in general has numerous benefits to creative professionals and laypeople of all stripes. Of course it's only in the past 40 years that we can even speak of something other than traditional sketching [Sutherland, 1963]. Although sketching provides many benefits, a large part of its success comes not from what sketching can help an artist do, but how easy it is to do it. Even if sketching was manifold more useful, it's unlikely that any but the most dedicated would spend time and energy sketching if it required hundreds of dollars of supplies, years of careful study, a large and dedicated workspace with specialized and expensive machinery, or an awkward wardrobe. Thankfully, although such accoutrements are no hindrance to sketching,<sup>5</sup> the activity itself

---

<sup>5</sup>An artist's beret is particularly effective.

requires no specialized equipment other than a pen or pencil and a suitable writing surface and can be done almost anywhere. What is more, sketching is practically an innate ability, a simple extension of childhood drawing, and requires no special training whatsoever. Best of all, sketching is not only useful, but can be a lot of fun too.

Let's take a look at exactly what traditional sketching entails. We will examine the process of sketching and how this process has been studied, and we will take a few words to illustrate where traditional sketching has its limitations, and how computer-based sketching is stepping in to address those issues.

### **2.3.1 The Sketching Process**

If you were to peruse the art books at your local library or bookstore you would find no shortage of instruction guides and technique books describing how to sketch. However, although some disciplines like animation have developed their own formalized methods for sketching, in general there is no right or wrong way to sketch. Sketching is an intensely personal activity, and although tips and tricks can be taught, the overall process is something that most people simply come to on their own, adapting their innate and rudimentary drawing skills from childhood into something they find useful. Several observational studies have revealed however that there are common patterns and methods to sketching used by artists and designers in different fields [Do, 2005] [Huot *et al.*, 2003] [Igarashi *et al.*, 1997a]. These studies suggest that most sketching activities can be divided into a number of related and overlapping stages, each characterized by the kinds of strokes the artist makes, and the role those strokes play in the final drawing.

The first study, which we have already briefly touched upon, was conducted

by Do *et al.* and looked at the sketching activities of architectural students and professionals. Ellen Do and her associates actually conducted three studies as background for the development of a designer's assistant application [Do, 2005]. The first, an observational study, involved sixty-two undergraduate design students who were asked to produce and interpret 'design diagrams', described as drawings composed of geometric elements that abstractly represent both architectural objects, and natural phenomena such as light and sound. Analyzing the results of the study, Do found that interpretations of the diagrams were consistent among the students, and that the students tended to use a limited set of symbols to represent different design concepts, occasionally augmenting them with textual labels and annotations. From these findings Do concluded that within the architectural and design domain, there is a conventional and consistent vocabulary of diagrams.

In a second study, two architecture instructors and two senior design students were presented with an architectural design case involving the design of a corporate office. The participants were asked to develop solutions to several typical design problems such as dividing the floor plan into workspaces or incorporating a specific piece of furniture. The intention of this study was to examine design sketches as opposed to diagrams as in the previous study. In examining the results, Do reported many of the same patterns of consistent use of symbols and shapes to represent objects like furniture or walls. Do also noted that when tackling specific problem, the designers chose to portray the situation in similar views. In the paper Do specifically notes that the students and instructors followed the same convention of depicting lighting concerns from a sectional viewpoint—that is, looking from the side rather than a plan view from above. Students in the

first study exhibited a similar preference for sectional or plan views for different design tasks using diagrams. Do also found that the subjects use labels and over-tracing to emphasize portions of their sketches, and that they often performed small calculations along side elements to help determine dimensions of objects or spaces.

In a final study Do conducted a survey of architectural and design drawings from famous architects like I. M. Pei, Mies van der Rohe, and others. Do found many of the same stylistic conventions that were observed in the work of the students and design instructors. By examining the progression of drawings over the course of a design, Do was also able to identify the processes and transformations that characterize the problem solving process used by the architects.

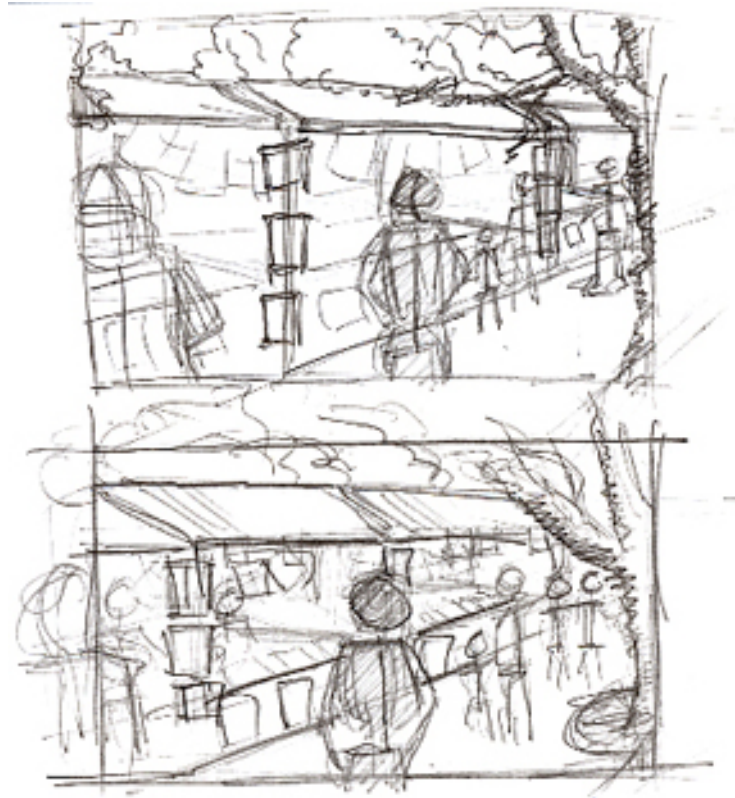
A more formal classification of the sketching process was outlined by Huot *et al.* [Huot *et al.*, 2003]. Their paper describes an observational study of architectural students of various levels, as well as four professional architects. The study participants were asked to dream up and sketch a building viewed from its exterior. As the subjects worked, they used a standard pen affixed to a digitizing tablet that allowed the subject to draw in a familiar manner while the researchers were able to record each stroke produced in a computer. After the task was complete, the researchers examined the kinds of strokes that were used by the participants, and what periods of time during the sketching task each type of stroke was used.

The researchers were able to classify the subject's input strokes into a taxonomy of seven types of strokes describing such things as perspective and guide lines, principle strokes defining the building's shape, and stylistic and improvement lines. Once the taxonomy was developed the researchers were able to tag

each line recorded during the experimental sessions. They then analyzed this information, and extracted the temporal distribution of strokes during the drawing process. The researchers' findings revealed recurring patterns among the study participants indicating distinct *stages* within the overall drawing process in which one type of stroke creation or another was most prominent. They found, for example, that lines used for guides or other constructive purposes are made almost exclusively within the first 10% of the drawing time. This observation was part of a larger progression from a “creative context” early in the drawing process when the majority of large-scale forms are laid out and made concrete, to later stages characterized by improvement or stylistic elements. The researchers were able to recognize similar stages of development among the participants, and even observed indications of divisions between the stages expressed as longer pauses between transitions in stroke type. This progression is an example of the “macro to micro” characteristic of incremental refinement.

### **2.3.1.1 Kinds of Sketching Lines**

Although not directly based on observational studies, many authors make reference to sketching as a progressive activity, similar to our concept of incremental refinement. They also cite the close connection between the stages of a sketching process, and the mark making used in that stage. Cherlin *et al.* for example describes the progression from so-called ‘constructive curves’, which define the shape and mass of the subject, to the progressive addition of external and internal details to further refine the drawing [Cherlin *et al.*, 2005]. Cherlin goes on to describe three types of strokes used in that refinement process, which form the basis of his sketch based modeling system.

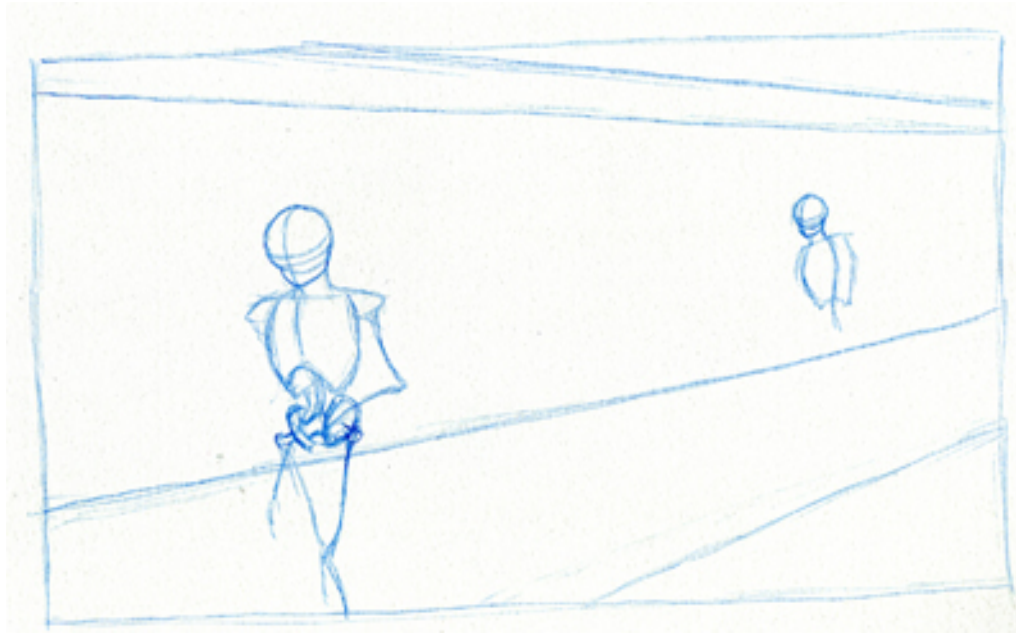


**Figure 2.5.** In this progression of images we can see the use of various types of sketching lines though the process of creating a drawing. This first image is a series of two thumbnail drawings, used by the artist to try out different compositional arrangements. Following major structural lines in the drawing we can see a number of perspective constructive strokes. Within the image these have been used as scaffolding to build the architectural components. We can also see rough symbolic stick figures generated from skeletal frameworks and scribbled outlines. Definite elements of the drawing have been overdrawn to create dark emphasized lines while minor details are only suggested through lighter strokes. (continued on page 40...)

Although we don't have experimental data from which to develop a similar classification for more general sketching, from the focused work by Do and Huot, and a healthy dose of self examination we can lay down at least a basic taxonomy of lines. Examples of the sketching process are demonstrated in Figure 2.3.1.1.

**I. Construction and Initial Lines** The artist generally starts out by drawing large, general shapes that form the major masses or general bounding regions of the subject. When the work is done in pencil these initial lines are drawn very lightly so that they can be refined and erased later. They are also often drawn with the largest and most dynamic strokes. By making single, broad, sweeping strokes the artist attempts to create smooth and aesthetic base shapes to define the form. Often the artist will be seen making several practice strokes above the paper before actually creating the stroke so as to get the proper feeling for the lines, something like a golfer practicing her putt before taking an actual swing. This practice is especially prevalent among cartoonists and professional animators who will often base their characters on one or more characteristic primitive body shapes—ovals, triangles, oblong rectangles, etc.—which are later refined into the character.

**II. Guides and Annotations** Not all of the lines created by the artist are intended as part of the drawing. Many artist draw extra marks that help them to place objects or flesh out the 3-dimensional shape of their subject before all of the details have been put into place. Feature guides drawn onto a cartoon character are one common example. A cartoonist or animator will draw lines following the curves of the 3-D form to assist them in correctly sizing and placing things like facial features. Another common example are perspective guides. Many artist working in strict one- or two-point perspective views will draw several lines extending from the vanishing points of the image from which they can estimate the angles of subsequent lines defining aligned geometry.

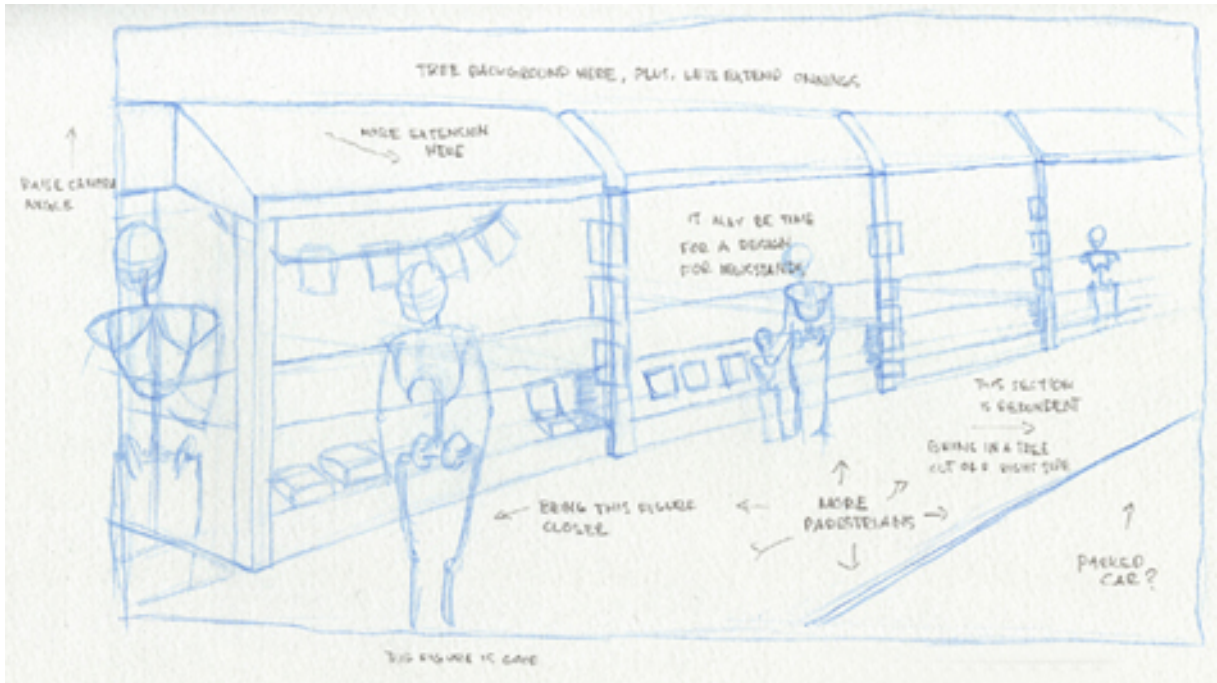


**Figure 2.5.** (continued from page 38) With a general idea of the composition, the artist begins a full sketch with basic construction lines to layout major planes and surfaces in the image. Here the artist has also started to work out the positions of figures. (continued on page 41...)

Artists also make small notations, symbols, and other markings that serve as reminders or thinking aids as they work. Many artists for example will draw a small arrow or sun shape to indicate the direction of a light source [Do, 2005]. They might also jot down small notes, perhaps describing the correct colors to use for a region of the drawing, or indicating what some, possibly ambiguous shape is so that they will know later on. Finally the artist may include roughly defined regions or very basic geometric shapes, which are intended as placeholders or indications of how the main subject of the drawing fits into a larger context [Do, 2005].

Because guide strokes are used to help the artist place other elements of the drawing they are usually produced in the early stages [Huot *et al.*, 2003]. Annotations on the other hand may be produced at any stage. Lines of these





**Figure 2.5.** (continued from page 40) As the composition progresses, the major elements of the drawing are laid out using more refined constructive lines. We can see that in some areas the artist has adjusted the constructive lines of the figures and newsstand to find the correct shapes and alignments. In this image we can also see annotations added by the artist to notate changes that will be needed in the next draft of the drawing. (continued on page 44...)

types are not part of the geometry, but they hold a great deal of meaning for the artist, and play an important role in the thinking process of sketching. They are generally drawn very lightly, and little care is given to making them neat or well defined. Although they may not be erased, they are still drawn in a diminutive style so as not to detract from the focal points of the drawing. These elements will probably not be included in the final work, or will be replaced by external elements created specifically for the purpose.

### III. Extractions and Corrections

Once the artist has defined one or two ma-

For shapes, the feedback process kicks in. Especially in these early stages large and drastic changes can be made to the sketch. These take the form of coarse-grained redrawing. The artist simply draws the shape again, right on top of the old shape. This technique is used when the entire position or shape of the current stroke needs to be changed. Usually the old strokes are left where they are rather than being erased. This serves both as a guide to the new stroke, and as a means of comparison when the new stroke is finished. The artist may very well determine that the old stroke was better, or that portions of each stroke have their merits.

Because these are very early strokes and will be subject to a great deal of alteration, these too are drawn lightly so that they can be later erased. If the artist decides to continue using the stroke then he or she will redefine it later by sketching over the exact same path, so there is no need to worry about its light appearance now. This allows the artist to select just the parts of the existing drawing that are good at a later date. In many drawings, nothing of these original strokes may survive to the final sketch.

Although the artist may wish to define the entire extent of an object in a single stroke, it is very difficult to maintain smooth lines for a long distance or over a long curve. This is because the human hand has a limited range of motion through which it is most expressive. If the contour of a stroke requires the artist to travel past these limits then the lines that result are harder to control. To see for yourself, try drawing a perfect circle about the size of a ping-pong ball on a piece of paper. You will find that unless you reposition your hand as you move around the paper, your circle is likely to come out rather distorted and wavy towards the edges.

In order to define these kinds of shapes an artist will often use several broad lines to form a general outline of a surface, and then come back in with shorter, more controlled connecting lines to form the intersections, joins, occlusion, and other interactions of the broader strokes. This allows the artist to use broad sweeping strokes for large, regular areas, and more defined strokes to connect them into a solid and seamless whole. It also allows the artist to reposition his or her hands, the tool, and even the drawing surface in order to get the best angle from which to draw each segment expressively. As more and more of the image is laid down, the sketching process becomes less concerned with correction of the overall shape, and more concerned with the creation or extraction of smaller details. This is primarily achieved with corrective over-sketching, which is similar to redrawing, but on a smaller scale. The artist will draw new lines directly over the path of an existing line, but change the course or character of the line in places. The correction marks may be along the entire extent of the existing line, but more likely they are over a small portion of a line, or across the intersections or near intersection of several existing lines. This is the heart of the incremental refinement aspect of sketching.

Over-sketching lines can be dark or light, and can be made quickly or deliberately. Most lines will be over-drawn multiple times throughout a sketch as the artist refines and re-refines the image to its final form. It is the appearance of overdrawing lines that principally exemplifies a sketch from a clean line drawing

**IV. Additions** Additions are the creation of new pieces of the model. The cre-



**Figure 2.5.** (continued from page 41) Here the artist has completed sketching in the background elements, but the depictions of figures are still represented by detailed skeletal frameworks. By laying down guidelines on these frameworks, the artist will be able to draw facial features, hair, and clothing more accurately. (continued on page 45. . .)

ation of these new segments follows the same patterns of incremental refinement and feedback discussed above. By developing a number of additions in different locations within a single sketch, and artist is in effect creating a hierarchy of elements differentiated by their level of detail and stage of completion. Although each element may be completed eventually, in some cases portions of the drawing are left in a less refined state. Artists use the variations in the level of detail over different parts of the sketch as a means to steer the viewer's attention to some areas of the drawing and to downplay the importance of other elements, which may only serve to set the scene or put the main subject into context [Strothotte *et al.*, 1994].



**Figure 2.5.** (

continued from page 44) Now that the final composition has been settled on, the artist goes to work constructing actual renderings over the skeletal frameworks of the earlier sketches. Once again we can see that the artist has focused a great deal of effort on the figures by their relative darkness compared to other elements of the drawing. (continued on page 47...)

**V. Completions and Embellishments** At the completion stage most of the actual forms in the sketch have been fully defined, as well as many of the finer details. However the individual pieces may not have a constant or uniform level of detail. In this stage the artist moves over the work emphasizing the final positions of lines by over-sketching them more darkly. The artist may also choose to remove some of the construction lines and remaining over-sketching lines to give the drawing a cleaner look. It is also at this stage that the artist may choose to add embellishing stylistic details such as shading, texturing, or small environmental additions. These elements may not be strictly necessary, but they have several aesthetic benefits: focus-

ing attention on important components of the drawing while downplaying others, giving the work a personalized look, and encouraging the artist to thoroughly examine each part of the sketch to detect any areas that might be ambiguous.

Time spent in this stage of the process is dependent on the purpose of the sketch. If the artist's intentions were only to lay down a few ideas, or work out a framework for later refinement then he or she may forgo finishing steps all together. On the other hand, if this sketch is intended for presentation, then the artist will use the completion stage to refine the work to a point where it will make sense to others.

Finishing strokes formed at this stage are usually very deliberate. For lines that make up the primary structure of the sketch the artist may choose to use very dark and well defined lines, but strokes used to accentuate features like shading and textures can come in many shapes and sizes. The strokes produced at this stage are intended as a final product, so the artist's overarching concern is aesthetics.

The term sketching covers a wide range of activities from very informal doodles to fairly involved diagrams. The progression presented here is probably a little more formal and rigid than necessary. While watching an artist the careful observer may be able to recognize the stages discussed above, but to be fair the sketching process is a rather fluid one, and transitions between each stage may only be clear in retrospect. To be sure, artists are not diligently ticking off such progression in their heads as they work, but are instead working instinctually towards an end. This progression is a consequence of the creative process, rather



**Figure 2.5.** (continued from page 45) With the sketch complete, the artist has returned with a different media to clean up the sketch and complete the drawing. Replacing sketchy overdrawn lines and suggested details are cleaner and more definite strokes. The artist has also added a number of embellishments, including shading and texture for some objects in the scene.

than a prescription of its proper form.

### 2.3.2 Limitations of Sketching by Hand

Even for those who may not have been familiar with sketching, it should now be clear why the practice is so popular. By utilizing simple tools that are around us all the time, sketching offers a way to harness your brain's creative potential with a method that could not be simpler. Unlike other forms of art, there are no messy paints or expensive supplies to buy; all that's required is a scrap of paper and a pen or pencil. There is no need to stop there, chalkboard, whiteboards, sidewalk chalk, even a stick in the dust, sketching can literally be done anywhere.

Unfortunately, sketching is not without its disadvantages. Thankfully, most of them are simply a result of sketching's finer qualities, a reflection of the fact that after everything sketching is primarily a planning tool, and not intended to produce finished works. First there are the obvious limitations of the medium, which in most cases is constituted by paper and pen or pencil. Alas, despite creative efforts by young artists everywhere in exploring the space beyond their paper and onto the coffee table, most sketching is limited to the size of paper on which you start. This may seem like quibbling and caviling, but because sketching is an exploratory exercise, it can be difficult or impossible to know which direction your imagination may take you. Though creative use of scotch tape one can sometimes allay the issue, but it's always frustrating to realize that the otherwise masterful sketch of your neighbor's dog will crop off the top three-quarters of her head.

It's in fact situations just like this one that prompt art instructors to exhort their students to draw a series of thumbnail sketches before starting on a larger project. Because 2-dimensional artworks depict a fixed perspective on a scene, there is no way to change the view once work has begun. Therefore it's important for the artist to develop a composition that includes all of the desired elements—a process that takes some trial and error—before it is too late to move things around. Sketching, like any drawing technique, has the same drawback, though fortunately most sketches are inconsequential enough that they can be changed or simply redone without much fuss.

We noted above that one of sketch's major benefits was that special supplies are not required. Being able to sketch with just a handy pen or pencil is certainly a boon, but because the artist only has one tool to choose from, the level of ex-



pressiveness or contrast he or she can create is limited. As we have said sketches can be very messy drawing, which can make the intentions of the artist difficult to follow. Some artists and designers choose to sketch, for example, with colored pens, markers, or colored pencils to help them differentiate objects within a drawing. However collecting a wide variety of these materials can be expensive, and as the spectrum of available colors grows, the materials become either less portable, more difficult to clean up, or often both. Also as the variety of colors increases it gets harder and harder to make changes in the sketch using overdrawing. As more and more pigment is added to the paper, it's no longer possible to change the path of lines or extract shapes from the muddy noise. Furthermore erasing lines made with such mixed media is almost always out of the question.

Finally, we come to an issue stemming from recent changes in the creative landscape. At one time a painter, illustrator, or other artist might work through several thumbnails, compose a final sketch, and then begin working with paints or inks directly onto their preliminary work. For traditional media artists this is still the status quo, but much of the professional art and design community is now based around digital representations of art. Unfortunately, as other portions of the design pipeline have made their way onto the computer, sketching has been something of a holdout, and suitable software and hardware for sketching directly into the computer is not yet widespread. This means that sketch artists must produce their sketches as before on paper, and then find ways to transfer that work into a digital form. Thankfully, newer, faster computers, new input hardware, and new computer art packages are starting to bridge this gap. What's more, the computer offers the sketch artist solutions to most of the problems discussed here. This is not to say that computers will soon supplant traditional

sketching as we know it, but is rather a sign that new interface technology is making it possible to transfer old skills into the digital sphere.

## 2.4 Sketching with the Computer

Given the importance of art to our culture, and the large number of people who practice it or appreciate its practice, it should be no surprise that at some point a computer scientist would attempt to tackle the issue of computerized art. The first solid example of a computer art program was called SKETCHPAD. It was designed by Ivan Sutherland, a student at the Massachusetts Institute of Technology as part of his PhD thesis in 1963 [Sutherland, 1963]. Although it was literally the first of its kind, SKETCHPAD boasted a surprisingly full and impressive list of features, and many of the interface designs and interaction mechanisms first seen in SKETCHPAD are still used in CAD and drawing software today. SKETCHPAD was designed to run on MIT's Lincoln TX-2 computer, one of the first computers to use transistors rather than vacuum tubes built and housed at MIT's Lincoln Labs.

The TX-2 was designed for batch processing tasks like many other computers of its day, and a large part of Sutherland's work was in reconfiguring the hardware and software of the computer to allow it to function interactively. The program itself took input from a light pen connected to a CRT display. The user could draw shapes using lines and simple curves, and even specify geometric constraints between elements, which the computer would then adjust dynamically. SKETCHPAD's interface was a revolutionary departure from the way even computer scientists were accustomed to dealing with computers at the time. As Alan Kay noted in a video presentation for Apple Computer, SKETCHPAD rep-

resented the first example of a graphical user interface, the first example of object oriented programming concepts, and the first instance of a non-procedural program. These features prompted Kay to describe Sutherland's work as "probably the most significant thesis ever done" [Kay, 1987].

Since Sutherland's initial foray into mixing computers and art, an entire field of computer science concerned with computer graphics has taken shape. Computer scientists quickly realized that computers offer a means by which many of the limitations of traditional graphic techniques such as those we have discussed here, can be addressed. First, a computer system creates a separation between the physical activity of an input, and the resulting visual output. To the digital artist this means that the computer can take the place of an entire art store full of supplies and then some. Rather than making physical marks on a page or canvas, the user's strokes are like a metaphor connecting their physical actions to any one of an infinite list of pens, pencils, or brushes, each with an infinite rainbow of colors. Not only are the tools now digital, but the artwork is as well. The artist can make copy after copy without fear of degradation. Plus, as copies become trivial to produce, the penalty for experimentation disappears all together. The artist can change, move, adjust, distort, add to, or subtract from their work as much as they like, simply undoing any change that goes too far. Finally, the tools and materials of the digital artist, along with their work exist within a digital studio large enough to work on a drawing a mile on a side, but small enough to fit in a memory stick linked to the artist key chain.<sup>6</sup>

Computer graphics programs for everything from highly standardized CAD drawing systems to children's paint programs are now used far and wide, and just

---

<sup>6</sup>Best of all, the digital studio is never a mess to clean up.

like the modern print, film, and audio industries, most commercial art and design work now involves computers in some capacity. In this section we will examine the tools used by digital artists and designers. We will also look at the computer art and design programs available today, and see how the practice of sketching figures into this digital medium.

### **2.4.1 Computer Sketching Tools**

The appeal of computerized art is easy to see from the perspective of publishers and editors. Digital artwork is effortless to copy, easy to manipulate, and trivial to combine with other elements like print or film. However, no matter how beneficial computer art packages may be to some, if art software stifles an artist's creative process no amount of cajoling is likely to drive artists out of their comfort zone in traditional media to an awkward and uncertain future.

With the rise of the graphical user interface for most computer tasks the mouse became the de facto input device, and is now a standard component of any computer system. Although mice are well suited for tasks like menu and text selection, artists generally find mice at least distasteful and at most deplorable as drawing implements. In an article about the development of Corel's Painter John Derry described drawing with a mouse as something akin to drawing with a bar of soap [Derry, 1996]. Artistic technique—like playing a musical instrument—is partially a physical skill, and artists develop those skills using familiar tools. Just as a pianist would find it awkward to play a concerto on a computer keyboard rather than a musical one, most artists prefer to work with the familiar metaphor of a drawing instrument.

The first tool to replicate this form of interaction was the light pen, the tool

used by Sutherland's SKETCHPAD. A light pen resembles a standard pen, but rather than a writing tip one end contains a small light sensitive element. The pen is designed to work in tandem with a CRT display device. When the pen is held against the display it detects the sweep of the electron gun repainting the screen, and by signaling this event to the computer system the position of the pen's sensor in relation to the screen can be determined. Wikipedia points out that people are probably most familiar with the light pens used by contestants on the television game show Jeopardy to record their answers in the final round. Sports commentators use a similar system to draw simple figures over a video playback to aid in explanation of the action. Although the light pen offers a better approximation to drawing than a mouse, many users find them uncomfortable to use for more than a few minutes. This is because the light pen must be held against the vertical surface of the CRT screen, causing the blood to rush from the user's hand [Kay, 1987].

Several other systems that evoke standard writing and drawing utensils have been developed since the introduction of the light pen. Most notable is the touch sensitive screen. Unlike the light pen, touch screen can be used with LCD displays, and require only pressure to be activated, meaning that the stylus need not be electrically connected to the computer. User can use pens of any shape or size, or even their fingers to make marks. A more advanced technology is the digitizing tablet, which can be used to replicate a drawing or writing experience, while at the same time providing the computer system with far more information.

Although many companies sell digitizing tablets, the market has been dominated by Japanese manufacturer Wacom, whose products are widely used both by professional and amateur artists, designers, and architects, and by researchers in

the graphics and interface fields. For this reason, discussions of digitizing tablets in this paper will center around the Wacom design. Wacom—pronounced *wa* as in ‘water’ and deriving its name from the Japanese word *wa* meaning harmony—was one of the first to develop a tablet system with pens that do not require batteries or a wired connection to the tablet. This freedom makes writing or drawing with the tablet more intuitive and natural. An image of a standard Wacom tablet model is pictured in Figure 6.7 on page 268.

Modern tablets like those produced by Wacom function by tracking the position of a transducer embedded into the stylus through a magnetic field generated above the tablet’s surface. This means that the tablet can track the position of the stylus from a short distance above even before the stylus comes into contact with the surface. Aside from the position of the tool, most tablet models also collect information about the pressure the user applies with the pen, allowing the program to adjust parameters based on this information such as the color or size of a paintbrush tool. The primary input tool for the tablet is a digitizing pen or stylus that resembles a standard pen but with a non-marking plastic nib at its tip. These pens usually feature a saddle button on their barrel that can be assigned like function buttons on a mouse, and a second eraser shaped tip at the opposite end, which can also be used for marking and is recognized by the tablet hardware as a different kind of input. Other tools are also available and can be used to collect additional information. For example special 4-D mice or puck like devices can be used to collect not only positional, but also yaw, pitch, and roll information as the device is held above the tablet. However the tools are not packaged with the tablets themselves and so are not as widely used outside of specialized applications.

Digitizing tablets are extremely well suited for artists and designers, and because they replicate a writing or drawing implement like a pen or pencil, they are an ideal fit for sketching in particular. Tablet models range in size and sophistication from very small tablets with work areas about the size of a postcard, often bundled with consumer photo touchup software, to large table sized units for industrial applications. Tablets have become very popular tools and are a staple of the professional computer artist or designer's studio.

Tablets are not without their limitations however. Because marks are made on the tablet surface but displayed on a separate screen, there is a disconnect between drawing and viewing that can be disorienting for new users. For most users a little practice is all that's necessary to get over the initial feeling of unease. In fact, many soon find the separation to be a boon rather than a drawback. Fekete *et al.* relate the comments of a professional animator using a tablet for three months and then moving back to paper and pencil. Rather than feeling comfortable in familiar surroundings, the animator found it annoying that his hand and arm were now always blocking a portion of his work on the paper, something the tablet disconnect prevents [Fekete *et al.*, 1995].

Even if tablets eventually win over most users, there are some situations where people feel, either correctly or incorrectly, that this disconnect is a hindrance. Some manufacturers are now producing special LCD displays that incorporate tablet hardware, allowing users to draw directly onto the screen's surface. Zeleznik refers to displays like this as co-location displays [Zeleznik, 1998]. Unfortunately these special displays can be very expensive, and are larger and more bulky making it difficult to use them on a flat surface. Even with these special screens it is necessary to place layers of protection between the display element and the

drawing surface to make the screen more durable. These extra layers create a separation of as much as a few millimeters between the user's pen and the actual pixels, causing a parallax effect [Henzen *et al.*, 2005] [Fekete *et al.*, 1995] where the point of the pen and the point drawn on the screen don't seem to exactly match up. Also because most programs are designed to work with non co-located I/O devices they display a tool cursor to show the current input position. The cursor can be very distracting on co-located displays, especially when there is any lag between the pen's movement and the cursor [Zelevnik, 1998]. The major factor limiting wider spread use of these screen devices is most likely their cost, which can range into the thousands of dollars even for small units. Despite this cost, efforts by Microsoft and a consortium of PC manufacturers have introduced TabletPC laptops that incorporate digitizing tablet screens into a portable laptop format. These devices have proven popular with students and certain specialized professionals, and their continued development means that the screen technology is likely to become more affordable.

Although digitizing tablets represent the state of the art in pen-based input at the moment, there are a number of research efforts working on new interface hardware. One notable example is the use of electronic ink displays for drawing input [Henzen *et al.*, 2005]. Electronic ink displays function by embedding a thin substrate material with thousands of tiny pigment containing capsules called microencapsulated electrophoretic particles. Each particle contains oppositely charged white and black pigment chips. Through the application of a electric field, one or the other color of chips are induced to move to the front facing surface of the cells, causing the particle to appear to turn black or white. By embedding a large number of such particles into a sheet of material, black and white images



can be produced. By integrating this technology with a digitizing system Henzen *et al.* developed a prototype drawing pad that allows the user to draw on the same surface as the display with virtually no parallax. The researchers reported very promising results, but the technology is still in its infancy, and probably many years away from a practical consumer product.

### 2.4.2 User Interfaces

No matter what the computer artist's tool of the future looks like, it's probably safe to assume that it will bear at least a passing resemblance to the humble pen or pencil. After all, over thousands of years of art in which we have had to invent new artistic tools, the basic shape of these implements have changed very little. For the computer artist this is both good news and bad. The good news is that his or her traditional artistic skills will be easier to transfer into the digital sphere. The bad news is that they will need one more tool on their already crowded computer desks. This is because although the digitizing tablet and its kin make very good artistic tools, the so-called WIMP style—windows, icons, menus, pointing—GUI interfaces we're all used to were designed to work with a mouse and keyboard, and as any tablet user can tell you, selecting buttons, sliders, text, checkboxes, menu items, and other interface widgets with a tablet can be a bit tricky. Furthermore, the placement of these interface elements can present physical limitations.

The first problem is the way movements on the tablet correspond to cursor movement on the screen. Just like the mouse, whenever the pen moves over the tablet that physical distance is converted into a virtual distance traversed on screen by the tool cursor. However all movements of the cursor by the mouse are relative. In other words, the cursor starts out in some position and when

the mouse moves, it causes a corresponding amount of x and y deflection to be added to the mouse's current position to place it in a new one. This is a very helpful feature for a mouse because it means that using only movements within a comfortable range for the user's wrist the cursor can be moved across a screen of any size. As the user's hand reaches the edge of the mouse pad, they simply lift, move back, and continue mousing.

Movements of the pen across the tablet on the other hand are fixed relative to the tablet's active area. The tablet tracks the position of the pen both when its tip is touching the surface, and when the pen is hovering a short distance above it. This behavior is essential to replicate the physical activity of drawing, where the pen is lowered to make marks and lifted to move to new locations. If the tablet functioned like a mouse, every time the user lifted the pen the cursor would stop moving, and when they placed it back down again, the system would always start drawing from where they had left off. Instead, the tablet moves the cursor to an absolute position every time it comes into view, so when the user touches the tablet surface with the pen in the lower right hand corner and then the upper left, the pointer instantly jumps to corresponding positions on screen.

This pen style of interaction is good for creating art, but it presents a problem for user interfaces because tool pallets and menu items are usually located around the periphery of the screen so that they don't block the central work area. This means that any time users want to change tools or use a menu function they need to make a larger movement with their hand or even entire arm to reach the correct screen position. If the artist is working in one specific area with several tools, frequently changing back and forth and constantly moving out of the work location is tiring and wastes time as the artist has to reorient themselves after

each trip.

There are some minor things that users can do to address this issue. Most computer graphics programs include extensive keyboard shortcuts to allow many common functions to be accessed quickly or bound to function keys on the tablet or pen barrel. Also, most tablets include a mousing mode in which the cursor's movements are relative, like a mouse, rather than absolute. However, neither of these offer a complete solution. Learning keyboard shortcuts takes time, and only so many commands can be reasonably accessed this way before it becomes too difficult to remember which key goes to which command. Mouse modes offer some support, but require some explicit mode change, usually through a keyboard command or access switches on the tablet or tool, adding overhead and the possibility of error. Despite these shortcomings most commercially available programs rely on methods like these owing to the fact that many were designed before tablets were in wide spread use, or because they support tablets only as an additional feature rather than the default method of input. Luckily the human computer interaction research community has been working in this area for some time and offers several possible solutions.

#### **2.4.2.1 Gestures**

One of the most widely adopted methods is gesture. Gestures are predefined regular marks made by the user that the computer system recognizes and interprets as commands. Long *et al.* present several reasons why gestures are an ideal counterpart to pen based computing [Long *et al.*, 1999]. Gestures are sometimes faster to complete because a single gesture can encapsulate both a command and the input data into a single action by the user. There are also many examples of

commonly used gesture marks, such as proof reading marks or correction symbols that are already commonly used or at least iconic, which makes them easy for people to learn and remember. To this list, we can also add that in most cases a gesture is performed in context, meaning that the user need not shift attention away from the work. As Long *et al.* point out, with the proliferation of pen-based computers gestures are becoming an increasingly popular interface method.

Exactly what constitutes a gesture varies from one implementation to the next. Some systems require that gestures be performed as a single stroke and constitute very simple patterns. Others can handle multiple strokes or complex symbols such as letters, words, or small pictures. Still others combine a tool operation such as lasso selection with a delimiter gesture to allow selection and operation in a single stroke [Hinckley *et al.*, 2005].<sup>7</sup> In most cases the strokes that define a gesture are not intended to create any visual elements and the lines that make the gesture are not displayed or disappear after recognition.

We usually think of situations in which the stroke is captured and in some way transformed as a recognition system rather than a gesture system. The most common example of this would be a handwriting recognition system in which hand written letters are transformed into text. However the line distinguishing these two philosophies can get very muddy indeed, and many systems implement both operations side by side, using one set of gestures to create marks and others to delete them. In terms of interface design, many of the issues facing these systems are the same, and so we will speak of them interchangeably.

In many ways gestures are not unlike handwriting recognition and so many of the techniques used to recognize handwriting are also used for gesture recognition.

---

<sup>7</sup>Gestures that include both command and input are sometimes called high bandwidth interface mechanism [Conner *et al.*, 1992].

Extremely simple gesture systems can be recognized with nothing more than a finite state machine [Zeleznik *et al.*, 1996], but for anything more than the most basic gestures a more advanced system is needed. The two most popular methods are *neural-networks*, and *feature-based* recognition [Long *et al.*, 1999].

Neural-networks were developed by the artificial intelligence and machine learning research fields as a way to model the pattern matching abilities of neurons, the cells that make up our brains. A neural-network consists of a number of neurons or nodes connected by links into a graph structure. Each node takes an input signal, processes that signal based on some internal formula and a set of parameters, and then produces a low or high output. The graph is constructed in such a way that raw information flows into one end, filters through stages or layers of nodes possibly in a feedback process, and finally produces a single guess as to which category is the best fit for the input. In order to use the network for pattern matching it must first undergo a supervised learning process in which a learning set of inputs and their correct classifications are provided. The system continually runs through the set, adjusting the internal parameters of each node, until the output of the network matches the desired results.<sup>8</sup>

According to Long *et al.*, neural-networks are often preferred because of their higher recognition rates in comparison to other techniques. However, this high recognition comes at the cost of the long and laborious training process that is required to develop the network before it can be used. To achieve good results learning sets of hundreds or even thousands of examples must be provided and processed, which can take a great deal of time. Despite this limitation, the accuracy of neural networks, and their ability to continue learning from the user in

---

<sup>8</sup>For a more thorough account of neural-networks see [Russell *et al.*, 2003].

the field have made them very popular for both handwriting and gesture recognition. For example, the handwriting recognition used in Apple Computer's Newton PDA used neural-network technology [Yaeger *et al.*, 1998], and is now the basis the Inkwell handwriting technology integrated in Apple's OS X operating system.

Whereas neural-networks process the entire gesture as a whole, feature-based recognition systems make their classifications based on the existence and distribution of a finite set of details extracted from the raw gesture [Long *et al.*, 1999]. As the name suggests, feature-based algorithms take measurements of elements within the raw stroke called *features*. These features can include things like the aspect ratio of the stroke's bounding box, the number of strokes used, the angle of the initial stroke, total length of the gesture, instance of crosses or self intersection, etc. In order to classify a set of gestures, the system is provided with a training set. Then as unclassified gestures are entered, the system statistically compares the features of the unknown input with those in the training set and calculates which class the new input most resembles.

A good example of a feature based recognition system is the calligraphic interface library CALI developed by Fonseca *et al.* [Fonseca *et al.*, 2002]. CALI uses geometric properties, shape filters, and fuzzy logic comparison to training sets to recognize gestures made from one or many strokes. By focusing on the deconstructed features of the gestures rather than solely on their overall appearance CALI can recognize the same gesture drawn in disparate sizes, arbitrary rotations, drawn with dashed or continuous strokes, and containing multiple overlapping lines. CALI's developers claim an accuracy rate as high as 97%. This rate may seem impressive, and it is, however in general feature-based systems are not as accurate as neural-networks, which boast rates closer to 98% or 99%. On

the other hand, feature-based systems require far fewer training examples—on the order of dozens rather than hundreds—meaning that they can be used in an iterative design process, and are much easier to implement.

No matter what recognition algorithm is used, gesture recognition, like handwriting recognition, is a very difficult problem, and for all but the most limited set of gestures 100% recognition rates are unrealistic. However, one report notes that users find recognition rates even as high as 98% inadequate [Briggs *et al.*, 1993], and so much research is focused on how to make gestures easier for the computer to recognize. Long *et al.* tackled this issue head on, performing an in depth examination of gesture systems that resulted in the development of gdt, a utility to help interface designers interactively create a set of gestures with a good recognition rate [Long *et al.*, 1999]. In the process of those examinations Long makes several key observations.

The first regards how gestures can be shared between programs. It seems logical to want to create some standard set of universal gestures that would function across all programs, but Long *et al.* suggest that this is not realistic. Although a small set of universal gestures may be appropriate for common activities in the same way that copy and paste are used in almost any program, most applications require their own set of gestures to accommodate their unique features.

Long *et al.* also note that using a general set of gestures has some consequences that may not at first be evident. Because recognition involves differentiating between each class of gestures, the composition of a gesture set has a large influence on the accuracy of the system overall. Stated simply, if any of the gestures in the set look too similar then the system is more likely to confuse them, driving its accuracy down. This is a fairly obvious concept but it has deep implications

for the design of gesture sets. For one thing this means that the gesture designer needs to understand the recognition system being used, and how to tune his or her gesture set to avoid elements that will be easily confused. Trying to alleviate this requirement prompted Long *et al.* to create gdt in the first place. It also means that as the number of gestures handled by the system increases, the more difficult it will become to find gestures the computer can differentiate, let alone ones that are iconic and memorable to users. As Zeleznik *et al.* put it, “one of the principal difficulties in developing a good gesture-based interface is managing the delicate tradeoff among gestures that are natural, gestures that are effective, and gestures that are effective within a system that may already use similar gestures...” [Zeleznik *et al.*, 1996].

These observations can have a large impact on the overall design of the program. For example, Long *et al.* report that many PDA users expressed a desire to have more gestures available and to be able to create their own. However, giving users this ability puts the users’ desires at odds with their demand for high accuracy. There are examples in the literature of programs that allow users to do just this [Igarashi & Hughes, 2001], however in general, Long *et al.*’s work suggests that small, specialized gesture sets are best.

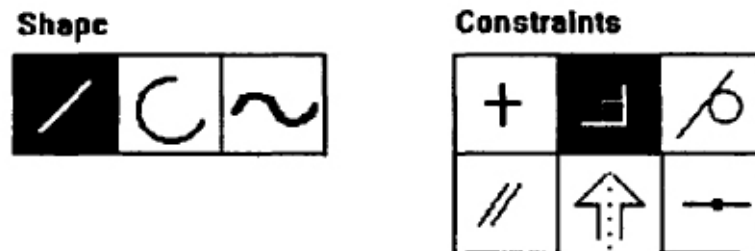
Even the most accurate system is bound to make mistakes from time to time at which point the user needs to correct the computer’s interpretation. Because this in fact tends to happen quite a lot, a good deal of gesture research has focused on how best to integrate the correction process into the user interface so that it doesn’t become a burden or annoyance to the user. In general this entails some means of informing the user what operation or gesture the system thought they entered and presenting them with a list of possible alternatives from which the



user can select their actual intention. An early example of applying this technique to a gesture based graphics program was presented by Eggli *et al.* as part of their 3-D sketch modeling prototype called Quick-Sketch [Eggli *et al.*, 1995].

Quick-Sketch allows users to quickly create technical drawings by converting their rough sketchy strokes into clean lines and regular shapes. The system also uses a drawing recognition system that attempts to infer constraints for new geometry created by the user by recognizing the type of stroke, the context, the current system preferences, and by analyzing the dynamics of stroke creation. Although this recognition process is not specifically based on gestures, the program still needs to inform the user about the recognition and offer them a way to intervene when the system is mistaken.

To aid the user in correcting the inference system, Quick-Sketch features a context sensitive menu containing several icons representing the different kinds of inferred shapes and constraints available in the system. When the user enters a stroke, the icons corresponding to the inferences made by the system are highlighted, showing the user exactly what the system changed. The user can also deselect interpretations they did not intend, or select new ones the system missed and their stroke is instantly reinterpreted and redisplayed—see Figure 2.6



**Figure 2.6.** The gesture correction interface from Eggli *et al.*'s Quick-Sketch modeling application [Eggli *et al.*, 1995].

Interfaces of this type have proven to be an effective means of negotiating the meaning of a user's ambiguous input. This technique has proven particularly effective in gesture and recognition based 3-D modeling programs because of the inherent ambiguities that arise when manipulating 3-D objects on a 2-D display system. We will look at two specific examples of modeling applications later on, one by Pereira *et al.* called GIDeS [Pereira *et al.*, 2000] and one by Igarashi and Hughes called Chataeu [Igarashi & Hughes, 2001], both of which utilize a variation of this mechanism.

Even with their inherent design difficulties, gestures have proven an extremely effective means of working with a pen-based computer. A pair of researchers at the University of Toronto even developed an entirely new desktop user interface called Bump Top that uses gestures to control file manipulation tasks [Agarawala & Balakrishnan, 2006]. However, not everyone is enamored with gesture interfaces. Some argue that gesture systems create more interface distractions than they remove.

The first issue centers on the realities of the recognition task. As we have discussed gesture recognition is not perfect and it is inevitable that the system will make recognition mistakes. When this occurs the user must correct the system or undo the command and try again. Most gesture system designers understand this limitation and strive to provide their users with a fast and intuitive correction interface so that they can make their changes and move on. If this only happens occasionally the overhead is not prohibitive, but if recognition is poor, or gestures make up a large part of the interface, unfortunate users can find themselves calling up the correction box over and over again. As you can well imagine, aggravation quickly ensues. Each time a correction is needed a user's train of thought is broken

and he or she must shift his or her focus back and forth from the work in order to make correction.

The second issue can arise from the way gestures are integrated into the system. Systems that use gestures purely as commands and not as drawing mechanisms need a way to differentiate which strokes are intended as artwork and which as commands to the system. Most programs accomplish this through some kind of mode switching in which the user must indicate to the system that their next squiggle or doodle is not a whimsical witch's hat or frowny face, but rather an instruction to the application. If gestures are an important part of the user interface then the user will probably be switching the mode back and forth quite often. If careful thought is not given to how the mode switch is integrated into the user interface then changing mode over and over again can become as tedious and distracting as using standard buttons and menus was.

One popular technique has been to use a modifier key [Baudel, 1994] [Zelevnik, 1998] that, when struck or depressed, signals the system to alter the current mode. The drawback to this method is the user needs to keep one hand on a keyboard at all times in order to activate the modifier, or else hunt down the correct key each time. It's also easy to forget that people working with TabletPC's or PDA-like devices might not have a keyboard in the first place. Some have suggested using the barrel buttons that come equipped on most tablet pens for mode switching [Zelevnik, 1998], however as Ramos *et al.* point out, it's very easy for the user to activate the barrel buttons by mistake [Ramos *et al.*, 2004].

Several researchers have suggested that a better solution might be for the system to make the determination itself. Saund and Lank [Saund & Lank, 2003] present an example of this type of system. The two researchers developed an

inference that tries to determine the meaning of a particular input based on properties like the context of the stroke, path length, and closure. Even with all of this information however, there are circumstances in which the user's input is ambiguous, and in these cases the system displays a contextual menu for the user to explicitly indicate his or her intentions. Although Saund and Lank reported generally positive reactions to their system, others feel that any distraction is unacceptable, especially in a creative environment. Sentiments like these have led some researchers like Baudel to reject gestures all together, stating that gestures limit the user's ability to draw freely [Baudel, 1994].

Another criticism leveled at many gesture systems is that they can be difficult for novices to learn. To make effective user of the system, users need to memorize the set of gestures that are available. This may be straightforward for highly iconic or simple gestures, but as the set of gestures increases in size users must inevitably refer back to some sort of user's manual or help screen to refresh their memory until they have the basic commands memorized. There is also a deeper problem here that is a little harder to see at first. Consider what happens when you start up a new program for the first time. In a program based around the standard WIMP interface, you'll be presented with a set of hierarchically organized menu commands, and in some cases a tool bar with helpful icons. Although you have never used the program before,<sup>9</sup> you can teach yourself what capabilities the program makes available simply by perusing the menus or mousing over the tools. If the user interface designer has done his or her job well, you should be able to quickly find any function of the program simply by looking under the most appropriate menu heading or tool group. Now consider a purely gesture based

---

<sup>9</sup>And like most computer users you probably haven't looked at the manual.

system. You start the program and are confronted with a blank screen. What can you do? What commands are available? How do you activate them? What are they called so you can look them up in the help system? And how do you access the help to begin with? In situations like this, your only recourse is to hope that the manual writer did a better job than the UI designer.

Very few pure gesture based programs exist today, and those that do, such as Agarawala and Balakrishnan Bump Top, use a very small gesture set designed to capture the sorts of intuitive strokes users are likely to make. However, although this is something of a convenient exaggeration, many common legacy interfaces that were and still are in widespread use suffered from exactly this problem, stark interfaces that offer the inexperienced little in the way of guidance. A good example are DOS and Unix-style command line interfaces [Kurtenbach *et al.*, 1994], which are completely opaque and frankly frightening to new users. Thankfully, most programs that actually utilize gestures more often include a gesture set to augment their standard WIMP interfaces. Still, these programs need a way to educate their users about what sorts of gestures are available and how to perform them. They also need a mechanism to train their users, slowly weaning them off of the need for references as they transition from novice to expert. A prominent research team in the field of user interface design—Gordon Kurtenbach, T. P. Moran and William Buxton—classified these design needs as the principles of *revelation*, *guidance*, and *rehearsal* [Kurtenbach *et al.*, 1994].

Many researchers have tried to tackle this problem by integrating gestures into familiar widget systems, or by using gestures to access widgets. A good example of this kind of solution is the animated crib sheet developed by Kurtenbach *et al.* [Kurtenbach *et al.*, 1994]. A crib sheet can be thought of as a tool tip text on

steroids. As the user works, at any point they can call up a crib sheet that displays the gestures available to the user in the current context. The sheet appears off to the side so as not to block the user's workspace, and displays pictorial representations of each gesture along with the command it performs. The user can even select a command and ask for a demo, in which case the selected gestures, possibly with several variants, is animated in context from the point at which the user initially invoked the crib sheet.

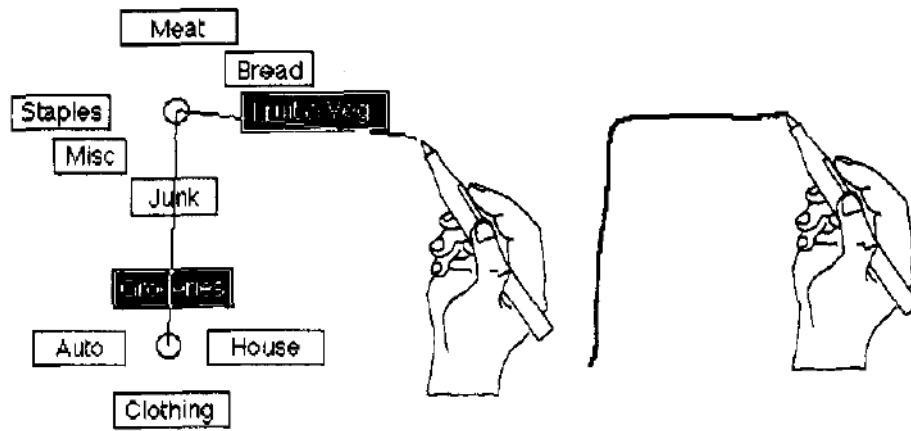
Kurtenbach *et al.* implemented a prototype version of crib sheets as an extension to a white boarding program. The researchers had promising results in user testing, but found that many users misinterpreted the crib sheet's purpose, trying to click the gesture icons as buttons or drawing the gestures over top of the examples rather than in context.

In the same article, Kurtenbach *et al.* mention another technique called 'marking menus' or 'pie menus', which Kurtenbach and Buxton have studied in the past [Kurtenbach *et al.*, 1994]—see Figure 2.7. Rather than providing the user with a visual indication of some iconic gestures, marking menus combine the idea of a contextual or right-click menu with the idea of a gesture to make the menu selection action into the gesture itself. The process works like this. The user first activates the menu, which appears as either a circular region in the case of a pie menu or a set of text boxes arranged in a circle in the case of marking menus, centered at the cursor position. To select a command the user simply draws a stroke in the direction of the menu item. The advantage of this system is that, as users learn where the commands they often use are located around the circle, there is no need to wait for the menu to appear. They can simply make the correct gesture.

Kurtenbach and Buxton's 1993 paper [Kurtenbach & Buxton, 1993] point out that not only does this save the user time, but it also helps make the transition for novice to expert user faster. Because the gestures is the same whether you allow the menu to appear or not, each time novice users peruse the menu system they are also inherently rehearsing the correct gestures. This is in stark contrast to novice users using a standard menu system and then having to learn a set of secondary keyboard shortcuts to become expert users. Kurtenbach and Buxton did extensive user testing to determine how best items should be arranged within the menus. They also explored having a hierarchy of items in the menu, requiring a zig-zag type of gesture, and found that hierarchies as deep as four levels were effective. Some years later Fitzmaurice, Khan, Pieké, Kurtenbach and Buxton developed an extension to marking menus they called tracking menus [Fitzmaurice *et al.*, 2003], which follow a users' cursor as they reach the edge of the menu system, something akin to pushing the lip of the lid of a jar with the tip of a pencil.

At the opposite extreme of the spectrum from the use of iconic gestures are researchers trying to develop new user interface widgets that are easier to use in a pen-based environment. One effort is trying to replace the idea of clicking, which works well with mice, with something the researchers call *goal crossing* [Accot & Zhai, 2002], which is more akin to drawing a check mark. In a goal crossing interface rather than being presented with a button the user is given a vertical or horizontal line. To activate the widget, the user simply draws a small mark crossing the line. By varying the number of times or direction of the crossing the widgets can also include additional functionality.

Accot and Zhai performed a number of user studies to look at the different properties of goal crossing and how it compares to standard interfaces. They point



(a) A conceptual illustration of a hierarchical marking menu from Kurtenbach *et al.* [Kurtenbach *et al.*, 1994]. Notice how the user's hand can form the same gesture with the menu visible, and invisible to access the same command.



(b) A modern example of a marking menu in Alias's SketchBook Pro 2-D drawing application.

**Figure 2.7.** Examples of marking menus.

out that normal pointing and clicking activity can be difficult when the pointing target is small. They also state that for many segments of the population such as the elderly, common mousing skills like double clicking and pointing can be difficult or impossible. They suggest that standard interface widgets could be replaced with goal crossing widgets, which can be easier for some people to use, and tend to take up less screen real estate than traditional clicking widgets.

In 2004, Apitz and Guimbretière presented a paper describing a TabletPC drawing program called CrossY that exclusively uses goal crossing widgets. The authors developed several novel systems to incorporate goal crossing into standard



program components such as file dialogues, scroll bars, check boxes, and modal alerts. They also extended the idea of replacing single buttons by allowing the user to string together several selections in a single stroke, making it possible for example, for the pen tool, a line width, and a paint color to be selected in one fluid motion. Some of their methods appear more effective than others, but overall the system provides a good example of goal crossing in a real world environment. Apitz and Guimbretière’s publication did not include a formal user study, but informal user reactions seemed to indicate that goal crossing is a good fit for drawing and pen-based computers.

#### **2.4.2.2 Pressure**

Because the tablet is used to move the cursor around the screen just like a mouse, we tend to think of tablets as a simple replacement for them. It’s easy to forget that tablets offer extra information that a mouse does not provide. The most useful is probably pressure. As we have discussed, most digitizing tablets and some special tablet displays measure the pressure with which the user marks on the tablet surface. This feature is designed primarily to manipulate parameters of a tool within a paint program, such as the size of a brush, or the color or opacity of paint. However some researchers have recognized that this information could also be used to adjust interface parameters. Tsang *et al.* present one simple example of using pressure to signal a mode change [Tsang *et al.*, 2004]. Within their 3-D modeling application a user’s strokes on the tablet are interpreted as line drawing commands. However if the user exerts a larger amount of pressure the command is interpreted as a gesture.

When used as an interface tool, pressure has the disadvantage that the user’s

ability to control it is not as granular as other more familiar input sources like position. Furthermore, some have also noted that pressure preferences differ from user to user, or change between sessions with a program. Tsang *et al.* for example noted that, although their system only distinguished between two levels of pressure—normal or light pressure for drawing commands and heavy pressure for gestures—variation in what was ‘normal’ pressure was enough to require personal adjustment for each user.

A more complete analysis of how pressure could be used in an interface was presented by Ramos *et al.* [Ramos *et al.*, 2004]. The authors performed a series of controlled and rigorous studies with 12 volunteers to see exactly how well users can manipulate pressure, and how best to use pressure as an input channel. The research team developed a series of tests requiring users to move a cursor along a 1-dimensional axis into a goal, using pressure and both the presence and absence of visual feedback. The researchers also tested four methods for the user to signal a selection using the digitizing pen: *click*—where the user performs a click operation with one of the barrel buttons, *dwell*—where the user maintains the desired level of pressure until a timeout is reached, *quick release*—where the user lifts the pen quickly from the tablet surface, and *stroke*—where the user makes a small lateral stroke.

The results of the user studies led the researchers to a number of important observations. The first has to do with the application of pressure itself. Pressure is reported to the system from the tablet hardware in hundreds or even thousands of levels, which is useful to express the gradual gradation of pressure within, for example, a brush stroke. However when the user is asked to discretize their own pressure, for example to activate a number of discrete levels of a parameter, most

users can differentiate no more than six distinct levels of pressure.

The authors also noted that many users find it hard to maintain a very low-pressure level, often reporting that “the tablet is too sensitive” when application of very low pressure is required. The authors go on to suggest that applications wishing to utilize pressure for parameter adjustment should consider using a non-linear transfer function to map raw input to the desired parameter range. Ramos *et al.* also found that providing the user with some visual feedback is important, and has a large effect on the user’s accuracy, even after several hours of practice.

Finally the authors developed the following advice regarding the selection methods. First regarding barrel buttons the publication reported results that support anecdotal evidence noted by other authors [Miller, 2005], that the barrel buttons placed on the side of the digitizing pen suffer from poor ergonomic design and are prone to accidental activation. It turns out that as users draw with the pen, it tends to move and turn in their hands. Eventually the pen turns so much that simply gripping the pen normally can accidentally trigger the barrel buttons. The authors also noted that it was difficult for users to click the side buttons while maintaining a constant pressure and position. Overall this prompted them to rate clicking as the poorest means of selection. As for the other methods the paper notes that users also had difficulty with the stroke method in which selection was signaled by a quick deflection of the pen. They found instead that both quick release and dwell are accurate and effective means of selection. Because quick release does not require the user to wait, which can be annoying for repeated actions, the authors suggest quick release as the best method. Based on their findings, Ramos *et al.* also present a set of conceptual pressure widgets demonstrating several ways pressure could be used in the interface.

### 2.4.3 Computer Sketching Applications

If you were to take a stroll down the computer graphics aisle at your local computer store, you're likely to see a wide variety of applications targeted to many different kinds of creative output. There are programs to help you clean up grandma's old photos. There are programs to help you design a brochure for your company. There are programs to help you draw up plans for that new greenhouse you're thinking of building. And there are tons of fun art programs to help keep the kids<sup>10</sup> entertained. You may notice however that there are not a lot of programs designed for sketching. Based on what we have already discussed, we know that sketching can be an immensely important component of any creative endeavor, and so the absence of programs catering to this so primal of needs seems a startling omission.

Exactly why sketching programs are few and far between is a subject open to speculation. Some may feel that sketching is such a trivial activity that spending money on such a program would be a waste. Many graphics programs tend to sit at the high end of the software market, and justifying a price tag of hundreds of dollars for something that can be done with 10¢ worth of paper and pencils does seem a bit wasteful. However, as we have already discussed, traditional sketching has a number of limitations, especially for commercial artists and designers, that could be solved by integrating sketching into the computer.

A better theory is perhaps this: because sketching can mean so many things to different people, it's very difficult to create a single tool that can satisfy everyone's needs. An illustrator or cartoonist for example might be interested in the different tones and gradations they can get among strokes while an architect is

---

<sup>10</sup>Or kids at heart!

more interested in easily making regular angles and parallel lines, or an engineer would like to include calculations and formulas. Sketching has such a motley and diverse set of requirements that combining them all into a single program is simply overreaching. Just as a single computer program for sketching is too broad, there is also no single program for art, and the diversity of computer graphics programs exemplifies this. However, if this were the only limitations then we would expect to see a multitude of computer sketching programs for preliminary work right there on the computer store shelves next to their respective counterpart programs for the later stages of the creative process. However this too is not the case.

Perhaps the best explanation for this trend is historical. Looking at the programs available today, most are geared towards the final stages of the production process, the time when a coherent and attractive final product must be coalesced from a jumble of parts. Early computer graphics programs found a logical inroad at this stage of the process because they offered an attractive alternative for commercial organizations looking to streamline what is usually the most expensive and laborious part of the production process. We say commercial interests here because in the early stages of computer graphics large commercial, governmental, and academic organizations were the only ones who could afford the large computers and advanced software necessary to do graphics work. With the introduction of personal computers and the growth of fields like desktop publishing and photo editing many of the industrial features and techniques were ported into professional and consumer grade software. Because of this trend most professional and ‘prosumer’ grade graphics applications target their core features at later stages of design.

For professional artists and designers, a lack of computer sketching software

may have caused a little jealousy, but did not represent a particular hardship. After all, most if not all of these professionals were well versed in traditional sketching techniques and were almost certainly putting them to use in their every day work. In fact, most of the artists and designers of the day would have had little if any experience with computers, and so a transition to a new sketching system and a whole new style of working might have been more disruptive than anything. However, as designers became more comfortable with computers—getting used to using a whole suite of professional applications in their day-to-day work—the limitations of traditional sketching and the difficulty of transitioning work from paper to pixels started to become clear, prompting artists and designers to begin looking for digital alternatives.

#### **2.4.3.1 Commercial Software**

Unfortunately for these digital sketching pioneers the professional computer graphics market was not as quick to address the need as they had been to recognize it. Perhaps the major software companies in the market saw such a product as beneath the prestige of their product line, or were concerned that corporate IT professionals would have a similar opinion and not place orders. Whatever the cause, professional artists and designers had to make creative use of the tools at their disposal.

Although the specific requirements of any particular sketcher may be diverse enough to preclude a single program, as we've said the mental activity of sketching can be done with almost anything, from a stick in the mud to high quality art supplies. This same sketching ethos extends into computer graphics where artists came to use the more sophisticated drawing programs at their disposal for

sketching. Almost any computer graphics program can be used in this way, but two programs in particular feature prominently in computer sketching.

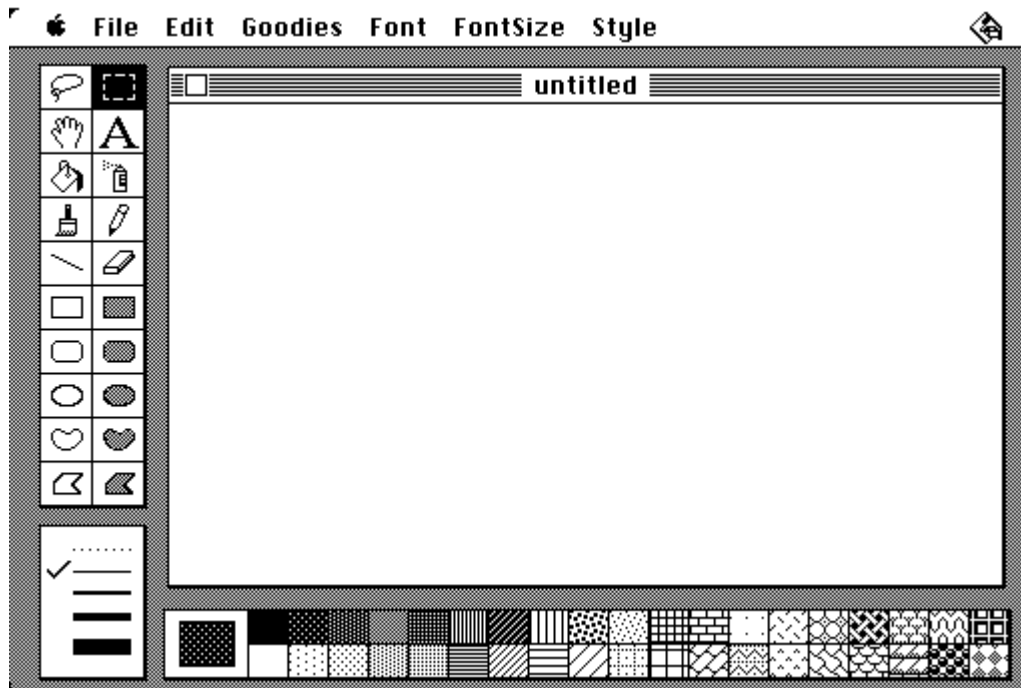
Probably the first commercially successful computer-drawing program, and the first GUI paint program was MacPaint written by Bill Atkinson. In the early 1980's Atkinson worked for Apple Computer developing a graphics system later named QuickDraw that was the basis of Apple's Lisa and Macintosh graphical user interfaces. According to an interview with Atkinson by Andy Hertzfeld [Hertzfeld, 2004a], a fellow Apple developer, Atkinson developed MacPaint as a testing platform for the graphics subsystem, and only later polished the program for public sales. Hertzfeld's interview describes how Atkinson, "wanted to create something that was simple, elegant and fun to use", and this design philosophy resulted in a famously influential user interface—see Figure 2.8

Released in October of 1983, MacPaint allowed users to draw with a variety of monochrome patterns, selected from a pallet fixed at the bottom of the screen, and included a number of different brushes and basic shapes, which were accessed from a tools pallet situated along the left hand side. Users could also make rectangular selections that were displayed on screen by an animated border of 'marching ants'<sup>11</sup> that made the selection stand out from the background. MacPaint also included a paint bucket tool that used a seed fill algorithm to fill areas with solid color, a simple magnification view, a lasso tool for irregularly shaped selections, and the ability to incorporate text. All of these features and UI designs, which originated with MacPaint are now part of the standard vocabulary of computer graphics.

MacPaint—along with MacWrite a WYSIWYG word processor program—

---

<sup>11</sup>The story goes that Atkinson invented this 'marching ants' visualization after seeing a novelty tavern sign that moved lights over a section of colored plastic to crudely simulate flowing water.



**Figure 2.8.** A screen capture of the original MacPaint interface, designed by Bill Atkinson. Users of modern paint programs like MSPaint or Photoshop will be familiar with the now ubiquitous toolbar and palette interface.

were bundled with the release of the first Macintosh in 1984 [Breen, 2006]. MacPaint was featured in marketing as a professional and educational tool using example images created by Susan Kare [Hertzfeld, 2004b], however the decision to bundle the application with the Mac meant that thousands of Apple users simply had the program around, and many<sup>12</sup> spent untold hours simply sketching and doodling with the program. When Microsoft Corporation first released its Windows operating system, a similar basic paint program called Paintbrush was bundled with the system, and its successor Microsoft Paint has been a standard element of every version of Windows since, leading to a similar ubiquity that continues to this day.

---

<sup>12</sup>Including your author.



MacPaint has since been replaced by much more sophisticated computer paint programs, however to the first generation of personal computer users programs like MacPaint represented their first experience with computer sketching, and similar programs like MSPaint continue to offer a quick and lightweight medium with simple tools that are well suited for quick-and-dirty computer sketching.

Although sketching can be a simple activity, not all of the programs that support it are simple. Surprisingly, one of the most popular programs used for computer sketching was in fact designed not for drawing at all, but photo manipulation. Brothers Thomas and John Knoll, one a PhD student at the University of Michigan and the other a designer for Industrial Light and Magic, developed Photoshop and licensed the program to Adobe in 1990 [Story, 2000]. Photoshop was and is primarily designed to allow users to perform many of the photo manipulation tasks that were once performed by photographers working with the specialized alchemy of films negatives and chemical washes in a dark room. Photoshop offers a wide variety of tools, a vast array of complex and sophisticated filters and operations, and includes a plug-in architecture that has spawned a thriving industry of add-on's allowing Photoshop to quickly include support for new technology.

Although intended as a tool to adjust existing photos, many of the same tools are also effective in creating original drawings. As Photoshop has matured its developers have slowly added additional functionality to these areas of the program, including support for digitizing tablets that allow pen pressure to adjust tool parameters like brush width or paint opacity. When first introduced Photoshop was not the first photo editing software available, but its price tag of under \$1000 made it an instant success, an a staple of the graphic arts community.

It may seem strange that a program that was originally intended for photographers would be useful for sketching, but Photoshop has passed through nine versions since 1990, and over the years added many paint and design features to cater to new markets like print and the web. Because of its popularity, in many professional and academic environments it is often as ubiquitous as paper and pencils, and its expanded features make it an attractive and familiar medium for computer sketching. Today's graphics market is replete with alternatives to Photoshop with myriad programs offering similar or better features in different price ranges. Our discussion here is by no means meant to claim that Photoshop's features are any better than others. However Photoshop's ubiquity and feature creep have conspired to make it a common choice for graphics designers, professional artists, and even students and hobbyists who need to do any kind of 2-dimensional computer graphics, including sketching.

Where the painting aspects of Photoshop are largely the result of a feature creep, computer graphics software has also been developed specifically as a first principles artistic tool. The most well know and widely used example is Corel's Painter.

Mark Zimmer, Tom Hedges, and later John Derry originally developed Painter as a computer graphics program designed specifically to mimic natural media<sup>13</sup> in the same way that a synthesizer or digital keyboard mimics a musical instrument [Contributors, 2006e; Derry, 1996]. Painter includes a staggering array of simulated art supplies including pencils, markers, pastels, charcoals, watercolors and oil paints, just to name a few. Not only are these supplies simulated, but so too is the feel of working with each. Oil painters for example can mix paint colors on

---

<sup>13</sup>To emphasize its natural media features early versions of Painter were even sold packaged in metal paint cans.

a pallet and load different colors on different areas of a brush. Watercolor artists can lay down layers of water that effect the flow of pigment onto the drawing, and alternately dry the canvas to produce other effects just as they would in real life. Painter also makes heavy use of input from a pressure sensitive digitizing tablet. Unlike Photoshop for which tablet support was a later addition, tablet access is an integral part of Painter and a key factor in simulating the feel of natural media, so much so that the program is severely limited when used with only a mouse and keyboard.

By mimicking the artistic supplies and techniques traditional artists are familiar with, Painter has several advantages for pure artist over other computer graphics programs. As John Derry points out in an article describing the early days of Painter [Derry, 1996], by mimicking natural media Painter allows the technique and skills an artist may already have to translate into working with the computer. This means that the artist with a distinctive style or a favored technique can move into working with the computer, or even move back and forth between physical and digital while maintaining a cohesive look to their work. Even more exciting perhaps is the ways in which the digital medium can expand the artist's abilities. Derry mentions that by working in the computer different media that might be mutually exclusive in the physical realm can be successfully combined. Furthermore, the computer can be used to create new artistic tools like shape changing brushes or tools that paint with images—functions a natural media artist could only dream of.

All of this fabulous diversity makes Painter an extremely versatile tool, and its list of capabilities includes features targeted at sketching. Painter includes an assortment of standard sketching tools including pens, markers, and graphite and

colored pencils that behave just like the natural media. This has the advantage of giving a sketch artist the same feeling of sketching on paper. This is also an advantage for other artists who can begin by making a sketch using the digital pencils and then move on to filling in their works with paints or inks, just as they would in real life.

The drawback to this system is that the large numbers of features included in Painter make it a rather heavy-weight program. The heft of painter is felt in the staggering number of options it offers, and the computational power required to offer them. Working with painter is like working with a large closet of supplies. Everything you could want is in there somewhere. This is a real boon in completing a full project, but in the early sketching stages it can feel more comfortable to simply grab a few pencils and a sketchbook, leave the room and all the supplies behind, and go sit outside. Small lightweight programs like MacPaint or MSPaint capture this feeling well, but larger, more fully featured programs like Painter or Photoshop can sometimes feel like swatting a fly with a bazooka.

Given the success of MacPaint and Photoshop we may be tempted to think that any computer graphics program can make a suitable computer-sketching medium. It might even seem logical that programs target at specific user groups would make the best sketching tools for work in that discipline. Although some degree of sketching can probably be done with any program, it turns out that there are some application domains that make particularly poor computer sketching programs. As we have discussed, the cognitive processes that drive sketching are dependent on a degree of ambiguity to fuel the processes of feedback and incremental refinement. This requirement can be a problem for applications designed for domains like architecture and technical drawing that require a high degree of

precision.

The CAD software used in these domains are designed to aid the architect or designer with features like constraints and dimensioning that make it easier to create drawings with an exacting level of detail. There is no doubt that this is important work, and stretching back to Ivan Sutherland’s SKETCHPAD, CAD applications have historically been one of the main driving forces behind the development of computer drawing programs. CAD systems are extremely well suited for their purpose, but because the requirements of design are so rigid, trying to sketch something—even something with very regular features—can be a frustrating experience in these applications.

Because the tools available are simply not designed for preliminary work, even with the resources and computer experience of today’s designers and engineers, many choose to begin their work with paper and pencil, only moving to the computer after they have a complete description of their design in physical sketches [Baudel, 1994].

More recently Alias|Wavefront<sup>14</sup> released what is probably the first commercial application directed explicitly at sketching. The development of the application is described in a publication by Miller [Miller, 2005]. Debuting in 2002 to coincide with the launch of Microsoft’s TabletPC platform, Alias Sketchbook Pro was developed specifically to cater to “creative professionals who do freehand sketching, and need high quality results” [Miller, 2005], a group that would include character designers, industrial designers, and fine artists.

In order to develop a sketching system that would cater specifically to this group, a design team within Alias performed extensive user research. By making

---

<sup>14</sup>Alias has since passed through several hands, and is now owned by AutoDesk.

clear cut descriptions of exactly what the goals of Sketchbook Pro would be early on, and working around a design philosophy of ‘elegant simplicity’, the team developed a focused set of features.

First and foremost Sketchbook Pro was designed for freehand drawing and so the program is expressly intended for pen-based computing, either on a TabletPC or a standard workstation with a digitizing tablet. Not only is artwork created with the pen, but the entire interface of the program is designed to be easily accessible with strokes and gestures. Rather than the standard palette and menu layout that has become so common in modern graphics applications, Sketchbook Pro uses a unique rounded tool pallet situated in the lower left corner of the screen. We saw an example of this palette in Figure 2.7 on page 72 when discussing marking menus. The remainder of the user space is filled with the drawing surface, removing the distractions of menus and floating toolbars. To keep the onscreen tools to a minimum, the interfaces make extensive use of marking menus [Kurtenbach *et al.*, 1994] to access additional tools and options. Further research publications also indicate that designers at Alias are developing new gesture-style menu and pallet tools [Fitzmaurice *et al.*, 2003] specifically for the application.

According to Miller’s publication, the design team did extensive work to find a feature set that would be both small, but also expressive enough for sketching. They intentionally shied away from features aimed at CAD or photo editing that are often found in other graphics programs. The team also rejected structured drawing tools like those found in object oriented drawing packages, noting that these might seem to be inline with the target audience, but are not representative of sketching. The designers also included features to integrate the program into the early stages of an existing workflow, allowing users to export their work into

common file formats.

Sketchbook Pro is a relatively new product, and it remains to be seen if it, or programs like it, will find acceptance as a regular part of the digital artist or designers toolset. However, as a fully featured program incorporating many of the best ideas and techniques of computer-based sketching, Sketchbook Pro offers a glimpse of the kind of user experience that general computer sketching may have in the future.

#### **2.4.3.2 Research Applications**

Although specific support for sketching has been wanting among commercial programs, in the academic and research community computer based sketching is a topic of some intense interest. Many researchers have recognized the important role that sketching plays in the creative process, and have developed prototypes or applications designed to cater to the needs of a sketch artist. However, the goals of these projects is not only to support the ideas and techniques of traditional sketching in this new medium, but to apply the features and abilities of the computer to make sketching easier, more expressive, more productive, and possibly more focused to the specific needs of each individual sketch artist. In some cases this entails designing systems that let the computer do some of the work, functioning almost like a design assistant. In others this means that the system offers virtual tools that take real world supplies like rulers, tracing paper, glue, or templates and transmute them into idealized digital versions.

Although many fields use the same basic traditional sketching techniques, for some applications certain aspects of the drawing can be more or less important than others. This distinction is especially apparent when we compare the sketches

made by those in engineering fields to sketches that might be produced by an artist or designer. For engineers, architects, and some scientists, geometric aspects of a drawing such as the straightness of lines, the measure of their lengths or angles between them, and properties of their relative positioning can be important. However, addressing these issues accurately in a traditional sketch can be difficult because of the careful measurements and specialized tools that are usually required. Using these techniques may result in a more accurate drawing, but they slow down the process and disrupt the fluidity that is an important part of sketching.

Traditionally, to address these issues an engineer simply begins by creating rough and inaccurate sketches, and using those as a guide only later does he or she draft more detailed drawings using the appropriate tools. The drawback to this solution is that without some degree of accuracy in the early sketches, it can be difficult or impossible to develop a practical solution to the engineering problem. If for example the artist needs to design a machine part to fit within tight spatial constraints, any design that does not start from carefully measured parameters is less likely to produce a viable solution. Current drafting and CAD software can aid the user in this area by allowing constraints to be specified, or stock shapes like circles and squares to be created quickly. However, because the interaction is no longer one of drawing but of selecting and arranging components and their constraints, the fluid nature of drawing is lost.

Recognizing these issues, a group of researchers of the University of Tokyo proposed that the burden of dealing with measurements and tools could be lifted from the user and handled by an intelligent application. Igarashi *et al.*'s investigated how a 2-dimensional drawing program could be developed that would allow



the user to draw in an uninhibited way, just as they would while quickly sketching out a design, while the computer system would assist them by cleaning up the drawing as they work [Igarashi *et al.*, 1997a]. In effect the system eliminates the need for an intermediate step where the sketch is cleaned up by providing the user with the power of alignment and measurement tools that directly respond to the drawing.

In an initial study the research team observed subjects as they were asked to perform drawing and design tasks with a standard object oriented drawing program [Igarashi *et al.*, 1997a]. The subjects' progress was carefully tracked and each moment of their drawing session classified under a number of general drawing tasks like planning, drawing, or correcting mistakes. The researchers found that a significant amount of user time was spent in what they term 'cognitive planning' in which the user is trying to plan out how best to achieve their goals using the available features in the program. They reasoned that current programs create a 'cognitive overload' by offering such a wide variety of features, and that by reducing the user's need for such planning the efficiency of the drawing process could be improved.

Based on their observations, Igarashi *et al.* developed a prototype drawing application called Pegasus,<sup>15</sup> that used a system called 'interactive beautification' to extract constraints from users' input strokes and automatically assist them in drawing a clean and professional diagram [Igarashi *et al.*, 1997b]. The program analyses the strokes already entered by a user and identifies relationships between them such as symmetry, perpendicularity, connections and alignments, etc., and then uses that set of constraints to adjust subsequent input.

---

<sup>15</sup>According to the researcher's publication, Pegasus is a charming (b)acronym for Perceptually Enhanced Geometric Assistance Satisfies US!

Pegasus’s user interface system is also of note. It featured a *suggestive interface* in which the program’s logic creates a set of possible interpretations for constraint of a particular input, and then displays each to the users, in this case as in-place dotted lines, allowing them to choose the proper interpretation. Igarashi and his team later extended the system to include a predictive drawing mechanism that could not only create constraints on future strokes, but also detect larger symmetries of multiple strokes and add the matching geometry automatically, even suggesting transformations such as flips and rotations [Igarashi *et al.*, 1998].

Whereas Pegasus’s focus was on the production of a precise diagram from a user’s drawing commands, catering to a specific use of drawing in the design process, other research applications have tried to embrace the scattered and ambiguous nature of sketching. A good example is provided by the Electronic Cocktail Napkin, a project developed by Do and Gross. As its name implies, the project was intended as a general thinking tool for designers, architects, and engineers who scribble simple notes or diagrams on scraps of paper. In a similar manner to Igarashi *et al.*’s intentions with Pegasus, by channeling those drawings into a computer program, Gross and Do hoped to create an ‘intelligent paper’ that could infer the context behind a user’s sketches and offer tools or resources.

The foundations of the work were derived from observational studies conducted by Do [Do, 2005], which we touched upon in Section 2.3.1 on page 34. Her intent was to understand how sketch is used in many different contexts and how a computer assistant-like design tool could be developed to aid its users. Do concluded from these studies that within the architectural and design domain there is a conventional and consistent vocabulary of diagrams. Do also observed that the design task as a whole can be divided into a collection of sub tasks, each

of which is best addressed by their own set of tools, and each of which is identified by their drawing conventions and symbol sets. This implies that a computerized assistant could be developed to recognize a standardized set of design elements and symbols and use that limited contextual knowledge to interact with the designers on a higher level by offering them appropriate tools.

Based on these and other findings from empirical studies [Do & Gross, 1996], Do and Gross developed the Electronic Cocktail Napkin as a prototype design assistant [Gross & Do, 1996] based on a design philosophy of “the right tool at the right time”. The program makes inferences about a user’s design intentions based on the input he or she draws, and then offers the user tools or options tailored to that specific design task. For example, the user can begin by drawing freeform shapes into the program. When the user pauses, the system applies low-level recognition algorithms to the marks that can identify simple glyphs like circles, squares, and lines based on a number of attributes. Once lower level symbols are recognized, a high-level recognition system tries to infer the user’s intentions from the collection of symbols and their orientations and spatial relationships. Based on that information, the system might guess that the user is, for example, drawing a connected graph, a floor plan, or a circuit diagram, and offer appropriate tools.

To deal with the ambiguities that inevitably arise, the system maintains lists of possible interpretations for each glyph, and once a larger context is established, attempts to coerce interpretations from ambiguous input based on the contextual information. The system also imposes a set of constraints based on the context that allow the user to interact with the drawing while maintaining its meaning. Thus a sketch identified as a graph might enforce connectivity, and maintain connections when nodes are repositioned. Finally, to keep the user apprised of the

system's understanding, the program displays textual labels alongside elements of the sketch it recognizes.

Once a context is developed, the system can pass the sketch off to any number of secondary programs that perform tasks specific to the context; the article highlights several. For example the sketch could be recognized as a lighting diagram, in which case the image can be exported to a lighting simulation program. The input could also be recognized as a mathematical expression that might in turn be evaluated by a calculator utility. The image could also serve as a search query into a database of design ideas and images to offer the designer inspiration for a design task. The system even allows the user to develop their own set of glyphs and contexts including high-level symbols that abstractly define more complex and detailed groupings of glyphs. By substituting these abstract symbols into diagrams the user can adjust the level of detail, or can define complex components with minimal input just as they would in a hand-drawn sketch.

The system developed by Gross and Do is primarily focused on architectural and interior design, although by tuning the recognition systems to a different set of symbols and conventions, it seems likely that a similar interface could be a boon to many disciplines. However, the system is dependent on its ability to recognize a predefined set of symbols and to infer the user's intentions from their use. This model is most efficient in design tasks that involve the arrangement and manipulation of a set of existing primitives. On this level the application is more akin to a visual programming interface, taking as its syntax a visual arrangement of symbols. For lower level design tasks, such as creating the shape of a new automobile or cellular phone, this system is limited by the novel nature of the designed output.

Though engineering and architectural tasks play a prominent role in many of the research applications, there are also examples that lean more towards traditional artistic uses of sketch. A 2001 project by Tolba, Dorsey, and McMillan focused on the field of perspective drawing [Tolba *et al.*, 2001].

Perspective drawing is a method of rendering objects that more closely approximates the illusion of distance in the scene by reducing the size of an object based on its distance from the viewer. Perspective drawing requires careful attention to the relative sizes and angles of objects in a drawing to maintain the illusion of depth. Though this can be achieved by sight alone, artists, designers, and architects are trained to use a system of geometric guide lines converging at one or more points called *vanishing points*.<sup>16</sup> By rendering straight lines in the image to follow towards these vanishing points the relative sizes of objects at the same distances are automatically aligned. Although this method makes perspective drawing easier, the process of defining and following the guides is laborious [Tolba *et al.*, 2001], and usually much more involved than simple sketching.

Tolba *et al.* developed a drawing system designed to make perspective drawing easier on the artist by using the computer to handle some of the grunt work. Rather than storing the user's strokes as 2-dimensional points, input is represented internally as projective space coordinates, which can be thought of as the result of projecting the original points onto the inside of a sphere with the viewer standing at its center. The strokes are then re-projected from the sphere onto a viewing plane presented to the user. By adjusting this plane, the view can simulate single-, two-, or three-point-perspective views of the same artwork, even transitioning between them. The user can also define vanishing points and have the computer

---

<sup>16</sup>This system of perspective drawing was originally developed and popularized by the Florentine Architect and sculptor Filippo Brunelleschi in the early 1400's.

generate a perspective grid to aid him or her in drawing. Although most strokes are entered into the system freehand, the program can also constrain straight lines to follow the generated guides, and even create simple perspective primitives like rectangles that follow the user's grid.

Along with simple drawing guides, Tolba *et al.*'s prototype also includes features one would expect only to see from a 3-D modeling application. Because perspective drawing is often used to render architectural subjects, the images contain a large number of repeating elements. To facilitate their creation more easily the program allows simple transformations and duplication of existing elements. Primitives like lines and rectangles can be moved through translations or rotations similar to those seen in 3-D modeling applications. The program can also do limited sweeps and extrusions of objects to produce solid looking forms. The application even allows users to define point light sources, and can apply surface highlights and shading to the image, or even generate shadows. Once an image is created, or as portions are still being defined the artist can make limited changes to his or her viewpoint, panning and tilting the virtual camera.

Because the researchers' system does not store distance information in the way a 3-D modeling application would, but only simulates the appearance of distance, some of the 3-D features are limited. Still, by relieving the artist of the tedious task of setting up and maintaining a set of perspective constraints, Tolba *et al.*'s application allows the user to focus their energy on the creative task. This is a good example of how, through the computer, the horizons of what is possible with sketching can be expanded.

#### **2.4.4 Limitations of Sketching with the Computer**

Computers seem to have staked their claim to a permanent and growing presence in the fields of art and design. From early applications that helped publishers with the drudgery and expense of image editing and layout, computer graphics has slowly filtered down into the hands of professional and amateur designers and artists who freely mix traditional techniques with the new abilities that computers afford. However, this trickle down process is far from complete, and only recently have software publishers and researchers begun to address the more ambiguous stages of the creative process, developing software that caters specifically to sketching. Given the importance sketching has to the creative process there is little doubt that this area of research will continue to grow, providing computer users with an ever improving computer sketching experience that may some day reach or exceed its traditional counterpart. Most of the limitations of current computer graphics software that make sketching with the computer difficult are merely the growing pains of a market that is just now figuring out what constitutes computer sketching. However, even with the brightest of prospects, there are a number of more fundamental limitations to computer sketching that may be difficult or impossible to address.

The most basic is also the most obvious: computer based sketching requires a computer. One of traditional sketching's most positive aspects is its portability. Be it paper and pencil, sidewalk and chalk, or dirt and a stick, in almost any environment the dedicated, inspired, or simply the board can find the tools required to make a sketch. Chaining the sketch artist to a computer then, represents a major hurdle to both the expressiveness and acceptability of computer sketching. Hope to address this problem lies not in the software, but in computer hardware.

Just a few decades ago large desktop machines that offered their users power and flexibility in exchange for sedentary interaction dominated the burgeoning personal computer market. Although desktop machines still account for the majority of personal computers sold, laptop computers are slowly but surely closing the gap [Hook, 2003], and it is widely acknowledged that laptop sales will overtake the desktop market in the next few years if they haven't already. New form factors like the TabletPC are already hinting at what may be possible. As the power of portable computers increases and their costs and form factors diminish, your personal computer may take the place of the paper back book you carry wherever you go, making computer sketching as portable at least as a paper sketchbook.

Some researchers see the trend not in personal computers but infrastructure. Many authors in the human computer interface field feel that this issue may one day be resolved by the spread of so-called ubiquitous computing, where in computerized intelligence will be a standard feature of many everyday objects, becoming as close at hand in the future as pencil and paper are today [Kasik *et al.*, 2005]. Predictions like these are still some distance away, but new research efforts like Henzen's electronic ink tablet [Henzen *et al.*, 2005] may one day make ubiquitous computer sketching a reality.

Visions of a sparkling future aside, it is important to remember that computer based sketching is not intended as a replacement to traditional sketching techniques, but an extension from those creative skills to the computer. Seen from this perspective then it is perhaps more fitting to look at computer sketching as simply another medium at the disposal of the modern artist or designer; paper and pencil, stick and dirt, pavement and chalk, tablet and stylus.

A second limitation of computer sketching is its cost. Professional computer



graphics applications are some of the most expensive software packages available to the average consumer, ranging from several hundred to several thousand dollars apiece. Hardware costs too can be a major concern as graphics applications can be particularly taxing. Major offerings from corporations like AutoDesk, Corel, and Adobe generally require the most up to date computer systems and components to achieve proper performance. Recognizing this issue, some companies have stepped in to offer stripped down versions of their professional level software for the consumer, while other companies offer mid-range programs containing most of the features a hobbyist or student would require. There are also a number of mature, free and open source software alternatives that include many of the same features as their commercial counterparts. The most well known example in this market is the open source GNU Image Manipulation Program (GIMP), a photo editing program with a feature set commensurate to Adobe's Photoshop [Kimball & Mattis, 1995].

It remains to be seen if software targeted specifically at sketching will follow these trends. If the current computer graphics market is to be taken as a model the initial costs may run quite high, but consumers may eventually have a variety of choices.

The final limiting factor of the applications we've discussed is not so much a limitation of existing technology, but a division in the way we think about computer graphics and visual art and design in general. All of the applications and concepts we have discussed thus far deal exclusively in generating 2-dimensional images. This makes sense because artworks in 2-dimensions accounts for a large part of what we see on a day-to-day basis. Books, magazines, posters, and paintings, all of these items are strictly 2-dimensional. So too is traditional sketching

because it forms a basis for these other 2-dimensional art forms.

However, as we have seen in discussing topics like perspective drawing, 2-dimensional art is intimately concerned with the 3-dimensional properties of the subjects it's used to depict. The desire to work with this information is further reflected in computer based drawing programs like Tolba *et al.*'s Perspective Drawing System [Tolba *et al.*, 2001]. In some cases drawings can suffice to render this information, but there comes a point when it simply makes more sense to deal directly in three dimensions. The artistic term for these constructions is 'model', and the discipline of making them, both by hand and in the computer, *modeling*. The distinction between 2- and 3-dimensional work has been a reality of the art and design community for centuries, but as we will see, with the application of computers the distinction between working in 2- and 3-dimensions is becoming increasingly arbitrary as techniques from both disciplines, including sketching, are applied simultaneously.

# Chapter 3

## Modeling Background

Although a traditional sketch may try to represent depth or 3-dimensionality, the effect is only an illusion—the skillful application of perceptual tricks by the artist to activate certain spatial cues in the viewer’s brain. Sketching is necessarily a 2-dimensional activity because of the physical limitations of paper and pencil; there’s simply no way to draw *into* or *out of* the page. This same dichotomy extends though the visual arts, dividing mediums like drawing, printmaking, and painting from sculpture, carving, and architecture.

This is not to say that some works don’t bridge this gap between surface and volume, however the techniques of carving, building, and sculpting are very different physical activities from drawing and painting, and so serve as a logical reference point from which to differentiate the two fields. This same division exists in computer graphics, dividing 2-dimensional art and design subjects from 3-dimensional modeling, which produces output that defines the three spatial dimensions of a subject.

In this section we will discuss what constitutes model building, and how models are used by various professions. We will look briefly at the traditional techniques,

but focus on how computers have transformed much of this work, just as it has for sketching and drawing. As with the previous chapter, the information presented here is by no means a complete picture of model building. The purpose of this information is two fold. First, we hope to lay down a basic foundation of terminology and techniques so that we can more freely discuss topics of modeling later in this paper. Our second aim is to prime the mental pump, so to speak, and stir the reader's imagination about the current state of modeling, and how it relates to sketch.

### 3.1 What Is Modeling?

Much like the term sketching, 'modeling' carries a variety of related but distinct meanings in English—from the act of defining the depth and volume of a shape, to acting as one while wearing a minimum of very expensive clothing. In the present context, we will take the term *modeling* to mean the creation or definition of the 3-dimensional shape of something. The output of this process we call a *model* because it is a secondary representation of the real or imagined genuine article.

Modeling, like sketching, is a fundamental stage in many design processes. However models differ from sketches in that creating a model is a much more labor-intensive process. The time it takes to build any given model will depend on the materials or techniques used and the level of detail the model builder wishes to convey, but in general making a model will require much more time and effort. One reason for this is that a 3-dimensional model simply contains more information than a 2-dimensional sketch. Whereas a sketch can focus on a number of isolated elements of a design almost exclusively, model builders don't have this luxury. Because even simple models have a higher level of complexity and take

time to create, model builders rarely jump directly into creating a model, but instead begin with some kind of plan so that costly mistakes can be avoided. This means that modeling usually takes place after an initial sketching stage where ideas can be formulated and made more concrete.

It's tempting to think that because modeling is a later stage in design that this means model building does not contribute to the *creative* process to the same degree. In some ways this is true. Building a model, either physically or virtually, does not have the same ambiguous and ephemeral qualities of sketching. After putting a good deal of time into a model it's much more difficult to simply scrap it and start over, and the time involved in making the model means that cranking out dozens or hundreds of test models for a single project is a major undertaking. However, where modeling falls short in flexibility and quantity, it more than makes up for it in feedback and quality. Despite its fantastically dynamic nature sketching is still a 2-dimensional activity and so the degree of spatial information that can be extracted from one or more sketches is limited. By converting an idea into a 3-dimensional model the artist or designer can look at a design from different angles and solve problems that are not obvious from a static 2-dimensional projection.

## 3.2 Uses of Modeling

Model building is practiced in any situation where the 3-dimensional characteristics of something need to be investigated. In most cases models, like sketches, serve as preparatory work for a later, larger, or more complicated project. One field that makes extensive use of models is industrial design. Despite the 'industrial' moniker, industrial designers are responsible for creating the look and feel

of countless numbers of consumer products, from salt shakers to bar bells. Industrial designers consider every aspect of a new product from its shape and color to the materials it's made of, and even the packaging it's sold in, in order to make the product easier to use, more effective, or to entice consumers into an impulse purchase. Even slight changes in the ergonomic shape or tactile feel of something like a cellular phone or an electric toothbrush can have a huge impact on the product's success, and so design firms produce dozens or even hundreds of mockups, carefully adjusting the shape and form, until the new design fits comfortably into a user's hand.

Models also provide a convenient means to communicate the feel or scale of a project to a third party. It's likely, for example, that a sculptor working on a piece of public art intended to be placed in, say, a public park or near an office building, would need to present a model to the local planning committee to have the design approved. This allows the committee to visualize what impact the full scale design will have on the space it is intended to occupy, and to see how it might effect things like traffic or access for emergency personnel.

For the same reason, it can also be helpful to create models of architectural structures like new high-rise buildings, shopping centers, or housing developments. These models not only provide city officials with information about the proposed construction, but also serve as advertising tools to perspective clients who might be interested in investing in the project. For architects, creating scale-models of proposed buildings is part of standard operating procedure, and architectural students are trained to construct models as part of their education. Architecture firms create both physical and increasingly digital models to help their clients visualize the plans for new buildings [Tolba *et al.*, 2001].

Not all modeling is done with the intention of creating a final product that is 3-dimensional. A good example can be seen in the animation industry. In order to describe the dynamic movements of a cartoon character in a scene, animators must have an intuitive understanding of the physical features of that character from many different angles. This is usually accomplished with a modelsheet [White, 1988] [Patterson & Willis, 1995], a page on which the character has been drawn from a series of angles and in various poses and emotional expressions. For some characters a single model sheet is sufficient to give the animators a clear picture of what they're trying to draw, but when the character is particularly complex, or needs to be drawn from many different angles precisely, a 3-dimensional model can be more effective. In some cases, a 3-D model of the character is created in clay or some other material to give the animators a better idea of its physical form. If the cartoon will incorporate digital animation as well, these clay figurines can later be scanned into the computer to make digital versions of the character suitable for animation, prototyping, or merchandizing designs.

A similar argument can be made for the digital models constructed for television and cinema. Here animators create 3-dimensional models of characters, objects, and scenery so that they can be combined with live action footage in a process called compositing. Of course when we go to see the movie in the theater or watch the show on television the visual image we receive is a 2-dimensional projection. It may be tempting to think that making 3-dimensional models in these situations is unnecessary. Although it is possible to draw these elements in 2-dimensions like a cartoon, when combined with photographs of actors in each

frame the animated elements stand out as unnatural and flat.<sup>1</sup> If the director's goal is to suspend the audience's disbelief—to make them really feel they are seeing a giant alien creature, dinosaur brought back to life, or mile wide interstellar spaceship—the cartoon look can be distracting. Instead physical 3-D miniature models called *actuals*, or increasingly 3-D computer models, are created and shot as they move through the motions of a scene. That footage is then combined with footage of the real actors to generate the final effect. Because both the models and the actors have physical weight, depth, and dimension, on screen they appear to interact more naturally.

Aside from representing the geometric characteristics of an object, models can also be built to represent an object's physical characteristics. Models like these are used extensively in the design and testing of vehicles. We've probably all seen car commercials featuring a sleek line of smoke following the organic contours of a new car's body. Similar testing is performed—although usually in less photogenic surroundings—on scale models of cars in wind tunnels to determine how air will flow around them at high speeds. This technique was originally pioneered by the aviation industry, which still uses extensive wind tunnel testing to design and develop new wing and body shapes with favorable aerodynamic characteristics.

For almost any kind of vehicle from high tech Martian probes or research submarines to the family sedan, small changes in shape can have drastic effects on the way air or water flows around an object. This information makes sexy looking commercials to be sure, but for something like a long haul truck, or a container ship these tiny differences translate to changes in top speed or fuel

---

<sup>1</sup>For some moves this animated look is actually what the director has in mind. A technique like this was used to add cartoon characters to live action footage in movies like *Who Framed Roger Rabbit* [Contributors, 2006c].



economy, both of which have serious economic repercussions.

Some of these characteristics can be calculated from simple technical drawing, however physics still struggles to satisfactorily explain the dynamic motions of turbulent fluids [Feynman, 1963], and so in some cases a physical model is the only option.<sup>2</sup> Models can also be constructed from materials a designer is considering for use in a final design. Models like these provide researchers with valuable information about the final weight, volume, strength, heat dissipation, electrical properties, and so forth of a design.

For some professions, the value of a model is its ability to demonstrate or replicate the mechanical characteristics of a design. This is especially true of models built by engineers and inventors, who often build models that simulate the mechanical actions of their designs. Sometimes these models can be quite intricate but at the other extreme, engineers are famous for concocting Rube Goldberg like contraptions out of scavenged parts and kitchen utensils held together with little more than duct tape, bailing wire, and good intentions.<sup>3</sup> Along with rapid prototyping, model building can be a valuable educational tool for engineers. By building models, engineering students can put the principles and theories they are studying into practice and gain valuable experience in solving real-world problems.

For many engineering institutions and professional organizations it has become a

---

<sup>2</sup>Modern computers have made it possible to simulate some aspects of fluid dynamics, but only at great computational expense. In most cases it's simply easier to build a small model of the proposed design and take empirical measurements.

<sup>3</sup>Surprisingly, one of the most popular media for building this type of contraption is a common children's toy: Lego bricks [Parks, 2005]—a popular brand of small interlocking plastic building blocks produced by the Danish LEGO Group. Legos come in many shapes and sizes that can be used to build any manner of structure, but more importantly among the cornucopia of bricks are a wide assortment of gears, shafts, wheels, pneumatics, hinges, and other mechanical components. More recently the LEGO Group in a partnership with MIT's Media Lab released a set of special bricks that allow a builder's model to be interfaced with a personal computer [Contributors, 2006f].

popular practice to sponsor model-building contests for students.<sup>4</sup>

While most models are used in a preparatory capacity, models are also useful abstractions for considering spatially complex problems. Many disciplines use simplified models of existing objects to help researchers reason about a problem more easily. For example, medical students and professionals through the ages have utilized anatomical models to help them learn the structures of the body without the need for an actual cadaver to examine. Working medical professionals continue to use these same models in their practice as an easy way to discuss the details of diagnoses and treatments with their patients. Referring to the model helps to communicate the complex matters at hand more directly.

For this same reason, models are often used as explanatory or exploratory tools in other situations. Procedural crime dramas on television or in the movies have popularized the ‘CSI treatment’ of a crime scene where the location is rendered as a 3-D model showing the positions of evidence and tracking the paths of bullets. While Hollywood’s depictions of the investigative process tend to push the envelope beyond what most state and federal agencies could afford or technology could create, scenes like those in the movies are at least founded on real life examples. The National Transportation Safety Board for example has software that interfaces with flight data recorders recovered from crashed aircraft that allow them to automatically generate a 3-D computer model of an aircraft in its final moments of flight. The National Aeronautics and Space Administration also make extensive use of 3-D computer animations to both plan space missions and to ex-

---

<sup>4</sup>Several well-known engineering competitions are held annually that ask students to build structures from seemingly weak or inappropriate materials. Many high schools and universities sponsor bridge building competitions in which students construct model bridges from thin strips of balsawood or dried pasta. The American Society of Civil Engineer’s annual concrete canoe competition asks students to construct life-size buoyant canoes made of concrete [Odson, 2006].

plain them to the public. Similar animation techniques have been popularized by science shows and the news media to show reconstructions or interpretations of crimes and accidents from the J. F. K. assassination to the second Iraqi war.

As you can see, model building, like sketching, is an important technique in many disciplines. Whereas sketches provide an artist, designer, or engineer the ability to develop their ideas, models are a way to take those primitive designs to the next level where the focus is shifted from developing a number of discrete elements to creating a cohesive whole.

### **3.3 Limitations of Physical Modeling**

First and foremost model building is a messy and labor-intensive process. In professional settings model builders usually work in dedicated studios where supplies and tools can be close at hand and the mess can be contained. Weekend hobbyists may be lucky enough to have a garage or basement to work in but those who can't are usually resigned to commandeering a room until forced to relinquish it by a spouse or visiting company. In any case, model building lacks the ease and portability of sketching, so building models on a whim becomes more difficult.

Model building can also be an expensive undertaking. Unlike sketching, which can be done with items readily at hand, model building usually requires specific tools and supplies. Some of the expense can be mitigated by using cheap or easily obtainable materials, however these material rarely have the same physical characteristics as the materials they are intended to represent, which limits experimental information that can be provided by the model. Model building also requires an array of tools to properly work with the modeling materials. Also because constructing a model can take hours or even days a single model can

represent a significant investment.

The cost and construction process for a model also has a large effect on the ways in which it is used. Because of their expense designers might be reluctant to subject a model to tests that could damage or destroy it. Even when models are constructed specifically for the purpose, any one model can only be used in destructive testing once, which limits the number of tests that can be performed. Furthermore, under even the best of circumstances a model can only provide experimental information about the model itself. Many physical principles do not scale linearly, and so although a model may be constructed of similar or identical materials to a proposed final design, if the model is at a different scale or incomplete, results of experiments performed on the model do not provide researchers with a full picture of the design's actual performance.<sup>5</sup>

Finally, traditional models present a number of difficulties as a result of their physical nature. Although we can easily create copies of 2-dimensional materials like drawings or photographs, there are still no convenient and inexpensive ways to make replicas of 3-dimensional objects. This means that every model must be constructed by hand. This also makes it difficult to share a model with people in different, disparate locations. The 3-D form of models also means they can be difficult to store. Whereas sketches, drawings, or photographs are flat and can be filed away efficiently, 3-D models usually cannot be stacked, folded, bound, or otherwise packed without considerable work or possible damage.

Obviously, none of these issues are severe enough to prevent artists, designers, architects, engineers, or hobbyist from constructing models on a regular basis.

---

<sup>5</sup>This is one reason why agencies like the National Highway Transportation Safety Administration and organizations like the Insurance Institute for Highway Safety perform automobile crash tests on actual cars rather than scale models, despite the obvious expense.

However, though these issues do not preclude the use of traditional modeling, they do place limits on how the craft of modeling is performed. As we will see, computer based modeling offers solutions to many of these issues.

### 3.4 Modeling With the Computer

Computers are now used extensively in the design of all manner of objects from ordinary consumer products, machine parts, and equipment, to large or staggeringly complex engineering tasks like aircraft, skyscrapers, and CPU designs. Whereas traditional physical models have been a staple of the design process for thousands of years, the speed and flexibility of computer technology had steadily eroded traditional modeling techniques in these fields [Tolba *et al.*, 2001]. Just like physically constructed models, computer models provide a simulation of a design that allow designer or their clients to better understand the form or features of a system before the expense of actually constructing it. Where computer models excel is their versatility as representations of the actual article.

A good example of the versatility computer models can offer is provided by the design and construction of Disney's California Adventure Park, an addition to the popular southern California attraction that debuted in February of 2001 [Jung *et al.*, 2002]. As described by the New York Times [Taub, 2001], during the early planning phases the designers of the new attraction considered building a large wooden model of the new facilities, but were dissuaded by the staggering size and expense of a model describing the entire 54-acre park. Instead, Disney's architects and designers created a virtual model allowing designers, executives, and local officials to fly through the new park and view it from any angle in various stages of completion. Designers were able to discover problems with the layout

of buildings and designs of attractions in seconds, and were able to make changes quickly, something that would have been expensive with a traditional wooden model. The model was also instrumental in negotiations with local government.

Computer models directly address many of the drawbacks inherent to traditional physical models. They are easy to store and transmit; they can be destroyed and recreated as many times as the designers wish at no added expense; and they can be imbued with physical properties to mimic the behavior of their real-world counterparts. It's also, in general, much cheaper and easier to construct a model in the computer. No special workspace is required, nor a studio full of tools and a mountain of supplies other than a suitable computer system. What's more, once completed, the computer model can serve a dual purpose as both a testing platform, and as a source for planes or blueprints describing exactly how the genuine article should be constructed. In fact, in many cases models for items like machine parts can be entered directly into manufacturing equipment to construct the desired parts [Bloomenthal *et al.*, 1998].

Just as the case with 2-D drawing, 3-dimensional computer models have also made inroads in the artistic, graphic design, and entertainment industries. To the general public, probably the most visible example has been the explosion of 3-D effects used in television and movies. Animation sequences developed with 3-D modeling techniques began to find their way into traditional animated movies like Disney's *Beauty and the Beast* [Contributors, 2006a] in the early 1990's. Since then, 3-D animation studios like Pixar and Dreamworks now generate features animated entirely in the computer on a fairly regular basis. Computer generated models are now also a mainstay of live-action cinema where 3-D animators and designers strive to create models so realistic they blend in with real actors and

photography.

### **3.4.1 Current Computer Modeling Techniques**

The techniques and representations used for computer modeling are almost as varied as the applications to which those models are applied. However, on a basic level the computer hardware that is used to display 3-D objects can only display a narrow set of geometric primitives. These include vertices, straight lines, and planar convex polygons. Thus, no matter the internal representation of the model, or the techniques used by the artist or designer to develop it, in the end the modeling application must provide the hardware with a representation of the model built completely from these components.

Early modeling applications exposed this level of granularity directly to the user, providing a working environment in which the artist directly manipulates geometric primitives or simple constructions there-of. For some applications those techniques are still appropriate, and features developed early on are still wildly seen in modern applications. However, as the speed and processing power of both graphics hardware and computers themselves have increased, new methods have been developed to introduce layers of abstraction that aid the user in creating more complicated models that use these primitive elements behind the scenes to approximate more complex forms. The following sections provide a brief glimpse of some of the most popular techniques.

#### **3.4.1.1 Construction from Primitives**

The most straightforward method of both building and editing 3-D models is to work directly with the geometric primitives. In many ways application that

function at this level are not unlike the 2-D dimensional object oriented drawing programs or technical illustration applications; the user is provided with a basic set of tools to generate points, lines, and polygons, and from these basic units more complex models are derived.

Although many programs are based on these simple primitive components, working at such a fine grained level can be extremely tedious, even from the most simple shapes, and so most applications also provide a catalogue of standard 2 and 3-dimensional blanks that can be called up and generated based on the user's parameters. These blanks act like basic building blocks for larger models, and might include simple boxes, spheres, cones, cylinders, pyramids, and others. Although these programs offer their users blanks that include rounded shapes, these are only approximations to truly rounded volumes, and represented by a patchwork of flat faces.

Once a basic framework of geometry has been established, the user develops his or her model by manipulating the primitive components with a simple and direct point-and-click interface. In many ways, this style of construction can be thought of like building a structure from toothpicks and gumdrops or tinker toys. The designer directly manipulates vertices (gumdrops) and edges (toothpicks) to create larger structures, and then pushes or pulls at the gumdrops to change the model's shape.

Internally, the application stores the model as collections of geometric information describing the 3-dimensional locations of each primitive component. This representation is often referred to as a *mesh* because the collection of vertices and edges form an unstructured grid describing the model's surface. For simple models this information can be stored in simple indexed lists of vertices, edges, or faces.



However as the complexity of the model grows,<sup>6</sup> these simple representations can quickly degrade into a ‘polygon soup’.

To provide a more organized picture of the data, many systems use more complex data structures that, along with this basic geometric information, include data regarding the topological relationships between the geometric components. These relationships might include, for example, which edges are incident to each vertex, which planar faces neighbor each other on a surface, or which vertices form the boundary of a face. By storing this information in a highly interconnected structure of pointers, algorithms can be applied that literally ‘walk the surface’.

In the tradeoff between granularity of control and ease of use, primitive-based point-and-click style modeling is a decidedly granular way to work. Depending on the application and the complexity of the final model, this can be both a boon and a curse. Computer aided design applications used to generate engineering schemata and blueprints for example rely heavily on these sort of basic tools because they are well suited for constructing the highly regular shapes of machine parts and mechanical devices. Artists and industrial designers on the other hand are offered little in the way of features to generate organic shapes or sweeping curves, and so modeling environments targeted at these markets rely more heavily on other methods. However, even for fully featured modeling applications that incorporate a variety of construction methods, primitives and point-and-click manipulation still form the kernel of many modeling systems.

---

<sup>6</sup>Some mesh structures may contain millions or hundreds of millions of triangles. Just think how many gumdrops you would need!

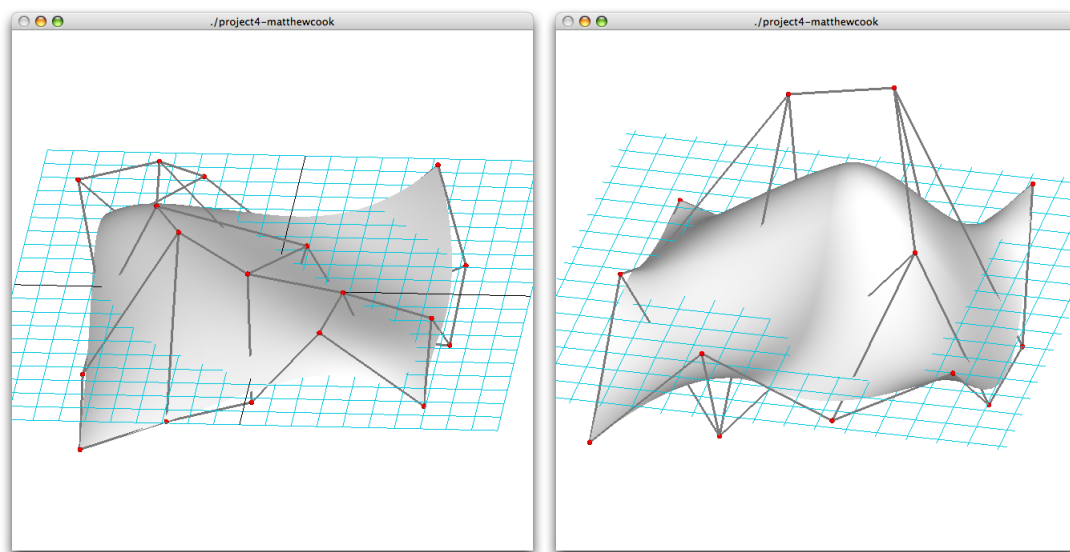
### 3.4.1.2 Curves, Surfaces, and Control Point Manipulation

A particularly weak aspect of primitive-based modeling is its poor facility for representing organic curves and smooth flowing surfaces. Even for mechanical parts shapes like circles, cylinders, and spheres are common elements that appear over and over in designs. Approximating these shapes with finely sub-divided straight lines and flat surfaces may suffice for visual displays, but they lack the physical and mathematical properties of their ideal counterparts, properties which may be important if the model is to be used in simulated environments, or as input to automated manufacturing equipment. Smooth curves are also important to industrial designers and artists who need to model not only regular cones and spheres, but freeform organic shapes. To cater to these design requirements it's often useful to represent the curves and surfaces of the model using mathematical rather than direct geometric descriptions.

To provide a more expressive way to model with curves and surfaces, parametric representations were developed [Farin, 2002]. Although a number of representations are in widespread use, they all begin with the same basic premise. A curve is described by a combination of control points—individual vertices in two or three dimensions which may not lie directly on the curve, but act as constraints, pushing or pulling on the curve's path. The pull of each control point on the path of the curve is determined by a set of basis functions. When supplied with a parametric parameter, the functions distribute influence between the control points, determining the degree to which each control point effects the curve. The linear combination of the control points, each scaled by these influence factors, form the coordinates of the point on the curve corresponding to the parametric value. By evaluating the basis functions at regular intervals over the entire para-

metric range, the path of the curve is completely and efficiently described. Bézier curves, B-Spline curves, and non-uniform rational B-Splines or NURBS curves are all common examples of parametric representations.

This entire process can be taken one step further by using a matrix of control points laid out as a grid in 3-dimensions, and two parameter values. Just as above, control points along each row or column define a curve that can be evaluated to a single point. These intermediate points, one for each row or column, then serve as control points of a second curve in the opposite direction and parameterized over the second parameter value. By evaluating over the parametric ranges of both parameters, a smooth surface is created—see Figure 3.1.



(a) Bézier Surface

(b) NURBS Surface

**Figure 3.1.** Two examples of parametric surfaces and the control meshes that define them. The image on the left is a Bézier surface of degree four with five control points in both the U and V parametric directions. The image on the right is a cubic non-rational uniform b-spline or NURBS surface with four control points in its U and V directions. The control vertices are marked with red spheres, and each is connected to its four neighbors with a straight line segment.

Differences between the various parametric representations mostly concern the basis functions they use. Some functions are more efficient to calculate, while others can represent special kinds of curves like circles or conic sections more accurately. However, all of the basis functions share certain properties that make them well suited to working in modeling systems. For example, most curve representations are affinely invariant, meaning that affine transformations like rotation, translation, and scaling, applied to the control points of the curve will produce the same results as transforming the curve itself. Most representations also exhibit a variation diminishing property, indicating that the path of the curve is limited to the interior of the curves *control polygon*, a structure defined by connecting the control points in sequence with straight edges.

The mathematical properties of the curve representations are reflected in the modeling system. For example, the variation diminishing property not only describes the behavior of the curve, but also reflects the way in which the user manipulates it. Because the curve's control points completely define the shape of the curve, they offer a convenient handle for interactive manipulation. By dragging the locations of control points the user can change the path of the curve.

The flexibility of the representation also stems from its mathematical underpinnings. Because the curve or surface is represented by a function rather than directly as primitive components, in order to display the model those functions must be evaluated into discrete points and then approximated by the familiar graphic primitives. At first this might seem like an inefficient way to display the model, but it means that the visual representation can be generated dynamically to fit the circumstances, ensuring that the curve contains only as many primitives as is necessary to appear smooth to the viewer. The mathematical representation

also translates to a substantial reduction in memory requirements. Because the curve or surface itself is completely defined by the control points alone, there is no need for the system to store all of the geometric information in a traditional mesh model.

Parametric curve and surface representations provide a powerful alternative to primitive based modeling methods. However, this increase in expressivity and flexibility is not without some drawbacks. Many, for example, point out that control point manipulation is not a very intuitive way to edit a curve [Henzen *et al.*, 2005]. Because of the variation diminishing property the general movement of the curve is not difficult to predict, but on a lower level it can take a good deal of practice before a designer can create a curve that exactly replicate follows the path he or she may have in mind. Another complication is the fact that as the complexity of curve's path increases, the number of control points needed to define the curve also increases. With more control points, even interactive manipulation can become bogged down because so many points need to be adjusted before the desired curve shape is attained. What's more, the large number of constraints can cause the curve to wiggle, ripple, or otherwise deviate from a smooth path— anomalies that can be difficult to remove by hand. As you might expect, each of these issues only multiplies in complexity as the designer moves from 2-dimensional curves to 3-dimensional surfaces. Despite these drawbacks, control point manipulation is a very popular interface.

#### **3.4.1.3 Procedural Modeling**

Whereas primitive modeling defines geometry directly from basic constituents, the parametric curves and surfaces in the previous section are abstracted, gener-

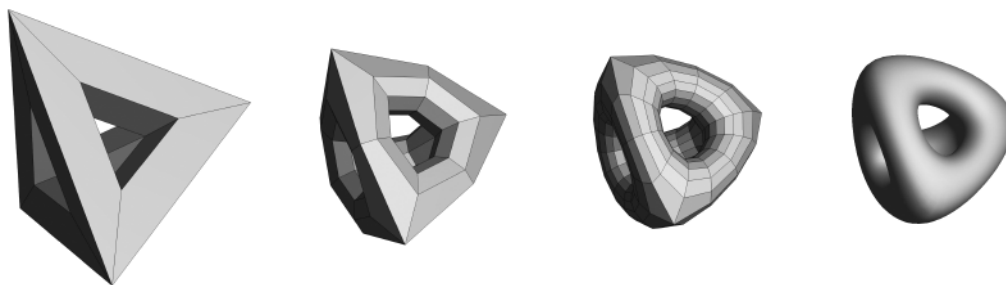
ating geometric information as the result of a process of calculation. By applying an algorithm or procedure, the modeling application can generate the primitives at a later date, adapting the display to the desired level of detail and reducing the memory requirements of the program. This basic idea has proven to be a popular one, and a number of modeling interfaces and common construction methods can be similarly described in terms of algorithms and their inputs.

Take for example the Play-Dough press you may have played with as a child. The press consists of a lever connected to a small holding area and a plunger that fits inside. As the lever is depressed, dough placed in the holding area is compressed and extruded through a small opening at the front of the device. The fun is in covering this opening with differently shaped dies so that as the dough is pushed through, it's extruded into interesting shapes like pasta or hair.

This same idea can be applied to a modeling interface, and works in a very procedural way. The designer supplies a 2-dimensional shape that is then swept through space to form a 3-dimensional volume with the original shape as its cross section. This operation is referred to, appropriately enough, as an *extrusion* or *sweep*. Supplying different cross-sectional shapes and sweeping paths can generate a wide variety of common 3-D models.

Whereas parametric curves and extrusions derive their effectiveness from interactive procedures, another procedural modeling technique call *subdivision surfaces* use a recursive method to generate smooth shapes from basic angular frameworks [DeRose *et al.*, 2000]. The idea behind sub-division surfaces is this. Consider, for example, a cube such as one created by the primitive modeling systems described in Section 3.4.1.1 above. The cube consists of six very flat faces and eight very pointy corners, and as such is in no danger of being mistaken for a sphere. Now

consider if we take each of the cube's face and subdivide it, creating new faces and new vertices, each positioned so that the new vertices lie on the surface of a sphere that encircled the cube. If we repeat this process several times, the resulting mesh will contain a much larger number of much smaller polygons arranged so that the harsh features of the original cube begin to melt away. The limit of this process, if allowed to continue, will define a mesh containing every point on the surface of the sphere, making it appear completely round—see Figure 3.2.



**Figure 3.2.** This figure from DeRose *et al.* demonstrates how subdivision surfaces are used to create a smooth shape from a simpler framework [DeRose *et al.*, 2000]. The image to the extreme right is the original control mesh consisting of 24 faces. To its right is the same mesh after one subdivision, and then two subdivisions. On the extreme right is the ‘limit surface’, which appears completely smooth.

This recursive subdivision procedure can be used to construct a model from a very basic and blocky 3-D framework, and then smooth the model to create a pleasing natural surface. Running this process to its limit to extract the surface is not computationally feasible, but it turns out that the procedure converges rather quickly, and after only a few subdivision-passes the visual results are more than sufficient to define a smooth surface. Because they offer a convenient middle ground between the full expressivity of parametric surfaces and simplicity of and compatibility with primitive construction methods and mesh data structures, many of the most popular 3-D modeling applications today include subdivision

surface features.

Subdivision surfaces also offer a number of advantages over other modeling methods. DeRose *et al.* provide the following three highlights, [DeRose *et al.*, 2000]. Subdivision surfaces can be used to model 3-dimensional shapes of arbitrary topology, including surfaces with holes in them, something that can be very difficult to achieve in a straightforward manner with parametric surfaces. The subdivision algorithms are also simple, relative to other methods for achieving rounded shapes, and run efficiently on modern hardware. Probably the biggest advantage however is the fact that multiple levels of model detail are built directly into the representation. This means that, for example, the modeling application can present the model with a limited number of subdivisions within the modeling interface, but then increase the subdivisions when a final rendering is generated. The low level-of-detail mesh can be easily rendered by even the most underpowered commodity graphics hardware, and provides the user with a snappy and responsive interface for interactive modeling without sacrificing full fidelity in the final model. This process can even be done dynamically, perhaps allowing the user to choose a low setting for underpowered systems and a higher one for top of the line models, or rendering at low resolution as the geometry changes, and then updating in the background while the user is idle.

At the far end of the procedural modeling spectrum are systems that provide not so much a modeling interface as a modeling programming language. Here, the entire process of modeling is expressed by providing the computer with a set of instructions that describe and generate the model in the same way that a sheet music describes a song, or a recipe can generate a birthday cake. This technology is especially useful for creating models of objects that contain repeating elements,



or self symmetry such as the geometric patterns in ornate architectural designs, or the fractal patterns of plants and trees. A good example of this technology is the GML programming language based on concepts of ‘generative programming’ developed by John Snyder [Snyder, 1992]. Although few modeling systems are based solely on a programming interface, because many procedural techniques work with common geometric primitives, primitive based modeling systems will often incorporate scripting languages or macro environments to augment an interactive user interface.

#### **3.4.1.4 Constructive Solid Geometry**

The modeling techniques we have discussed up until this point revolve primarily around creating single model components, but do not discuss how these components are combined to form a complete model. Very few models contain only a single component. Most are constructed of dozens or even hundreds of basic forms carefully arranged by the designer. For some applications, simply placing the model components in proximity to one another is sufficient to indicate their association. However, in other instances the topological representation of the model is as important as its visual appearance. This is especially true when the model is created to be used within a larger simulation, for example as a new aircraft or ship’s hull design in a fluid dynamics simulation, or a military vehicle that will undergo ballistics or electromagnetic tests.

One modeling technique that addresses these issues specifically is called Constructive Solid Geometry or CSG. Rather than building models from primitive components like points, lines, and polygons, CSG modeling systems are built upon 3-dimensional modeling primitives like plane, boxes, cylinders, and spheres.

Models are constructed by positioning and then combining these solid components using CSG operators, a collection of functions over the components that resemble the basic set operations like union, intersection, and difference. The operations and their arguments can be arranged into a binary tree structure in which leaf nodes are CSG primitives, interior nodes CSG operators, and the root node the expression of the final model. By performing only the operations within any given subtree, one can also recover constituent parts of the model in different stages of construction. This makes it extremely easy to swap in and out different configurations, or to save stock components and reuse them over and over within a design. As the components are visually combined, their structural representations are also effected so that the final product of the modeling process is a single connected structure representing both the exterior surface of the object, and the volume it displaces.

For certain classes of model, CSG based modeling is a particularly good fit. For example, most machine parts or mechanical fittings, though complex, are composed of simple shapes. Wheels, rods, gears, plates, stops, drill holes—each of these is easily represented by basic CSG operations. Furthermore, although the final product of the modeling task may be highly complex its design can be expressed simply by the CSG operations and primitives on which they act. In this way, CSG modeling can be thought of as another example of a procedural modeling method. These features make CSG models very efficient to store and transport, and a popular alternative to primitive-based modeling.

### 3.4.1.5 Implicit Modeling

One drawback to constructive solid geometry is the fact that calculating the results of CSG operations on primitive model components, which are in turn represented as mesh structures or other collections of the 3-D geometric primitives, is an extremely non-trivial task. For each operation, the precise intersections between the two possibly arbitrarily shaped modeling components must be calculated, a process that is fraught with tricky corner cases and numerical instabilities. To combat these issues, many modeling systems that utilize CSG style operators choose to use an alternative implicit mathematical representation for their geometry that describes not only the surface features, but the volume of the model more directly.

This form of modeling is often called *solid modeling*, or *implicit modeling*, or *volumetric modeling*, and is built around the fundamental operation of point classification: the ability of the system to state unequivocally for any given point in the modeling space whether that point is inside, outside, or on the surface of the model. One common method of achieving this system is to build the basic CSG forms as combinations of implicit functions. Many basic shapes such as spheres, infinite planes, and infinite cylinders can be simply expressed by implicit formulas. Other basic shapes can be represented as combinations of the above. For example, a cube can be constructed by combining six planar half spaces, each represented as an infinite plane. Similarly, a cylinder is an infinite cylinder truncated by two planar half spaces, a cone a conic section intersected with half spaces, etc. Using these representations, the CSG set operators that were once so complex to calculate for a mesh structure suddenly degenerate into simple mathematical inequalities between the implicit functions.

Half-spaces and sphere equations are just the tip of the volume modeling iceberg. Researchers have developed myriad methods of representing not only regular shapes by more general natural looking forms using various versions of implicit functions. These include methods like radial basis functions [Carr *et al.*, 2001], variational implicit surfaces [Turk & O’Brien, 1999] [Karpenko *et al.*, 2002] [Araújo & Jorge, 2003] [Karpenko & Hughes, 2006],<sup>7</sup> and convolution surfaces [Bloomenthal & Wyvill, 1990] [Wyvill & Guy, 1998] [Schmidt *et al.*, 2005; Tai *et al.*, 2004],<sup>8</sup> This field of modeling is relatively new, and in many cases the terminology becomes a bit muddy. Volumetric or implicit modeling method are also sometimes referred to as *level set surfaces*, *blobby surfaces*, *equipotential surfaces*, and *algebraic surfaces*.

The implicit functions themselves are sufficient to describe an entire model, however most graphics systems are unable to display them directly, and so the implicit representations must be converted to a secondary format for display. Several general methods have been developed that can extract from volumetric models the necessary geometric information. The most common algorithm is known by the rather upbeat moniker ‘marching cubes’ [Lorensen & Cline, 1987]. The marching cubes algorithm begins by dividing the modeling space into a series of equal cube-shaped sub-volumes called voxels, which can be thought of like a 3-dimensional analogue to the pixels on a computer display. The algorithm examines each cube in the volume and determines which if any are transected by the implicit surface by classifying its corner points as inside, outside or on the model’s surface. For those cubes touched by the surface, the exact points of intersection between the

---

<sup>7</sup>Variational implicit surfaces were originally developed to generate approximating surfaces for data visualization.

<sup>8</sup>Convolution surfaces define a volume by convolving an implicit kernel function along a network of skeletal paths.

cube and the surface are calculated, and triangles are generated to describe the surface within the cube. By combining the triangles from all transected cubes the entire surface emerges.

While this method produces good surface representations, it can be inefficient even for small volumes. Some systems choose instead to use a simple ray-tracing or ray-casting algorithm, which generates surface polygons only for visible portions of the model, or calculate pixels of a rendered image of the model directly, bypassing polygons all together. There have also been recent efforts in rendering models as a swarm of discrete painted points rather than connected geometry [Witkin & Heckbert, 1994]. Although each of these methods has met with some success, each has its drawbacks, and none is particularly efficient as the size of the model increases.

In general, the computational complexities of implicit or volumetric rendering methods have largely relegated them to the fringes of the commercial 3-D computer modeling market. Implicit models represent the far end of the abstraction spectrum from primitive based point-and-click modeling applications, and much of that abstraction comes at a very high computational cost. However, as the power and capabilities of graphics acceleration hardware and computer technology in general continue to advance, these barriers will fall and implicit modeling will likely move out of the research labs and into the hands of artists, designers, and, engineers.

### **3.4.2 Current Modeling Applications**

Unlike the market for sketching software, the marketplace of 3-D modeling software is flush with alternatives covering a wide range of price points and target

applications. However, more like the digital imaging market there are several large players that tend to dominate the 3-D modeling software landscape.

Although the distinction is not always a clear one, 3-D modeling software can generally be divided into two classes: computer aided design or CAD software, and visual modeling software. CAD software serves as a computational alternative to traditional technical drawing and drafting techniques, and is used by engineers, architects, and other technical professions to create highly detailed diagrams, plans, and schemata. To serve this market CAD applications provide features that make the process of defining the technical details of the model easy and efficient. The user can, for example, keep tight controls on the length of objects or the measure angles; components can be easily aligned, grouped, and replicated; and the interface provides helpful guides, grids, and snap points to ensure that elements meet cleanly or are symmetrical.

CAD applications are available over a wide price range, from simple consumer applications targeted at weekend hobbyist, to fully featured professional applications that can cost thousands of dollars. A dominant player in this field is AutoCAD [Autodesk, 2007a], a suite of applications and tools for 2-D and 3-D modeling produced by Autodesk Inc. AutoCAD is a professional grade computer aided design tool marketed to design, engineering, and architectural firms. Models in AutoCAD are constructed from a set of basic primitive shapes, using a point-and-click interface and set of standard tools, however the program also includes support for modeling with parametric curves and surfaces and some solid modeling features. Along with these basic components, users can draw from a large library of predefined stock parts and basic model shapes to use as templates in their constructions. AutoCAD is also highly customizable, and includes hooks allowing a

number of scripting and programming languages to interact with the application. This gives the program many of the capabilities of a procedural modeling system, and allows the program to be integrated into a professional workflow.

Given its large feature set, AutoCAD also comes with a steep learning curve, and an even steeper price tag. Never the less, AutoCAD and applications like it have become a standard tool in industry. Although AutoCAD may be outside the range of general consumers, other applications are available that offer some of the same functionality. A recent entrant into this field is SketchUp [Software, 2007]. SketchUp was originally developed by @Last Software but was acquired by Google in 2006. SketchUp provides a very basic set of construction features incorporated into an extremely user-friendly interface. Still based on a point-and-click system, the modeling interface incorporates innovative construction methods like ‘push-pull’, which allows users to quickly create solid objects from 2-dimensional shapes. The program avoids the steep learning curve of more traditional CAD systems and is targeted at amateurs and hobbyists who need a basic CAD system without the expense or complication of more professional tools. Although SketchUp’s feature set is not as impressive as AutoCAD, because a version of the tool is offered free to download, a strong and growing community of supporters has quickly sprung up around the program, using it to produce many impressive designs.

At the other end of modeling spectrum are visual modeling applications. These programs lack most of the drafting related features that make CAD applications so appropriate for defining technical models. Instead they focus on artistic or entertainment markets, incorporating features that allow artists and designers to finely control the visual appearance of their models, including close attention to lighting, surface material, and texture mapping features. The models created with

these systems are widely used as visual prototypes for new products, in place of photographs or images in graphic design and layout applications, or as artistic accents or elements in larger compositions. Artistic modeling applications are also the tools of choice for animation and effects companies who create computer generated imagery for movies and television, not to mention the growing videogame market.

Although these applications lack the highly technical features of their CAD cousins, they instead provide features that facilitate the creation of organic and natural looking objects, something that can be difficult in CAD software. Like the CAD applications, artistic modeling programs work from a set of basic primitive shapes, but generally include much wider support for the creation and editing of parametric curves and surfaces, subdivision surfaces, mesh deformation, and other tools used to make rounded and organic forms.

An example of this sort of modeling application is Maya [Autodesk, 2007b]. Maya can be used to create any manner of object including highly detailed animated characters, vehicles and spaceships, convincing scenery and terrain, and the sort of brilliant and abstract unclassifiable displays of light and color that make theater goers ‘ohh’ and ‘ahh’. Maya is widely used in the entertainment industry to create 3-D models for movies, television, and video games.<sup>9</sup>

Designed as a professional grade application, Maya incorporates many of the same sort of industry level features of programs like AutoCAD such as programming and scripting support that allow it to integrate into a production pipeline. Also like AutoCAD, Maya comes with a hefty price tag that places it outside the financial grasp of most non-professionals. In this instance Maya’s designers are

---

<sup>9</sup>In fact, Maya’s designers were awarded an Academy Award for Scientific and Technical Achievements in 2003 [Newell, 2003] for their contributions to the cinematic arts.



more understanding, and provide low-cost and free version for the program to students for non-commercial work, however these version are also stripped of some features and place watermarks on content that some find bothersome. Luckily, the open source software community also provides several alternatives. Perhaps the most popular is Blender [Blender Foundation, 2007] an open source 3-D modeling application that sports many if not most of the features of commercial packages like Maya. Because of their wealth of features, both Maya and Blender have fairly complex interfaces and steep learning curves.

### **3.4.3 Limitations of Computer Based Modeling**

As we have seen the introduction of computer technology to the modeling process has brought with it many benefits. Computer modeling is at once, less expensive, more efficient, more expressive, and able to interact with other aspects of the creative process in ways that simply weren't possible before. However, computer modeling is not without its limitations.

For amateur and semiprofessional designers who may wish to use computer-based modeling in their own work, one of the largest barriers to entry are the price tags of professional computer modeling software. Compared to a large fully stocked modeling studio, a professional 3-D modeling package and computer on which to run it may represent a substantial savings for a design house or architectural firm. However, 3-D modeling software like AutoCAD or Maya come with a hefty four digit price tags, a substantial investment for a weekend hobbyist or a small private firm on a tight budget. This financial reality has prevented 3-D modeling software from proliferating the general consumer market in the same way that professional grade computer imaging software already has.

Seeing this issue, and sensing an underserved market, many software manufacturers are now offering simplified or scaled-down versions of their software targeted directly at consumers. These applications do not provide the full feature set of their larger cousins, but come at more reasonable prices for students or hobbyists who may not need the full power of a complete CAD or modeling system.

On the professional level, although 3-D modeling software has prevented the need for some traditional models to be constructed, in some instances, hand built models are still important. To an architect for example, a highly detailed computer model of the new building may contain a great deal more information, however when presented the plan to the public, images projected on a computer screen simply don't have the same impact as a physical scale model. Thus, computer-based modeling capabilities do not completely supersede those of traditional modeling techniques, but instead offer an additional tool in the creative process.

Another drawback to professional level 3-D modeling software is its staggering complexity. Most fully featured packages include not one but several different modeling methods that can be used in conjunction to create the final design. Designers and artists must understand primitive based point-and-click construction, parametric representations and control point editing, Boolean operations, subdivision surfaces, as well as a whole host of techniques to control the visual look of a model such as lighting, textures, and material properties. Learning this information is not necessarily difficult but it does take considerable time and training.

A major contributor to the steep learning curve of 3-D modeling software is the interface designs common to these applications. Although methods like point-and-click construction, and control point manipulation are well suited to the underlying representations of the 3-D model, from the perspective of the

artist or designer these interfaces can be cumbersome, tedious, and unintuitive. For an artist trained in traditional drawing or sculpting techniques it can be especially frustrating to spend hours constructing a model that would have taken only minutes to draw or sculpt. Software manufacturers are sympathetic to these concerns, and are beginning to address the deficiencies in their interfaces. However progress has been slow, and it's not entirely clear what constitutes an easy and efficient modeling interface.

These difficulties have spurred ongoing efforts by many researchers and software companies to design more intuitive means of working in 3-D. SketchUp is a perfect example of early foray into this new field of modeling interface design. A particularly active segment of this research now focuses on sketching as a possible metaphor for a better modeling interface. In the research community, an entire field of study has now grown up around the concept of sketch-based modeling, which attempt to allow artists and designers to draw their 3-dimensional models more directly.

## Chapter 4

# Bridging the Gap Between Sketching and Modeling

In the last section we caught a glimpse of the growing importance of 3-D modeling in the realms of design, entertainment, engineering, and art. No longer simply the tools of high budget research projects or a flashy extravagance for a big budget sci-fi movie, 3-D models and the techniques and applications that create them are now slowly trickling down into the mainstream. Thanks to the maturity of the computer graphics community, on a computational and theoretical level computer modeling is a well-developed science. However, as is so often the case, much more research has focused on how best to describe, render, and represent the 3-D model rather than on how users of the 3-D systems should best go about constructing them.

Because of their familiarity with the mathematical underpinning of the modeling process, and their involvement as a driving force in the development of many early modeling technologies, early interfaces have favored techniques that sacrifice intuition and experimentation for the high degrees of granular control required by

professionals in this market. However, as 3-D software has moved out of engineering workshops and into design studios or suburban homes, the interfaces have been slow to catch up. Despite such dolorous overtones, artists and designers are a resourceful bunch, and have made the most of what they've been given. None the less, demand in the marketplace for these issues to be addressed, coupled with the early work of some researchers, has led to a thriving research field in the development of new interfaces that bridge the gap between the technical benefits offered by the computer, and the traditional expertise wrapped up in centuries of real world artistic technique.

Modeling is not the only industry that has faced the challenge of integrating computers into a system steeped in traditional technique. Other industries that were largely or completely paper based have made similar transitions, replacing paper with pixels in some parts of the pipeline, or even making a full transition from one end to the other. Working with computers provides the artisans of these communities a degree of flexibility and even creativity that may have been difficult or impossible under the former regime. At the same time, the industries were able to retain the talents and skills associated with their traditional techniques even in the new virtual environment. Observing how this transition was made is instructive as a means of gauging how best to bring about similar change for the field of modeling. A particularly good example is provided by the animation industry.

## **4.1 Computerized Animation Systems**

Given the staggering number of cartoons that can be seen on television every week or which premier at theaters throughout the year its often easy to forget

how complicated a process animation is. Simply shooting a live-action television show or feature film can easily cost hundreds of thousands, or even millions of dollars, but beyond the logistics of managing a cast and crew, turning the actor's moves and dialogue into a reel of film or a video tape is almost an automatic process. The process is literally point and shoot. The camera threads the frames of film past the shutter many times a second and images are recorded by the light bouncing off of the actors and scenery and into the lens.

Making a cartoon on the other hand is an endlessly laborious process, requiring drawing and redrawing of every part of the film by professional artists. The production works as follows. Once a script is chosen, storyboarding is the first visual step of the animation process. One or more storyboard artists draw a series of simple pictures describing the camera angles and major actions in each scene of the cartoon [White, 1988], forming a sort of comic-book version of the entire film, shot for shot. These pictures are then posted up on boards so that the animators, directors, writers, and others can read through the script and visualize each scene. Storyboard drawings generally don't contain much detail and are drawn in a sketchy style. Once the sequences are fully planned out in storyboard form, the process moves along to the animators who hand-draw each individual frame of the film. Unlike storyboards that are largely for internal use, the quality of each hand-drawn frame is directly reflected in the quality of the finished product. The differences between a stilted and robotic cartoon mouse and a vibrant and dynamic one that seems to leap off the screen are encoded in tiny details of style and movement that animators must train for years to perfect. These hand-drawn frames are called key-frames and in-betweens. Key-frames depict the extremes of a character's motion and are produced first by the most

senior animators. In-betweens fill in the motion between the key-frames and are drawn by more junior animators or even by outsourced labor. The frames are drawn in pencil on sheets of paper and later transferred to transparent acetate cels, either by scanning them into the computer or by tracing or xeroxing each one individually. Once the frames are inked, each must be painstakingly hand painted to give the characters color. Finally, backgrounds, multiple frames and other elements are all combined and photographed to produce each frame of the rough film, which must still be edited and combined with sound, music, and voice talent to create a finished work.

At 24 frames per second, even a two-minute short can involve thousands of drawings and translate into months of work. To help manage the complexity of projects, most animation is done in an assembly line fashion where each segment of the film moves through a number of discrete stages [Fekete *et al.*, 1995]. Larger studios working on feature length projects can usually devote entire sections of their staff to each segment, but smaller independent studios may not have that luxury. In either case it's easy to see how difficult it might be to keep track of everything from one stage to the next. It's no wonder then that animation studios are integrating computers into their workflow. By using the computers to help alleviate some of the logistical grunt work, the hope is that the animators and other professionals who work on the cartoon will be better able to focus their efforts on the creative aspects of the program, and in turn, produce higher quality cartoons more efficiently. However, despite the widespread use of computer to handle the logistical aspects of animation, studios and artists were slow to move to computers for the animation process itself. Thus, until recently computers have played a largely supporting role in the animation process.

It is possible for animators to draw the individual frames directly into the computer, and some basic animation software includes these features. This method is often used, for example, for web-based Flash animations, which are more often built from geometric primitives than hand-drawn lines. Unfortunately for more traditional animation, capturing the feeling of hand drawings for computer processing has always been problematic [Henzen *et al.*, 2005], and so many studios still produce all of the initial artwork by hand and then go through a tedious, expensive, and error prone process of scanning, cleaning, cataloguing, and transferring each drawing into the computer in a form that can be manipulated.

One early exception to this trend was Walt Disney Feature Animation, the division of Disney Corporation responsible for their feature length animated films [Contributors, 2006g]. Disney has been using a proprietary computerized system called CAPS – Computer Animation Production System – since 1987 [Robertson, 1994]. Because CAPS was a proprietary system and only used in house, details about its features are not entirely known. However some general information is available [Contributors, 2006d]. CAPS was jointly developed by Disney and Pixar in the early 1980’s though it did not see widespread use until the 1990’s. The system was designed to replace the ink and paint process where the animator’s paper frames are traced onto cels and then painted. Instead, the animator’s pencil and paper drawings are scanned into the computer where virtual ink and paint is applied. CAPS also made it possible to integrate scanned background artwork into the system, taking the place of older rostrum camera technology that was used to photograph each cel along with its background.

As computers grew more powerful and more affordable, commercial off-the-shelf computerized systems like PEGS, Toonz, and Animo were introduced, and



have started to replace components of the animation pipeline for other studios [Fekete *et al.*, 1995]. Computer systems are now used at many different stages of the animation process, and have given rise to a boom industry in small, low cost, independent animation studios that would not have been possible just two or three decades ago. More recently, software has been developed that incorporates large swaths of the production pipeline from storyboarding all the way to final ink and paint. The first system to incorporate all of these stages, save the initial music and voice recording, was TicTacToon, the design of which is described in a 1995 publication by Fekete *et al.* [Fekete *et al.*, 1995].

TicTacToon includes many of the features of other animation workbench systems. By converting the artwork into digital formats, the animation process becomes more flexible. Characters are stored as resolution independent vector artwork and can be painted within the system from an infinite pallet of colors. Common motion sequences like walk cycles can also be stored and easily reused, and because they are represented as vector graphics, they can be scaled or transformed to fit into new situations with a minimum of work.

Like CAPS the program replaces the need for a camera system and allows backgrounds and characters to be combined and shot in a virtual camera system, however TicTacToon's capabilities move well beyond those of CAPS, especially in the early stages of production. The program's major feature was the introduction of a vector-based sketching system that allowed animators to draw storyboards, key-, and in-between frames directly into the computer in a way that mimics the traditional paper-and-pencil process. The animator draws into the system using a digitizing tablet, and his or her stroke is displayed as a simple graphical feedback plotting the raw points. Once the pen is lifted, the program analyzes

the raw points and converts them into a chain of quintic Bézier curves<sup>1</sup> that more accurately describe the path and vary in width with the pressure of the pen. Systems that convert user strokes into smooth line representations are nothing new, but TicTacToon incorporates a carefully tuned algorithm that makes the conversion process appear almost instantaneous, something that the program's designers describe as a necessity for the animators.

The developers of TicTacToon went to great pains to develop a system that would be accepted by animators, observing animators as they worked and developing the software through a series of prototypes using their feedback. Based on those observations the developers incorporated a number of features to make the artists feel at home. One traditional technique used by in-between animators is called *flipping*. In order to see the progression of frames the animator will stack several successive drawings together, leaving each one between their fingers, and fan them back and forth like a flip book. By flipping the frames quickly back and forth the animator can check the motion of the character to ensure that it's fluid and natural. TicTacToon includes a similar feature that allows the animator to create small sequences from their drawings. Not only does the program replicate this technique, but it makes flipping more powerful. Because they only have so many fingers between which to leaf pages animators working with the traditional system can only flip through a few frames at a time, but thanks to the computer the animator can flip through as many frames as he or she likes.

Fekete *et al.*'s original publication in 1995 noted that the software was already seeing use in the animation industry. Since that time, TicTacToon transitioned

---

<sup>1</sup>Bézier curves are an example of a parametric curve representation as discussed in Section 3.4.1.2 on page 114. More detail about Bézier curves is provided in section 5.1 on page 5.1 and in Section 6.4.6 on page 286.

to a commercial product, marketed by Toon Boom Animation, Inc., and since replaced with a product called Toon Boom Studio with a commensurate feature set [Toon Boom, 2006]. TicTacToon and its successors have proved very popular, and are now used by many studios. The authors of the original paper describing it's design mention that some of the strongest proponents of the system have been the animators themselves, in some cases convincing their studios to adopt it. These reports lend credence to the belief that the system provides not just a replacement for traditional animation, but a marked improvement for studios and artists alike.

The progression of the animation process from hand labor to computer systems has also opened up the opportunity to combine animation with other computer graphics techniques. Like Hollywood films, computer generated graphics first found their way into cartoons as special effects, often isolated to a single scene or used for a specific purpose. Many people will remember the ballroom scene from Walt Disney's 1991 animated feature *Beauty and the Beast* [Contributors, 2006a] in which Belle and the Beast share a romantic dance as the camera flies and spins around them showing an elegant ballroom and reflective chandelier. The ballroom, its features, and the sweeping camera moves were generated as a 3-dimensional model and then the digitized 2-dimensional characters were inserted into the scene. The use of 3-dimensional content in cartoons has since exploded thanks to the increasing computerization of 2-D animation and the wider availability of 3-D modeling software. Most animated films still rely on 2-D techniques for their characters and organic elements, but highly regular inanimate objects such as architecture or row upon row of books in a library are tedious and unrewarding for human animators to draw over and over in each frame. By defining them once

in the computer and then allowing the 3-D camera to generate frames automatically the animators can focus their work on the characters. However, because these computer-generated elements lack a degree of imperfection that comes from human work, they can sometimes appear static and stogy, lacking the vitality of the hand-drawn elements. Some studios are now addressing this issue by using non-photorealistic rendering techniques, which render computer graphic elements with a hand-drawn look.

Disney, always the innovator, is taking this one step further. According to an article by Robertson, Disney is now using a system called Deep Canvas that allows artists to build 3-D set and background elements, and then paint directly onto the models [Robertson, 1999]. Deep Canvas was specially developed for Disney's 1999 animated feature Tarzan [Contributors, 2006b]. The animators wanted a system that would allow the camera to move in and out of the scenery like a rollercoaster as the viewer follows Tarzan swinging from vine to vine. They considered 3-D modeling systems like the one used for Beauty and the Beast but were dissatisfied with the computer-generated look, and wanted the background to appear as lively as the characters themselves. According to the article, with Deep Canvas animators draw elements like leaves using Disney's in house 2-D animation software. Those drawings are then applied to the surface of a 3-D model as if they were coats of paint. When the process is finished, the 3-D geometry becomes transparent, leaving the 2-D leaves hanging in air but allowing further background elements to leak between them. The technique of rendering the strokes is based on Meier's 1996 paper on Painterly Rendering for Animation [Meier, 1996].

The creative potential of combining 2-D and 3-D techniques is obvious, and

the discipline of non-photorealistic rendering is pursuing many avenues of research in this area. One of the most exciting recent projects is called WYSIWYG NPR [Kalnins *et al.*, 2002]—what you see is what you get non-photorealistic rendering. This software allows a user to paint and draw directly onto 3-dimensional geometric models using a variety of colors and simulated media styles, similar to Disney’s Deep Canvas system. The user can also draw along the silhouette lines of the model, making its outline appear drawn or painted. Stylized hashing strokes can also be applied to the model, which rather than being attached to a point on the surface, instead react to the position of the viewer and light source to simulate artistic shading. The artist can even define the strokes at different levels of detail. Then as the painted object moves forward or backward within a view the details change dynamically to maintain a coherent look. The researchers responsible for the system have even developed methods of dynamically updating the user’s strokes so that painted elements maintain a realistic look through an animated sequence [Kalnins *et al.*, 2003]. See Note 1.

---

**Note 1** Non-Photorealistic Rendering

---

The field of non-photorealistic rendering is itself a fascinating area of research, and likely to become increasingly important as 3-D modeling and animation become more prevalent. Further discussion of the field is beyond the scope of this document, however for interested parties Tateosian *et al.* [Tateosian & Healey, 2004] provide a good general overview and introduction to the field. The following authors provide more detailed discussion and introduction of various techniques: [Gooch *et al.*, 1998] [Strothotte *et al.*, 1994] [Johnston, 2002; Kalnins *et al.*, 2003, 2002; Lake *et al.*, 2000; Raskar & Cohen, 1999; Saito & Takahashi, 1990; Sloan *et al.*, 2001; Zenka & Slavik, 2004] [Girshick *et al.*, 2000] [Buchanan & Sousay, 2000; Markosian & Lee, 2000; Markosian *et al.*, 1997]. Finally, [Götze *et al.*, 2005] [Wood *et al.*, 1997] [Petrovic *et al.*, 2000] offer examples of the use of NPR techniques.

---

Because of their flexibility, computers tend to enter a new domain in one or

two small areas and then rapidly expand, incorporating more and more of the job until tightly knit systems or unified workbench applications subsume most or all of the process. To many this rapid transition can be disconcerting or even frightening. Especially in domains that rely heavily on creative professionals, those artists and designers may worry that they will be forced to work with limiting computer systems that will sap their work of its vitality and leave their hard won artistic skills to languish. It is true that the transition period for these industries can be a rocky one as the abilities of computer systems struggle to reach parity with the abilities of traditional methods, but the animation industry provides a perfect example of how careful attention to the artistic, creative, and technical requirements of the *artists* leads to rewards on all sides.

Traditional animation techniques are not now, nor will they ever be completely replaced by the computer systems. However by alleviating grunt work, allowing artists to focus on what's important, providing powerful tools to augment and improve traditional techniques, and opening the door to new tools like 3-D animation, computer animation tools have made an effective bridge for time-honored animation skills into digital media.

## 4.2 Model Annotation

Just as the infiltration of computer technology into the animation field began at its edges, addressing the menial and mundane tasks of the profession and then slowly spreading into the core competencies, the same is true of the gradual introduction of more sketch-like features into conventional computer modeling applications. Although the bulk of traditional computer modeling interfaces may lack the intuitive interactive qualities of their real-world counterparts, we are be-

ginning to see the encroachment of minor sketch-like features into tangential areas of computer modeling.

Several research efforts are now focused on developing systems that allow their users to annotate 3-D models. This many seem like an odd application, but the researchers make a compelling case that a ready market exists for this functionality. As we have seen from our discussion of sketching, one common element of a sketch are annotations added by the artist. These may be small calculations as in the case of architectural or interior designers, instructional notes made on character sheets by animators, or editorial input generated by customers of an industrial designer critiquing a new design. Although annotation is not technically part of the rendering process of a sketch, the annotations function just like the drawn elements to communicate information that the artist or viewers feels is important.

A good example is provided by the architectural community. In the past few decades, the design and architecture world has transitioned from a workflow based around drawings and hand made models to a system of technological assets. Despite the benefits of such a system, because there are no paper plans to jot notes on, or physical model on which to affix a PostIt note, communicating and documenting the dozens or hundreds of suggestions and ideas generated in strategy sessions and review meetings with clients is difficult. This problem can be addressed in part by simply attaching text documents or small textual notes to image and data files, but some ideas, particularly those involving spatial concerns, are difficult to communicate when placed out of context into a secondary file.

This need has prompted the development of model annotation applications. One such system is called Space Pen [Jung *et al.*, 2002], and was developed by

Thomas Jung along with Ellen Yi-Luen Do and Mark Gross of the Electronic Cocktail Napkin project. Space Pen allows a group of users, either together or connected over the Internet, to collaboratively walk through a VRML model of a design or floor plan and add annotations by drawing directly onto surfaces, a process akin to spraying virtual graffiti. Users can also type longer notes and affix them to surfaces as small PostIt icons that others can access and read.

Whereas the SpacePen system limits the user to adding annotations to the surface of an existing model, several projects have been developed to allow the user to literally draw in mid air. The first was inspired by the 1955 classic children's book *Harold and the Purple Crayon* by Crockett Johnson [Johnson, 1981]—see Figure 1.1 on page 4. In the book a small boy, Harold, goes on an imaginary journey, taking with him his purple crayon. As he travels, Harold uses the crayon to draw all of the elements of his environment from a forest to a boat to a picnic. Taking this imaginative tale as a cue, researchers at Brown University developed Harold, a prototype program that allows users to create their own world by drawing it [Cohen *et al.*, 2000].

Users of Harold begin standing on an empty landscape. They can then create anything they like simply by selecting colors and tools from a small palette and drawing in mid air using the mouse or other input device. As the user draws the system automatically creates an invisible planar surface parallel to the viewing plane called a billboard, and projects the marks onto its surface. Each new element the user creates is constructed on its own billboard, and although the user's strokes are placed in 3-D positions, they only define 2-dimensional geometry. Because the artwork only defines one side of an object, as the user moves around the environment the boards turn to face them from any angle. Harold also allows



users to make simple adjustment to the surrounding landscape by drawing silhouettes along the horizon that are converted into 3-dimensional rolling hills. Users can even look up above them and draw across the sky. In this case the strokes are projected into the inside of a large sphere that spans the visible portions of the heavens.

Unlike Jung *et al.*'s SpacePen, Harold is primarily intended for a younger audience, and designed to facilitate creative exploration and storytelling for young children similar to coloring with crayons. To target to this audience the program's interface is simple and visually based, encouraging the child to try different tools and colors without worrying about textual menus or complex commands. The authors note that many of their design inspirations came from other children's software packages, specifically Kid Pix, a whimsical paint program that emphasized a simple interface, process more than product, and an open-ended design philosophy [Hickman, n.d.].

While Harold may seem little more than a toy, the project demonstrates that 2-dimensional elements have a place, even within a 3-D environment. Although the 2-dimensional strokes do not define 3-dimensional geometry from the perspective of the program, simply by arranging them in 3-D space the viewer, with a little imagination, can use them as abstractions to visualize 3-D structure. In 2001, Bourguignon *et al.* presented a system that applies this style of interaction to some real world applications [Bourguignon *et al.*, 2001].

Bourguignon *et al.*'s system, like SpacePen, is designed to work with an existing 3-D model, and like other annotation programs it allows the user to draw 2-D elements directly onto the surface of the model. However, like Harold, users can also draw in mid air, creating freestanding drawings. Mid air drawings are

projected onto a partially visible plane, which is generally parallel to the view. The depth of the plane can be adjusted by the user, or inferred from surrounding elements. Using the system users create two kinds of strokes: line strokes, 2-dimensional space curves defined in a 3-D position; and silhouette strokes, which actually define a limited 3-dimensional model. To generate models from silhouette strokes, the system converts the user's drawing input into transparent troughs or partial tube shaped 3-D surface onto which the visible line is projected. The trough is shaped by the curvature of the stroke and the visible portion is located at its apex. As the user's view changes, the visibility of the stroke attenuates, giving the illusion of a 3-dimensional surface contour. Because these strokes actually have 3-D surfaces associated with them the system can perform limited occlusion with other elements.

The authors provide examples of a number of possible applications for the program. As one would expect the system could be used to annotate existing geometric models loaded into the program. However, because strokes can now be placed anywhere, the user is free to make rudimentary indications of new elements that should be added or larger changes that could be made beyond whatever existing geometry may already exist. The authors present an example of creating a landscape design around a prefabricated model of a house. The additional abilities of the program also open up new opportunities. The authors suggest that it could be used as an entirely new medium to create 3-dimensional illustrations. For example the authors posit a concept they call 'guided design' in which a user can load a very simple prefabricated structural model such as a dress maker's mannequin, and then use 3-D strokes to create the actual model on top, literally drawing the cloths over the model's curves.

SpacePen and projects like it represent an effort on the part of interface designers to provide professional users with a means to transfer the methods and routines that were widely practiced before the introduction of computerized systems, but which did not seem to make the transition. By providing, for example, architects the ability to annotate models of their buildings along with their clients, they can apply traditional methods and techniques that have been time tested to the new technology. At the same time, we can see in projects like Harold and Bourguignon *et al.*'s guided design interface that these same ideas, once developed for one discipline, can turn out to have many applications. By incorporating new and old ideas together an entirely new medium with fresh creative possibilities can be formed.

### 4.3 Sketch-Based Modeling

These two examples, animation and annotation, provide valuable insights that can be applied to the design of a successful modeling interface. First, from the field of animation, we can see the vital importance of understanding not only the workflow of professionals in the field, but also the creative process that drives that production line. In order to foster that same process in the computer, a successful interface must focus on the areas of traditional technique where the artists and designers get their creativity and derive the character of their work. From our lengthy discussions of its importance to the creative process, sketching now seems the most logical area of focus. The same conclusions have been drawn by researchers in the fields of modeling and interface design and are the basis for sketch-based modeling research.

The traditional practice of sketching makes an attractive target as a basis for

new modeling systems for two reasons. First, as we have seen, both the physical activity and mental processes associated with sketching form the basis for problem solving, development, and general creative thought for most of the professions with a vested interest in 3-D modeling. From architects and engineers to artists and animators, sketching is an integral part of the design process and its practice a familiar tool at many stages. Second, and perhaps more revealing of the current climate of modeling applications, sketching is most utilized at early stages of the creative process when designs are vague and details scarce. This stage is also the most underserved by the current crop of commercial modeling applications. Because of the complexity of their interfaces and the technicality and granularity of the editing styles they offer, the process of early experimentation that is such an important part of sketching is often severely encumbered by the overwhelming inertia of the interface itself.

Despite the importance of sketch as a traditional technique, it's important to consider sketching's suitability to the modeling task. After all, we consider the 3-dimensional model as primary-geometry and its 2-dimensional projection as secondary. If current 3-D modeling applications provide an interface that allow us to work in the primary domain directly, then why continue to bother with the 2-dimensional domain at all? Can sketching skills even translate to this medium or are we trying to use the wrong tool for the wrong job?

To answer these questions, we must consider once again not only the physical activity of sketching, but the mental processes that drive it. Just because 2-dimensional sketches and drawings do not contain explicit depth information, this does not mean that 2-dimensional artists don't take the 3-dimensional positions of objects into account. On the contrary, artists pay very close attention to the

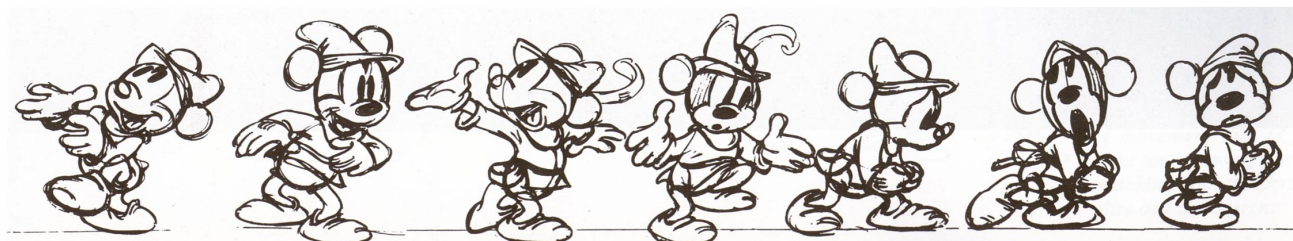
spatial relationships of their subjects, and use a variety of techniques to encode and represent that information in their work. We have already discussed how, for example, artists use perspective-drawing techniques to create views of buildings and architecture that give the illusion of depth.

Furthermore, sketching may offer advantages to working directly in 3-D. Despite the special reasoning abilities of the brain, working in three dimensions turns out to be more difficult for most people than working in only two. This should be a familiar sentiment to anyone who's tried to solve a Rubik's Cube. The puzzle is so challenging because it forces you to keep track of the positions of each colored square on all sides at the same time, and to consider the consequences a single move will have on the colors of each face. This is not to say that 3-dimensional reasoning is beyond the average person—and indeed many people can solve Rubik's puzzle—it simply means that working in two dimensions can be more intuitive, placing less demand on the artist's cognitive abilities and leaving that extra thinking power free to be used for creativity and problem solving.

Durand, in his paper discussing computer depiction, addresses this same issue [Durand, 2002]. He makes the point that it can be beneficial to describe a model in terms of secondary perception-based phenomena like occlusion and contour, rather than working directly in primary space because the artist is more adept at adjusting the model in its secondary representation. We humans are in fact so adept at the alchemy of creating 2-dimensional projections that can subvert those very techniques and use them creatively to express or emphasize ideas. We are also often more aware of the arrangement of 3-dimensional objects by their 2-dimensional projections. For example, although subway lines or street grids are laid out on a 3-dimensional landscape, it is much easier to fix your location or

plan a route by looking at their 2-dimensional projections on a map.

A subtler example is provided by Mickey Mouse’s ears. We all know the trademark shape of the famous mouse ears, so much so that three artfully arranged circles are enough to call the character to mind. However, you may not have noticed that even in his fully rendered animated form, Mickey’s ears are drawn as two circles, no matter the angle from which his head is viewed—see Figure 4.1. This so-called dynamic or view-dependent geometry is a common technique in animation [Rademacher, 1999], and underscores the fact that our mental image of the 3-dimensional configuration of familiar objects can sometimes be so strong that inconsistencies in their visual depiction slip our notice. More importantly, Mickey’s itinerant appendages demonstrate how the artist’s understanding of 3-dimensional space and the viewer’s perception of that space is a finely tuned skill.



**Figure 4.1.** Animation roughs of Mickey Mouse as he waves goodbye on his mission to capture the giant of the bean stalk in a short film *Walt Disney’s Fun and Fancy Free* (1947) [Johnston & Thomas, 1995]. Notice that although Mickey’s head turns from one side to the other, his hears remain as flat circles attached to his head.

A further advantage to working in the 2-dimensional domain stems from the realities of computer hardware. Although some advanced and experimental display devices and input technologies can allow humans to interact with computers in a virtual 3-D environment, this equipment is largely relegated to research labs and sports a hefty price tag. Although computers may be able to repre-

sent 3-dimensional objects, commodity hardware limits our interactions with that environment to the 2-dimensional computer display, 2-dimensional mouse, and keyboard. Thus in many ways, working with the 3-D modeling environment is not unlike a traditional sketching process in which 3-D subject matter are translated and rendered in 2-dimensions. The computer simply offers an opportunity to make that translation process a more dynamic one. Rather than selecting a single view of a 3-dimensional subject, with the computer the artist is free to change and update that view at any point during the drawing process. The fact that this aspect of drawing fits so well into the interpretation of computer modeling suggests further that traditional artistic skill can be translated to the digital medium.

This sentiment leads us to the lessons we can draw from our second example, how the introduction of annotation features to conventional modeling sparked novel uses of the technology. We can take from this example the fact that, although sketching will form the basis of our interaction method, our goal should not be to reproduce verbatim the traditional techniques. Discussing sketch and its relevance in modern design, Garner expressed similar sentiments. The objective is not to recreate sketching exactly as it exists in paper and pencil, or even to recreate the physical sketches as output, instead the goal is to adapt the computer so that the same mental process that makes sketching so useful as a creative tool can be used with the computer [Garner, 2001]. In this same vein, our application of sketch to 3-D modeling should serve as a guide to inform and inspire the design of the system. If a successful interface can be developed, the ultimate hope is that as computers take a larger and larger role in art and design, the division between the 2-dimensional and 3-dimensional worlds will become more arbitrary.

# Chapter 5

## Related Work in Sketch-Based Modeling

Not so long ago sketch-based modeling was almost unheard of, composed of only a handful of projects. However as the call for more intuitive modeling interfaces has grown louder and more urgent, research in this area has swiftly grown, and there is now a small but thriving community of researchers working on problems in the field. Because sketch-based modeling is mainly a shift in the interface of modeling applications, much of the research today focuses on different ways to support sketch and other traditional artistic idioms as an effective means of modeling input.

As the examples in the previous chapter demonstrate, the process of bridging the capabilities of modeling systems with the techniques of traditional art and design hinge on factors from two opposite ends. On the one hand, the way that users are expected to interact with the modeling systems must change to better accommodate how artists and designer work, and in turn how they think. At the same time, artists and designers will need to adapt to new techniques afforded by



the computer. Thus, the design philosophy for this new means of modeling must be one of ‘meet-in-the-middle’.

As straightforward a concept as sketch-based modeling and the philosophies behind it may sound, the implementation of such a system has turned out to be anything but. Although there seems to be resounding agreement that the current modeling interfaces leave much to be desired, there has been little consensus even on what sort of artistic idiom should take its place. In this section we will examine some of the interface mechanisms and modeling techniques that have fallen under the broad heading of ‘sketch-based modeling’. Within each we will review several representative approaches and techniques, both historical and modern, and attempt to ascertain which methods have been most successful.

## 5.1 Sketch Input

In order to allow artists and designers to interact with a sketch-based modeling program, the application needs to provide a comfortable and intuitive means to enter data. In the case of sketching, simple drawing strokes are the fundamental technique, and so reproducing the look and feel of basic drawing in the computer will go a long way towards allowing the traditional and innate skills of the user to transfer to this new medium. At first blush this would seem like a straightforward transfer. Computers are quite adept at representing lines in a variety of formats, and CAD and computer art packages have included line and curve capabilities from the very beginning. However there is a fundamental difference of opinion between the way computers think about lines, and the way we humans tend to.

To the computer, lines and curves are represented as a series of discrete points. The simplest lines are formed by directly connecting the points in sequence with

a number of straight line segments into something called a *polyline*. By stringing together a large number of points with very short line segments, the illusion of a visually smooth curve is created. Polylines are easy for computer to deal with and provide a visually effective representation of a curve. However as computers began to be used to design the smooth curves of aircraft wings and automobile bodies, designers needed a curve representation that also embodied the mathematical properties of the curve as well as its shape.

As we noted in Section 3.4.1.2, parametric curve representations were developed to fill this need. One of the most widely used parametric curve representations are Bézier curves, which were independently developed by two engineers working for French auto manufacturers Renault and Citroën, Pierre Bézier and Paul de Casteljau. Bézier curves offer an efficient and numerically stable means of representing polynomial curves of any degree as a set of control points.<sup>1</sup>

Because parametric curves were developed by engineers and designed for use in environments like industrial design where the properties of a curve are very important, most CAD software exposes the underlying representation of the curves by displaying the control points and allowing the user to adjust them directly. For an engineer this is ideal because it allows him or her to make changes to the curve in the most precise way. This means, for example, that the designer can adjust the curve to achieve or maintain properties like aerodynamic characteristics or continuity. Parametric curves have also found their way into most modern computer art system. Although artists and designers may not be as interested in their aerodynamic properties, parametric curves offer a compact and resolution independent way to represent complex curves.

---

<sup>1</sup>For a more thorough discussion of Bézier and other parametric curves and their properties see [Farin, 2002].

Unfortunately, when graphics software programmers borrowed parametric curves from the engineers, as we saw in Section 3.4.1.2, they also borrowed their control-point based editing system. Although editing curves by their control points allows a certain level of accuracy, the process is very unintuitive. Slight adjustments of control points can have drastic effects in some cases, and complicated methods of splitting curves and adding constraints are necessary to limit the effect of an edit to one portion of a line without affecting others. Many researchers claim that an understanding of the underlying mathematical representations of the spline are necessary for a designer to be able to use control point manipulation properly [Fleisch *et al.*, 2004; Michalik *et al.*, 2002]. This may be something of an exaggeration as many artists learn to use control point manipulation through practice and experience, but the issue remains that the method is far from intuitive and not conducive to the creative process.

Whether discussing polylines or parametric curves, to the computer the lines themselves are almost secondary, a consequence of the underlying position and characteristics of the points. Artists and designers on the other hand tend to think of a line in terms of one or more atomic strokes rather than a collection of points. In most cases the underlying representation of the data structures should not matter to the user. The user only cares that the computer be able to draw a line, and beyond that it doesn't really matter how. To provide the user with a stroke rather than point-based experience, drawing and modeling applications utilize a conversion and fitting process. Algorithms have been developed that can quickly and efficiently convert a series of discrete points produced by a hardware device during the artist's stroke from a polyline into a piecewise collection of parametric curves called a *spline*. This spline curve interpolates or at least approximates the

artist's intent.

For some applications, the simple creation of strokes in this more familiar manner goes a long way towards making users feel more comfortable with the interface. Once strokes are created, more traditional editing operations like those found in object-oriented drawing systems can be used to further adjust or edit the curves. In fact, for some professionals, the entire focus of the creation process is on creating strokes and little consideration is given to further editing once a stroke is laid down. Baudel for example points out that many animators have trained themselves to draw the 'correct' line the first time, and would rather scrap a drawing a start over than make any further corrections. This practice stem from a belief that these pure strokes preserve the character of the lines and give the animator's works vitality [Baudel, 1994]. Practices like this prompted Fekete *et al.*, the designers of the TicTacToon animation system, to forgo line editing features in their application all together [Fekete *et al.*, 1995].

Although some animators may be satisfied with their 'pure lines', as we know from our discussion of the sketching process in Section 2.3.1, most sketching is characterized by a continuous process of correction and revision where artists use techniques like overdrawing to adjust and enhance the path of previous strokes. Thus, line-editing features can sometimes be just as important as line creation. Unfortunately for the computer this is where the trouble arises. Although artists use overdrawing techniques to create a large collection of visible lines, the artist's actual intention behind that collection of strokes—the so-called 'line hypothesis' [Henzen *et al.*, 2005]—is only visually suggested by the totality of the existing lines, but is not necessarily coincident with any one of them. This causes difficulties for an application that is expected to deal with an ephemeral line that is not explicitly

represented by any of its data.

The need for a more intuitive system, both for the artists and the application designers, has prompted several research efforts. The classical work in this area was a project by Baudel [Baudel, 1994]. Baudel developed a so-called ‘mark-based interaction paradigm’ for editing an existing spline curve. Once a curve is defined, the user first selects the curve and enters an edit mode. The user then draws a new path over a portion of the curve, just as they would when overdrawing. The system matches the new curve segment to the old, finding the two locations along the original curve that are closest to start and end points of the new segment, and then replaces the intervening segment of the original curve with the user’s new input, blending the transitions between the new and old segments at both ends. Baudel’s system also featured the ability to delete segments of curves or join the ends of existing curves using similar methods. Once corrections are made, rather than accumulating strokes as in a traditional sketch, the system is continually adjusting a single curve to correspond to the user’s intentions.

Baudel’s original work proved inspirational to many later research efforts. Several sketch-based modeling system incorporated Baudel’s basic algorithm for use in 2-dimensional input. Michalik *et al.* use a very similar system as the 2-dimensional editing method for b-spline curves and surfaces [Michalik *et al.*, 2002]. Pereira *et al.* developed a prototype modeling application that converts 2-D input into 3-D models, and extends the sketching metaphor to allow users to make cuts in 3-D models by extruding sketch-edited 2-dimensional curves into curved surfaces that are then intersected with and subtracted from 3-D volumes [Pereira *et al.*, 2003]. Karpenko *et al.*’s SmoothSketch modeling system also used a mark-based editing interface based on Baudel’s to allow users to edit silhouette curves defining

inflated 3-D shapes [Karpenko *et al.*, 2002]. Furthermore John Alex in his thesis work points out that although the source code details are not available, Adobe appears to be using a system at least similar to Baudel's in action in current versions of it Illustrator vector graphics art application [Alex, 2005].

Although this sort of mark-based interaction paradigm may seem like an ideal editing method for drawing strokes, the process does have its drawbacks. The most difficult issue is in deciding how the system will interpret the user's inputs. Baudel's original system incorporated explicit drawing and editing modes that determined how the user's strokes are interpreted. Although such an explicit interface makes things a bit easier for the system, as we know from the nature of sketching, creation and correction operations are performed in quick succession or even simultaneously, and so jumping back and forth between modes slows the user's interactions with the system.

Other developers have attempted to alleviate this issue by dynamically interpreting each stroke as either an edit or creation, usually based on intersections or proximity to existing curves. However experience with these methods has proven that reliably determining the user's intentions in this way can be haphazard at best. Even when a corrective interpretation is correctly applied, it can be ambiguous exactly what portions of the curve the user wants to keep and what parts he or she would like to replace.

A further issue arises as the system attempts to smoothly combine strokes from a number of separate stroke events into a single smooth curve. Because the strokes were not generated by a single input, there is no single smooth path that is likely to interpolate or even closely approximate the combination of multiple segments. Where corners or other discontinuous features are desired this arrange-

ment suffices, but in most cases some degree of smoothing and blending will need to be applied. This requires a delicate balance between smoothness and accuracy. If blending is applied over larger portions of the curve, then as more and more editing and more and more blending is performed, the correspondence between the user's original inputs and the resulting path of the curve can begin to fade. If on the other hand blending is applied close to the joints between segments then the curve's smoothness will be disrupted. In either case it can be difficult for the user to predict what portions of the curve will be effected by any particular edit, which can quickly lead to frustration.

Several other researchers have also addressed curve creation and editing methods in an attempt to address limitations in Baudel's original designs. Fabian Di Fiore and Frank Van Reeth developed a system to mimic paper-and-pencil animation that took a slightly different tact [Fiore & Reeth, 2002]. Unlike most system that collect the user's stroke input as a set of discrete points and then apply a batch process to convert them into a spline representation, Di Fiore and Reeth used a system of on-the-fly curve fitting so that the creation of the curve is more immediate to the user. The two researchers also took a slightly different tack to a digital interpretation of overdrawing. Rather than using subsequent strokes to replace a segment of an existing curve, their system uses additional strokes as attractors that pull the path of an existing curve, but don't replace the curve segments all together.

It's unclear if using corrective strokes as attractors rather than literal corrections more closely mimics an artist's intentions when overdrawing, and indeed it may be more an issue of personal preference than a single solution. One response to this theory was proposed by the SketchAR system developed as part of the

European research project SmartSketch by Fleisch *et al.* [Fleisch *et al.*, 2004]. The system developed by Fleisch's team differs from Baudel's correction methods by first converting the original curve and the user's input into a set of discrete points at uniform intervals. The algorithm then continues as Baudel's by selecting the segments of the original curve closest to the user's correction. Rather than replacing the segment outright, the system uses a user adjustable parameter to weight the effect of the new curve, allowing the edit to act as either a strict replacement or a partial attractor depending on the user's preference. The authors also provided a second parameter that defines the smoothness with which the new curve segment will join with the old curve by defining the number of successive segments past each splice point effected by the edit. Once the alterations are made to the discrete points, the entire curve is converted back to a spline using a least squares fitting process.

Henzen and several other researchers later teamed up with Di Fiore and Reeth to developed a system intended for use by animators and cartoonists using a specially built electronic ink display [Henzen *et al.*, 2005]. Henzen's system also opted for an on-the-fly curve fitting system, and borrowed many of the interface characteristics of Di Fiore's original design, which fit well into the exotic display system. Along with stroke-based editing Henzen *et al.*'s system also allows users to make changes to lines, here represented as chains of Bézier curves, through affine transformations, although the underlying control point structure is still hidden. Henzen *et al.*'s system considers issues of line stability resulting from such edits, and dynamically adjusts the number of discrete curve segments along a curve to maintain smoothness through transformation.

As for stroke based editing Henzen *et al.* actually developed two line systems,



which the authors dubbed ‘nervous hand’ and ‘smooth hand’. For the smooth hand system they used a method much like Baudel’s where segments of the original line are simply replaced. Nervous hand takes quite a different tact. Each stroke entered by the user is painted to the screen and slowly begins to fade over time. As strokes are built up along an existing line, the line is drawn towards the most saturated, acting like an attractor system. This system is unique in that it continues to display the user’s editing strokes, providing a visual similar to the physical version of overdrawing.

John Alex in his thesis work also used this idea of displaying multiple lines to provide a history of editing activity [Alex, 2005]. However in Alex’s system these extraneous editing strokes are stored on a separate layer from the actual geometry and are not utilized as part of the modeling process. Alex’s line-editing system, dubbed ‘Pentimenti’, incorporated several novel features. The first, and most visually apparent, is a control widget in the form of a small adjustment handle that extends at a right angle from the path of a selected curve as the user’s cursor comes close. By dragging the widget to or from the curve the user can adjust an editing radius that surrounds the curve. Strokes applied within this radius are interpreted as editing commands whereas those originating outside are taken to be the definition of a new curve. This allows the system to avoid the need for an explicit specification of creation or edit mode by the user.

Alex’s other major contribution is the ability to create and edit lines containing vertices with valence higher than 2, meaning that the path of the curve can split or intersect forming forks or t-junctions. A user can draw a new curve ending close to an existing curve to create a join, or can create a fork by drawing an editing stroke starting near the path of an existing curve and then veering off at

a sharp angle. User can also join forked or jointed paths back together with a sewing operation.

Examining these stroke capture and editing methods overall we can draw a few general conclusions. First, although some of the direct editing systems show promise, in general the ambiguous and inexact nature of sketching does not lend itself well to interpretation by the system. Thus, those systems that rely on the user to make explicit corrections provide a more intuitive and less frustrating interface. By the same token, it's important to remember that the corrective and overdrawing strokes an artist creates have more value than simply as piecewise replacements for past mistakes. Overdrawing strokes provide the user with a history of the sketching process, and are the raw materials that fuel the mental processes of feedback and incremental refinement. By the same token, not all of the artist's lines are drawn to indicate their subject, but are sometimes created as foundations or scaffolding onto which further strokes will be laid. Thus it's important that the system provide the user with tools that cover the entire range of sketching strokes, not just those that make up the final drawing.

## 5.2 Gesture Created Primitives

Probably the first serious example of a sketch-based 3-D modeling system was developed Zeleznik, Herndon, and Hughes. In 1996 these three researchers presented a paper describing their proof-of-concept modeling program called SKETCH. The trio's publication is quick to mention that many of the modeling techniques and interface designs used in SKETCH were based on the work of other researchers in the field, however SKETCH is generally regarded as the first example of a complete system explicitly targeted at the preliminary 'doodling stage' of 3-D design.

Consequently, nearly all of the other sketch-based modeling projects presented here borrow in some fashion from Zeleznik *et al.*'s early work. In particular the application centers on two specific interface features, gestures and non-photorealistic rendering, making a strong case for why these elements support the familiar concepts of traditional sketching.

The research team emphasized in their publication that SKETCH was designed with the intent of combining features of paper-and-pencil sketching, which they describe as rapid and approximate, and traditional CAD software, which offers the user greater flexibility in editing and viewing a design. The goal of bridging these two camps had a heavy influence on the interface design of the system. This is most evident in the user interface, which is completely controlled by a system of gestures. Almost all input by the user is supplied to the system through a standard three-button mouse with only minimal additional signals provided by modifier keys on a keyboard. Modeling input is created with the primary button of the mouse, interaction gestures with the middle button, and camera adjustment and viewing operations with the third button. This design was selected explicitly to avoid the need for complex menu systems and tool palettes so common to other drawing and modeling applications.

To issue creation or editing commands the user draws one of a number of gestures using the mouse, which are projected into an invisible film plane parallel to the user's viewpoint and situated between the user and any scene geometry. Despite this heavy reliance on gestures, strokes recognized by the system as gestures are surprisingly simple, consisting of intuitive point and click operations like click and release, click and drag, click and pause, etc. Gestures are interpreted based on their context within the scene, the mouse buttons used, and the pres-

ence or absence of modifier keys, meaning that all recognition can be preformed by a simple finite-state machine model. Furthermore, once a gesture has been performed, its stroke remains visible on the film plane until another gesture is begun. This allows the user to select their previous gesture, reposition it, and reissue the corresponding command without having to reform the gesture.

Users can create new geometry in the form of several primitives such as boxes, spheres, cones, and a number of other basic shapes, as well more complex extruded regions or surfaces of revolution. New geometry is created through sequences of strokes that combine into a gesture defining both the shape to be created, and details of its form such as orientation and position. The researchers made an effort to select stroke sequences for each primitive that correspond to the visual characteristics, usually the configuration of edges, to help make the process more like sketching the shape on paper. The system also uses the direction of the user's drawing strokes to combine the new geometry with existing forms using CSG style operators. Gestures are also used to edit geometry once it has been created, and again an effort was made to select gestures that evoke similar operations in paper and pencil. The authors offer the example of resizing an object, which is often indicated on paper by overdrawing the edge of the form to extend it. To mimic this intuitive action SKETCH recognizes the creation of two nearly coincident gesture strokes over the edge of an existing object as an indication that the object should be scaled. Similar visual idioms can be used to position, transform, and remove objects.

Anecdotal reports of user interactions with the system reported by the authors seem to indicate that the gesture based system was largely a successful interaction mechanism. However, Zeleznik *et al.* point out several difficulties that cropped

up during the development process. They report that the particular set of gestures used by the system were constructed in an ad hoc fashion, selected because they seemed to be the most intuitive or appropriate at the time. While this was successful in selecting gestures with a strong visual connection to the operations they represent, some gestures proved to be difficult to use on a regular basis. Specifically they found that gestures involving a pause quickly became annoying as the user must disturb their fluid interaction to make the pause. They also highlight an early ‘toss out’ gesture that was originally used for deletion because of the strong visual and physical relationship to the operation, but which had to be scrapped because of a lack of precision. These findings are in agreement with many of the properties of gesture systems we discussed in Section 2.4.2.1.

In addition, although the authors’ publication does not report problems drawing gestures using the mouse, other researchers have cited this point as an area in need of improvement [Pereira *et al.*, 2003]. Zeleznik *et al.*’s paper does conclude suggesting that a digitizing tablet may be a more promising interface device. Later effort by members of the original research team extended SKETCH’s interface to use a digitizing tablet or display [Zeleznik, 1998], or a pen in combination with higher degree of freedom input devices and virtual reality viewing systems [Forsberg *et al.*, 1997].

Along with the need to replicate the physical activates of sketching in their interface, Zeleznik *et al.* also recognized the important role of the visual characteristics of sketches. As we know, the sketchy look and feel of sketches serves a dual purpose. First to emphasizing the ambiguity of a scene to the designer to help them look past the simple geometry that is depicted and see the idealized mental image they are working towards. Second, the sketchy look conveys to third

parties the approximate and preliminary nature of the model. In order to preserve this property in SKETCH, all geometry can be rendered using one of several non-photorealistic rendering techniques. The paper describes a few methods including a sketchy silhouette line algorithm that uses white filled models and jittery dark edges, a charcoal rendering effect, a watercolor effect, and others.

Although SKETCH was a very early prototype, its designers envisioned a variety of applications that could benefit from similar interfaces including rapid prototyping, CAD design, storyboarding, and animation. SKETCH's revolutionary interface was more than enough to spur many researchers on to try sketching as an interface metaphor, and to adapt other computer graphics algorithms and techniques to translate traditional sketching techniques to the world of 3-D modeling.

Zelevnik *et al.*'s early work proved the efficacy of gesture interfaces as a means of creating 3-D models, and showed that by choosing gestures that mimic traditional or logical sketching metaphors for 3-D primitives a modeling application can tap in to a designer's freehand drawing skills. Programs like SKETCH inspired a number of research teams to try gesture interface or to integrate gestures into their existing code bases. A good example of such an effort is the research team of Pereira *et al.* who integrated a gesture interface into a vector drawing system called IDeS [Pereira *et al.*, 2000]. Following the team's progress through a number of their publications we can see that as their gesture system became more mature the need for other construction methods was all but eliminated.

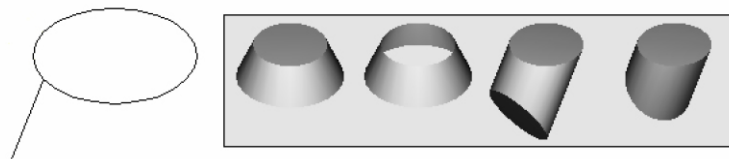
The team's first paper [Pereira *et al.*, 2000] describes their initial prototype called the Gesture-based Intuitive Design System or GIDeS. The application was built on the foundations of IDeS, a traditional menu driven design application

previously developed by the same researchers. While the underlying kernel of the system remained, the interface of GIDeS was completely redesigned to take gesture, drawing, and editing commands from a digitizing pen. To distinguish the drastic shift in interface design, the authors refer to their new interface style as a ‘calligraphic interface’.

The original GIDeS system incorporated two different methods for creating model geometry. The first and more traditional system, borrowed directly from IDEs, took the user’s freehand drawings of objects and tried to reconstruct them. The user can assist the system within a 4-viewpoint axonometric display by drawing the figure from different angles and connecting their shared vertices. The second system resembled that of Zelenik *et al.*’s SKETCH, allowing the user to create one of ten primitive 3-dimensional shapes such as cubes, spheres, cones, extrusions, and surfaces of revolution, using a canned gesture.

GIDeS made several important improvements over the SKETCH system, including allowing gestures to be created in any order or with strokes in any direction, but the true novelty of Pereira *et al.*’s calligraphic interface was its ambiguity resolution system. Rather than trying to differentiate between strokes meant for either editing system, GIDeS simply interpreted all strokes as possible gestures. When a combination of strokes is entered that matches a possible command a small contextual menu of icons called an *expectation list* appears to inform the user that the gesture can be converted. At this point the user can select the icon to instantiate the primitive, or the user can simply ignore the suggestion and continue drawing. An example expectation list from the researchers’ publication is pictured in Figure 5.1. The same system also uses expectation lists to offer the user several interpretation options when the meaning of a gesture is ambiguous.

The use of a contextual disambiguation mechanism is not new,<sup>2</sup> however where program's like Quick-Sketch used these mechanisms for feedback after a recognition event GIDeS actually involves the user in the recognition process and no recognition act is completed until the user signals their acceptance by clicking an icon in the expectation list.



**Figure 5.1.** An example of an expectation list (right) generated by a user's input gesture (left) in Pereira *et al.*'s GIDeS system [Pereira *et al.*, 2001]. Notice how a single gesture could conceivably be interpreted as a number of different 3-D constructions.

In early user testing of the GIDeS system Pereira *et al.* reported positive feedback from users. After this initial success the research team continued to refine their prototype, focusing primarily on the gesture system. The team's next publication [Pereira *et al.*, 2001] highlights a refined interface philosophy they now term the *Reduced Instruction Set Calligraphic Interface* or RISCO—playing off the well know acronym for the RISC microprocessor instruction set architecture. As we have discussed, a primary concern with gesture-based system is the makeup of the gesture set. The system designer must strike a careful balance between gestures that are intuitive and easy for the user to learn, and gestures that are distinct enough for the recognition system to differentiate. After working with the calligraphic interface system Pereira *et al.* realized that by allowing the user to help differentiate the meanings of ambiguous gestures the interface can

---

<sup>2</sup>Although the authors' publication does not cite previous work in this area we have already discussed Quick-Sketch by Eggli *et al.* [Eggli *et al.*, 1995], which uses a very similar mechanism.



accommodate more similar gestures that better approximate traditional drawing.

Within this new RISCO system, all modeling, editing, and viewing operations are now handled by the gesture system. The user's input is processed by three separate recognition modules; one for commands; one for 2-dimensional drawing primitives such as circles, curves and lines; and one for 3-dimensional primitives. As before the gestures to access this functionality can be drawn in any order, and are not dependent on the dynamics or direction of the strokes. However the system does take the direction of strokes into account when new geometry is created on top of an existing model. By indicating a directional stroke either into or out of the existing geometry the new primitive can be combined with the old model as either a CSG union or subtraction operation.

In this new version Pereira *et al.* also introduced a set of higher level editing operations designed as more powerful and intuitive replacements for the standard modeling transformations familiar to most CAD systems. Rather than clumsily trying to position objects by hand—a difficult feat when working with a 2-dimensional display—GIDeS uses a gluing operation in which the user draws a line between an object such as a vase, and the position on another model where he or she would like to place it, such as on a desk. The system automatically reorients the first object and places it to rest on the second. Internally the object is also attached to the modeling hierarchy of its base, and so moving the base object also moves any objects glued to it. The user can also adjust the position of a glued object in relation to its base with an adjustment mode.

From the first version of GIDeS to the second the gesture recognition mechanism matured a great deal from a mere feature of the interface to the primary interaction mechanism of the program. Having proven that the RISCO interface

philosophy was a viable one, in their next publication [Pereira *et al.*, 2003] Pereira *et al.* focused on developing GIDeS into something that could bridge the gap between preliminary sketches and precise engineering models. To achieve this functionality the team integrated a system of context-based anchoring constraints to allow the user to define constraints like absolute position and alignment of components of a model. However, the research team is quick to point out that the constraint system was not added at the cost of the freehand drawing style of the interface. As they state in their publication, "...it is possible to specify rigorous alignment and positioning without resorting to coordinates, through the judicious use of constraints and visual commands."

The publication's description of the GIDeS system continues the RISCO design philosophy, completely scrapping the IDeS recognition system for purely gesture base creation and editing focusing on a 'constructive and incremental approach' where in designs are built up from the combination of 3-D primitives. The system also incorporates an oversketching feature based on Baudel's curve editing system [Baudel, 1994] that allows the user to edit their strokes before recognition. The paper also describes the researchers' experimentations with adaptive expectation lists, which remember a user's previous preference for a specific interpretation among choices, and reorders the list on subsequent incarnations placing historical choices first. According to their publication the authors found that users interacted less with the adaptive lists, suggesting that users tend to be consistent in their input.

Work on the GIDeS system continues, and Pereira *et al.*'s most recent publication [Pereira *et al.*, 2004] describes a new version dubbed GIDeS++, which features an integrated system of recognition subsystems the authors call a 'cascading

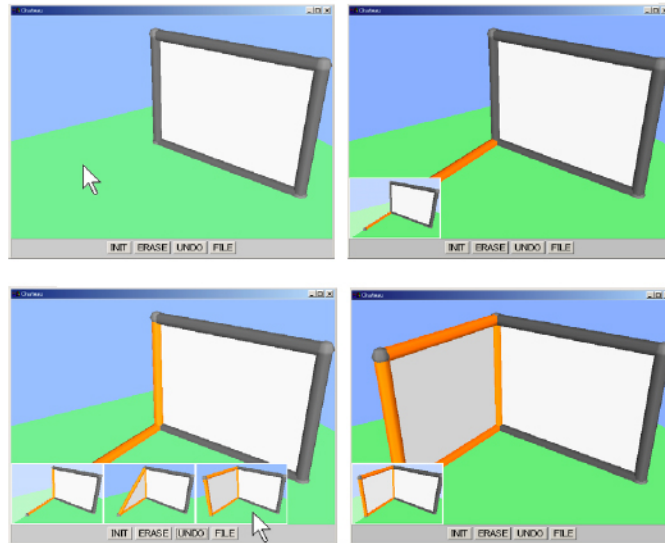
recognizer'. GIDeS++ now features four separate recognition subsystems; one for commands; one for 2-dimensional primitives; one for 3-dimensional primitives; and a final Boolean recognizer, which is responsible for recognizing or inferring Boolean CSG operations between instantiated 3-dimensional primitives. Rather than supplying each module with the user's input, raw strokes are first delivered to the command subsystem for recognition. If no commands are recognized or the user dismissed all of the expectation list suggestion then the strokes are passed on to the 2-dimensional recognition subsystem. Here strokes are recognized as primitive shapes, lines or curves. When a new 2-dimensional primitive is created it is automatically entered into a drawing graph and the 3-D recognition subsystem is triggered, which analyses the shapes within the graph based on connections and topology, looking for 3-D primitive gestures. Finally, if a 3-D primitive is recognized and will be instantiated over an existing object, the Boolean subsystem is triggered to determine what if any CSG operations are implied by the gesture. In this way the user's input cascades down the recognition chain until a suitable interpretation is found.

The progression of the GIDeS system from a program designed for reconstruction of 3-D models from freehand drawing input such as that used by its predecessor IDeS to one focused completely on gestures and 3-D primitives provides an important example for sketch-based modeling design. We may be tempted to say that GIDeS doesn't really carry the concept of sketching into 3-D modeling because the program deals in commands and primitives rather than freehand strokes, and it's probably sentiments like these that drove Pereira *et al.* to include a freehand drawing system in their original designs. However experience with the gesture system, and the concept of RISCO that allows gestures to be much more

intuitive and ambiguous, prove that gestures are an effective and intuitive way to bridge the gap between sketching and modeling. RISCI gestures represent a comfortable middle ground between the ambiguity of traditional sketching with all of its creative benefits, and the requirement of the computer for explicit instructions.

The system of expectation lists used in GIDeS offers the user a way to sort-of O.K. the system's interpretation of their input. This means that the process of communicating with GIDeS is based on a call and response mechanism of presenting a gesture and then approving a recognition, a process that divides the usually fluid creative process into discrete steps. In order to preserve these fluid interactions researchers Igarashi and Hughes developed a 3-D modeling system [Igarashi & Hughes, 2001], which takes the gesture recognition process from a structured discussion to an ongoing conversation.

Igarashi and Hughes called their interaction system a 'suggestive interface'; so named because the system continually observes the user's input and offers suggestions as to what they might be trying to do. Their paper describes a proof-of-concept 3-D modeling system called Chateau designed to allow quick and fluid modeling of architectural structures (such as French Chateau's) using the suggestive interface. Models created with the system are limited, consisting only of straight edges, planar walls, and other architecturally common features. Rather than striving for a polished and clean look, the program intentionally emphasizes the gross nature of the models by displaying their components as whimsically colored tube like cylinders joined with balls and surrounding colored walls—see Figure 5.2. Although the system does incorporate a few literal gestures such as a scratch out command for deleting, dragging out straight lines and selecting existing lines perform nearly all of the modeling operations.



**Figure 5.2.** A construction using Igarashi and Hughes’ Chateau modeling system [Igarashi & Hughes, 2001]. The small images in the lower corner of each display represent suggestions by the system of editing operations based on the user’s current input.

Models are constructed in an incremental fashion out of straight lines, which are created simply by dragging out lines that are projected onto whatever geometry is behind them. Thus the process begins with the ground plane and moves up onto walls and other surfaces as they are constructed. As the user draws the lines he or she creates are automatically highlighted, focusing the system’s attention on those components. After each operation a suggestion engine analyzes any selected lines and offers the user a set of possible construction operations. For example, two attached perpendicular lines might prompt the system to create a planar rectangle or triangle bounded by the two lines, or a transparent construction plane onto which the user could add further lines. The suggestions are presented to the user on the lower portion of the screen as a series of smaller, thumbnail-sided complete renderings. The user can mouse over any of the suggestions to view it in full, click a suggestion to perform the operation, or simply ignore the suggestions

and continue to draw.

In some ways the suggestion system is very similar to GIDeS's expectation lists in that the suggestions are presented for the user to choose from. However operations in GIDeS are activated by isolated gestures, whereas Chateau's suggestion system is continually analyzing the set of selected components created by the user and offering new operations. This means that the user can shift the systems focus at any time simply by making different selection, allowing them to come at a single operation from many different paths and incorporate components created at different times. Igarashi and Hughes refer to this kind of interaction a 'mediated gesture system' because the process of recognition is more like an ongoing conversation where single inputs can trigger an operation or be used as the pretext to a more complex command.

We previously discussed a drawing assistant program designed by Igarashi and a group of other researchers called Pegasus [Igarashi *et al.*, 1998], which utilized a system of interactive beautification to predict constraints like perpendicularity and symmetry and offers the user possible additions to their drawing. A similar predictive system was also integrated into Chateau, this time focusing only on selected components of the model for its predictions. The other unique feature of Chateau is a 'drafting assistant' functionality, which allows the user to create snapping and joining constraints as they draw lines by simply passing over or touching a line during a dragging operation. In some ways this interaction is also a gesture mechanism allowing a user to define both new geometry and its constraints in a single operation.

As a full modeling system Chateau is quite limited, but it serves as an enlightening example of Igarashi and Hughes' suggestive interface, and provides a quick

and efficient means of generating simple models. By breaking the rigid gesture recognition process out of discrete operations and into an ongoing conversation Chateau moves one step closer to the fluid creative process that sketch-based modeling is striving for.

From a wider perspective, we can see that systems like SKETCH, GIDeS, and Chateau, have approached the issue of developing 3-D geometry from 2-dimensional input by interpreting that input as a limited set of commands to the system. Using even simple gestures like those in SKETCH, the researchers found this to be a reasonable means of interacting with the system. However, its interesting to note, as evidenced by the evolution of a project like GIDeS, how the complexity of the gestures have evolved, incorporating not only simple commands, but modeling input as well until it becomes difficult to distinguish between something constructed from a gesture and something constructed from a drawing. Igarashi and Hughes' Chateau system blur this line even further by interpreting the user's input and interactions as a whole in a continuous conversational process. This suggests that perhaps direct interpretation of the user's drawings may be a viable avenue of inquiry.

### **5.3 Artificial Intelligence and Machine Learning**

In the late 1960's and early 1970's some of the first artificial intelligence and computer vision researchers began to ask if a computer could be taught to comprehend the spatial arrangement and 3-dimensional shape of objects in a picture the same way human can [Guzmán, 1968]. Medical science tells us that the ability to see depth in our environment, the skill that lets you catch a ball in flight or navigate through a crowded room, is the direct result of your physiology. Each of your

eyes provides your brain with a separate field of view, but because your eyes are positioned on the front of your head, those two fields of view overlap. Your brain makes use of this redundant information to triangulate the depth of objects you see, comparing the position of an object seen from one eye with the slightly different perspective seen in the other. This same physiological characteristic, called *stereoscopic vision*, is shared by many other animal species, especially predatory species to whom the ability to catch a moving target means the difference between eating and going to bed hungry.

Because stereo-vision is based on simple physical principles and geometric properties it was a logical target for early computer vision researchers. Today computerized stereo-vision is a solved problem, and many modern robots are equipped with stereoscopic cameras or other range finding sensors that work almost identically to human vision to give the robots a sense of depth. Stereo-vision is extremely effective in physical environments, but there is one area where the comparison between robot vision and people vision quickly breaks down. If you were to show your human friend a photo of objects in a room they could quickly and effortlessly tell you the position of each item relative to any other, but show that same photo to a robot and the most it could likely tell you is the distance from its cameras to your hand holding the photo.

Photographs and line drawings of 3-D objects are projections of the 3-dimensional geometry onto a 2-dimensional plane. We can think of the projection as taking all of the items in the scene at any depth and somehow squishing them down until they are completely flat. Thus from one perspective all of the items appear to have their original positions, but each no longer has any depth, and so there is no direct way to measure the distance between any of them. However when we

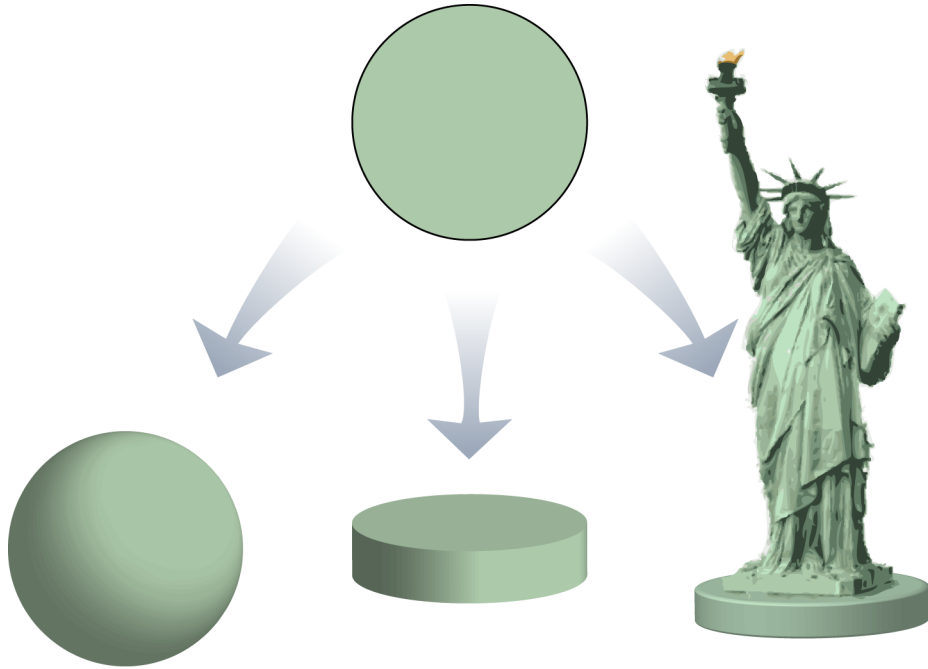


look at the picture our brains can reconstruct the 3-dimensional shape of each object and describe the relative distances between them. This translation and reconstruction process is so fluid and natural that our brains make it look like child's play, but although it seems to come naturally to us, it turns out that the underlying task is surprisingly complex.

After all, without depth information any given 2-D line drawing could conceivably correspond to an infinite number of possible 3-D shapes. What's more, it's impossible to view every angle of a 3-D object from a single drawing, and so without views of the occluded portions of geometry it's impossible to tell, for example, if a circular shape is a ball, a pancake, or the pedestal of a statuette of The Statue of Liberty seen from below—see Figure 5.3. From this perspective seeing anything in a drawing begins to sound like an impossible task, but of course we all perform this miracle hundreds of times a day without giving it a second thought.

In reality our brains are unable to construct a complete description of the object, but by accepting some level of ambiguity, and drawing conclusions from clues in drawing such as the direction of lines, where they appear to begin or end, shadows and shading, etc., we can see past the literal 2-dimensional projection and construct a plausible model of the 3-D scene in our heads from this overall gestalt [Lehar, 2004].

Exactly how the human brains learns this skill is still a matter of debate. It's possible that the brain comes pre-wired to recognize certain geometric configurations as their 3-dimensional counterparts, however its more likely that this skill is an intuition, learned over a person's lifetime of comparing the world around them to the depictions of that world they encounter. Lipson and Shpitalni call this an



**Figure 5.3.** Three dimensional objects can contain a wealth of detail, or be composed of very simple shapes. However, when displayed in a 2-dimensional projection, the viewpoint of the viewer can be as important as the form of the model in relaying the information in the image. In the example above, we can see that three very different 3-dimensional objects when viewed under the correct conditions can all appear to look like a simple circle.

*acquired geometric correlation* [Lipson & Shpitalni, 2002]. Furthermore, Hoiem *et al.* point out that although a single photograph may theoretically present an infinite number of interpretations, using common knowledge about how the world works we can usually discard the vast majority of those possibilities as patently silly.

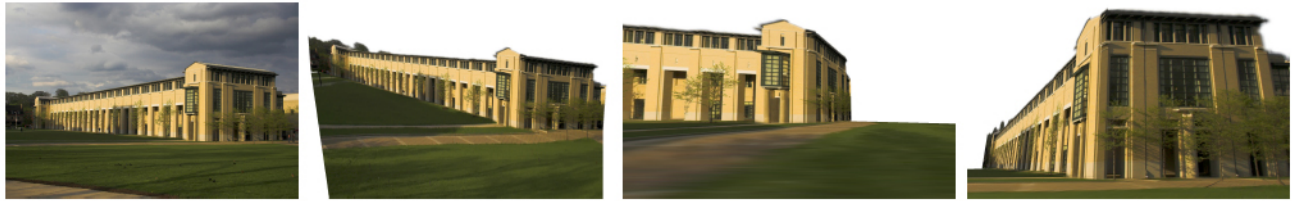
Pieces of solid matter do not usually hang in mid-air but are part of surfaces that are usually smoothly varying. There is a well-defined notion of orientation (provided by gravity). Many structures exhibit high degree of similarity (e.g. texture), and objects of the same class tend to have many similar characteristics (e.g. grass is usually green

and can most often be found on the ground).—[Hoiem *et al.*, 2005]

We humans can easily apply this kind of common knowledge about the status quo of physical reality, but explaining these principles to a computer is another matter. However as children our parents don't sit us down and explain the intuitive principles of gravity and balance, and so it seems logical that a better option for computers might be to undergo a similar learning process in order to learn what makes sense and what doesn't.

This was exactly the process used by Hoiem *et al.* to reconstruct the 3-dimensional elements of photographs and automatically build 3-D walkthroughs from a single view, a task many in the computer vision community had written off as either unsolvable or at least computationally infeasible [Spice & Watzman, 2006]. Hoiem *et al.* collected a training set of 82 images of outdoor scenes from a popular internet image search and used their software to divide the pictures into regions, labeling each. This training set was then used to construct likelihood functions that assigned scores to similar regions in candidate photographs, classifying them as sky, ground, vertical elements like buildings, etc. Using decision trees their software was then able to automatically find vertical elements in a photo and then prop them up against the ground like cutouts in a popup book. By wrapping the original image around a plane with these cutouts the image can be viewed from alternate angles. One of the authors' examples is provided in Figure 5.4.

In their publications the researchers note that their system has several major limitations. Only outdoor scenes can be processed and the program has difficulty dealing with more complex images like crowds, slats, stairs, or images with multiple



**Figure 5.4.** A traditional 2-D photograph (extreme left) is processed by Hoiem *et al.*'s automatic photo popup system [Hoiem *et al.*, 2005] to generate a simple 3-D model allowing alternative views of the scene (right).

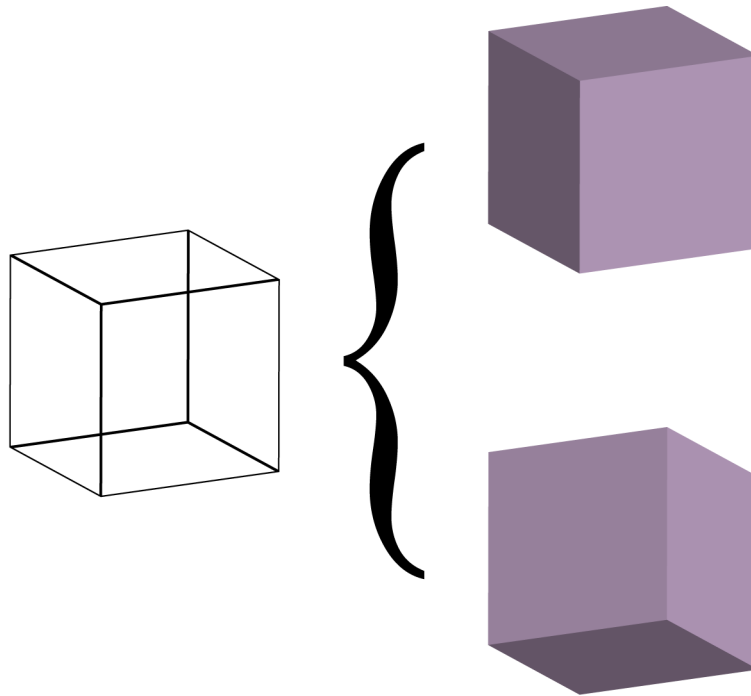
ground planes. Even with these limitations they report that only about 30% of images produce an accurate model. Given that this task was previously deemed impossible these results prompted Takeo Kanade, an early pioneer in computer vision research, to call their work a breakthrough [Spice & Watzman, 2006].

Geometric correlation has also been applied to the field of sketch-based modeling. In their 2002 publication Lipson and Shpitalni describe a modeling system that can reconstruct a user's 2-dimensional wire-frame drawing into a 3-dimensional model using a similar method [Lipson & Shpitalni, 2002]. Although the researchers' paper does describe some applications of their techniques, it does not describe a particular interface or specific modeling program. Instead the paper focuses on the reconstruction system itself. The technique they describe is based on a system of correlation tables that act as a discrete probability function comparing the 2-dimensional angles of the wire-frame drawing with candidate 3-dimensional reconstructions. The tables themselves were constructed from a training set of 100,000 randomly generated and flattened 3-D scenes. The authors note that other correlation models such as neural networks or Bayesian networks could easily be substituted for the tables. Once the tables are constructed, reconstruction of the drawings becomes a standard optimization problem to find depth

values for each vertex that maximize the correlation functions.

Lipson and Shpitalni exhibit a number of successful reconstructions of planar objects in their paper, both as computer graphics renderings and physical rapid prototypes created by feeding the models to a milling machine. Their results are impressive, but it's unclear if their technique can perform at the interactive rates that would be required by a real world user. To be fair their system is only a prototype, but even they admit that the optimization process is difficult. They mention trying several well-established techniques including hill climbing, simulated annealing, and genetic algorithms, but the nature of the problem creates a search space fraught with pitfalls. As an example the authors point to the Necker cube illusion—see Figure 5.5. We see this illusion whenever we look at a wire-frame picture of a cube. When edges in the drawing cross there is no indication as to which line is in front and which behind, meaning that there are two perfectly valid interpretations of the drawing.

Fundamental limitations like the Necker cube may seem at first to suggest that the acquired geometric correlation techniques are not suitable for real world applications. However, it's telling that these systems are confused by the very same images that confuse human perception. To this extent Lipson and Shpitalni make the interesting observation that the degree of correlation calculated by their system is, in some ways, a metric of the 'understandability' of a particular drawing. Figures that have a low correlation when compared to the training set also tend to be difficult for human observers to correctly interpret. To be sure, before this technique can be applied in commercial or consumer software more efficient optimization and storage methods will need to be devised.



**Figure 5.5.** The Necker Cube is a classic optical illusion. Look carefully at the cube represented as a transparent wireframe on the left-hand side of the image. By selectively interpreting either the bottom-right face or the top-left face of the wireframe as the front of the cube, this image can be interpreted as a representation of either of the two solid cubes on the right-hand side of the image. Can you see them both?

## 5.4 Optimization and Line Labeling

The idea of constructing a global understanding from a number of smaller clues sounds suspiciously like an optimization problem, a technique that is already put to widespread use in artificial intelligence systems. If somehow the computer could key in to the same kind of visual cues, perhaps that local information could be combined into a global understanding. It was observations like these that led two researchers, David Huffman [Huffman, 1971] and Maxwell Clowes [Clowes, 1971] to independently developed a formal method for reconstructing a specific class of line drawings. Today this method is referred to as *Huffman-Clowes line labeling*.

The system described by Huffman and Clowes deals with a specific set of 3-D shapes called trihedral planar objects, which consist of only flat faces and contain vertices with no more than three incident edges. They also require that the object be depicted from a ‘general position’—that is, from a viewpoint not perfectly aligned with some face of the object such as looking at a cube head on so that it looks like a square. These restrictions may sound constraining, but this class of objects contains a surprising diversity, including many boxy shapes common to architecture and engineering.

Examining these shapes, Huffman and Clowes realized that no matter the overall shape of a model, given three incident edges there are only so many ways the edges can meet at a vertex that make physical sense. These junctions are usually named for their shape, such as T-junction or L-junctions, and each line entering the junction can be assigned one of three labels: *convex* lines are those which would pop out of the page towards the view; *concave* as those moving back into the page; and *occluding* edges, which fall along the outer silhouette of the figure and represent the limits of what the viewer can see from the viewpoint of the drawing.

Huffman and Clowes produced a catalogue (called the Clowes-Huffman catalogue) describing each of the possible configurations of three lines at each of the junction types. Based on these configurations they were able to prove that a valid 3-D projection of a trihedral planar object must necessarily have a consistent labeling. Because each line in the drawing is straight, the label assigned to the line never changes along its length, so the label assigned by selecting a junction type at one end must match the assignment at the other extreme. This means that reconstructing a 3-D shape from a 2-D line drawing is simply a matter of

satisfying a constraint for every junction and line in the drawing.

Although Huffman-Clowes line labeling was originally defined only for trihedral objects other researchers have tried to extend the process to more complex figures. Varley *et al.* mention extended trihedral objects in which three planes meet at each vertex; tetrahedral in which no more than four edges are incident to a vertex; and normalons, in which all of the edges and face normals of the object are aligned with the principle axes [Varley *et al.*, 2004a]. Adding restrictions as in the normalon case tends to work well and can help the system in the reconstruction process, however Varley *et al.*'s article points out that extending line labeling to higher degree figures, as in the tetrahedral case, is more difficult. The problem lies in the explosive growth in complexity involved. With each additional vertex degree the number of junction types in the catalogue multiplies rapidly out of control.

Another major limitation of Huffman-Clowes line labeling is the fact that the figures can only consist of straight lines and planar faces. Straight lines make the reconstruction process easier because a straight line is guaranteed to maintain the same labeling along its entire extent. Curved lines on the other hand may change their labeling from one end to another [Varley *et al.*, 2004b]. Despite the computational advantages, the need to address curves did not escape the early researchers, and indeed Huffman also described a labeling system for smooth objects and curved silhouettes, which was later expanded upon by Williams [Williams, 1997]. This work was the basis of the Karpenko's SmoothSketch contour reconstruction [Karpenko & Hughes, 2006], which will be further discussed in the next section.

Line labeling can be used to deduce a great deal of information about a drawing, but labeling alone is only useful in reconstructing the visible portion of a



drawing, producing what is sometimes called a  $2\frac{1}{2}$ D image, because it contains 3-D information for the front facing geometry, but nothing behind it, as if the figure was mounted flat against a wall. For this reason when line labeling is used it is usually teamed with other related systems that help to fill in this information and create a complete—or as complete as possible—3-D object.

When applied to photographs or depictions of actual objects line labeling, though limited, has proven effective as a means of rudimentary computer vision. Line labeling also seems an obvious fit for sketch-based modeling, however certain considerations must be made when working with human generated drawings rather than exact representations of an object such as a photograph or 2-dimensional projection of an actual 3-D form. The reason is that artists don't always draw geometry consistently, and in fact will occasional create drawing as intentional subversion of the principles behind Huffman-Clowes labelings. Frédo Durand in his paper discussing computer depiction points out that it's possible for a drawing to have a consistent labeling locally, but contain inconstancies globally. Situations like this result in so-called impossible figures [Durand, 2002] in which objects seem to entangle themselves or create matter out of their negative spaces. You're probably familiar with many impossible figures made popular as optical illusions.

Although such aberrant cases exist they are hardly the norm, and among professions who would most likely utilize a 3-D modeling application they are not a concern. More concerning to a line labeling system however are the minor mistakes and inconstancies that are a normal part of any drawing made by a human being. No matter how skilled a draftsman or how careful a graphic designer is, any reasonably complex drawing is bound to contain lines that are not completely straight, vertices that don't exactly connect, and edges that are

slightly off parallel.

Two of the first researchers to recognize the potential of this system and apply line labeling to a CAD interface were Ian J. Grimstead and Ralph R. Martin. Grimstead and Martin developed the first CAD prototype based on a system of Huffman-Clowes line labeling and linear equations that was capable of converting a user's 2-dimensional hidden line drawing into a complete 3-D boundary representation [Grimstead & Martin, 1995]. Because of the limitations of the Huffman-Clowes catalogue their system was only capable of interpreting polygonal objects with tetrahedral vertices. The system also had several other limitations stemming from its prototype status such as dealing with only a single model and disallowing holes or other more complex topological features. As the user draws their model into the system using a digitizing tablet the program tidies up the raw inputs, enforcing features like line straightness, connection, and parallelism. The system then finds one or more labelings and negotiates with the user to select the proper configuration. Once a labeling is decided upon, a system of linear equations is used to determine the z-coordinates of each vertex in the drawing and then using this partial reconstruction the hidden portions of the model are reconstructed to create a full, 3-dimensional model.

Grimstead and Martin's program proved that line labeling could be an effective if constrained means of reconstructing not only photographs and computer generated scenes, but also human created drawings. As a CAD system their prototype was a bit limited, but it paved the way for research on line labeling in the sketch-based modeling community. Varley, Martin, and Suzuki's RIBALD research program provides a good example of a fully-featured, state-of-the-art line labeling reconstruction system [Varley *et al.*, 2004a], which incorporates many of

the techniques developed to support 2-dimensional reconstruction.

The RIBALD system is not a full modeling system, but instead a reconstruction kernel that can be run stand alone for testing, or integrated into an modeling system or interface. RIBALD is capable of reconstructing a single manifold polyhedral object drawn from its ‘most informative angle’. In order to create the 3-D object RIBALD first reconstructs the frontal geometry of a drawing using line labeling and a system of equations to determine z-coordinates. Then the system attempts to classify the drawing into one of several types in order to reconstruct the back-facing geometry. According to the researchers, for certain classes of objects such as normalons, trihedral objects, and extrusions this process is fairly straightforward, however for more general figures in which edges could be placed at arbitrary angles and locations, only very simple drawings can be reconstructed reliably.

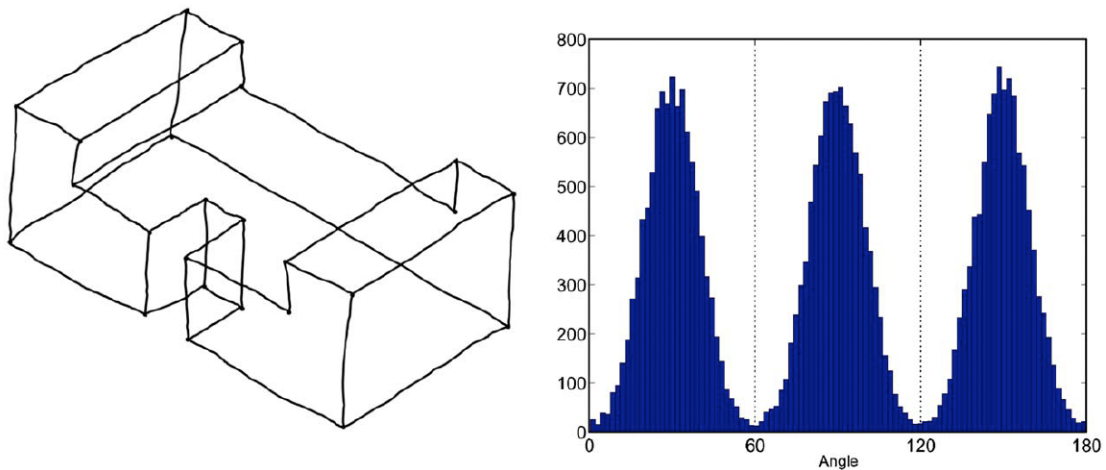
In a second paper Varley, Takahashi, Mitani and Suzuki presented an example modeling system that utilizes RIBALD to create reconstructions of curved objects [Varley *et al.*, 2004b]. In the system they describe, the user first draws a polyhedral template consisting of straight lines and planar faces, which is reconstructed by the RIBALD system. The user can then draw a second object consisting of freehand curves over the template. The curved drawing must have the same vertex/edge graph as the template because this topological information is used to form a connection between the reconstructed topology of the 3-D template and the new curves that define its surface. Using the template information the curved object is reconstructed as a 3-dimensional shape and a triangulated mesh is generated to represent its geometry. The researchers suggest that this method could be an effective means of creating car body designed, product prototyping, and even

facial modeling.

A number of optimization-based methods have been proposed to deal with the inherent complexity and format limitations of Huffman-Clowes line labeling. Masry, Kang, and Lipson provide a prime example of a sketch-based CAD project using a reconstruction system along these lines [Masry *et al.*, 2005]. Branching off from Lipson and Shpitalni's earlier work [Lipson & Shpitalni, 2002] with machine learning, Masry *et al.*'s publication describes a sketch-based CAD program designed around a simple TabletPC interface. The user defines his or her model by drawing a 2-dimensional wireframe representation into the interface. Sketches are required to be connected and to generally conform to an overall orthogonal axis system, but unlike most line labeling system they can contain planar curves. The authors point out that a wide variety of engineering design drawings fit into this set.

Rather than performing a global reconstruction on the user's drawing, Masry *et al.*'s system uses an elegant propagation process. The system first analyses the user's sketch and creates a histogram called the 'angular distribution graph' based on the angles of all of the lines relative to the sketch plane—see Figure 5.6. Because the figures conform to an overall orthogonal axis the distribution of angles in the drawing will manifest as three humps in the histogram, which correspond to the three primary directions. The system then selects the vertex in the sketch that most closely matches this distribution to be the origin of the coordinate system. Once the origin and the three edges incident to it are reconstructed, the reconstruction process can simply propagate out along each edge and through each vertex until depth information for the entire model is determined. For this initial reconstruction, curved lines are replaced by straight lines connecting their

endpoints, however once initial reconstruction is complete the curves can then be constructed based on the 3-D positions of their counterparts and the assumption that each curve is planar. According to the researchers' publication this reconstruction system is capable of generating 3-D models from sketches of as many as 50 lines at interactive rates.



**Figure 5.6.** This figure from Masry *et al.* shows a user's wireframe drawing, and the associated angular distribution graph [Masry *et al.*, 2005]. Three peaks can clearly be seen in the graph, indicating the three primary axes of the lines composing the drawing.

Each of the line labeling techniques we've discussed uses an optimization process to narrow down the field of possible reconstructions. Optimization has proven to be an effective technique at combating the explosive solution space of reconstruction, but even the most effective optimization can't completely abate the underlying ambiguity of the problem. As we pointed out earlier the solution space for reconstruction is scattered with local minima and maximum. Even when a valid solution is found—as exemplified by figures like the Necker cube—several solutions may be equally valid. Most of the projects we have seen combat these issues by presenting the user with a list of alternatives, or simply by picking a

convenient solution and relying on the user to correct the system when it's wrong. For many applications such as technical CAD drawings this kind of interaction may be sufficient, but for others it can be unsatisfying or even irritating.

A number of researchers have suggested providing the reconstruction system with contextual information in an effort to produce a more accurate result. This has the added bonus of allowing the designer to also work within that context, allowing them, for example, to add details to a larger project or experiment with different solutions in place. The Stilton system developed by Turner, Chapman, and Penn is a good example [Turner *et al.*, 1999].

Stilton has two features that set it apart from the other reconstruction systems mentioned previously. The first is the way in which drawings are collected and processed by the system. Stilton is built around a VRML browser that displays a pre-constructed 3-dimensional scene to the user. The user is free to walk around in the space and add new geometry into the context of specific locations by making a wire-frame drawing of a planar model where he or she wants the new geometry to appear. The system then reconstructs their drawing into a 3-D shape and integrates it into the VRML scene.

What's unique is that because the drawings are placed in the context of the VRML model they are drawn in a perspective view similar to perspective drawings, which is more familiar to architectural artists than the orthogonal view used by the other systems. The idea is that an architecture or design firm could load the planes of a building or space into the system and then designers could move through the environment creating smaller details like architectural accents or furnishings.

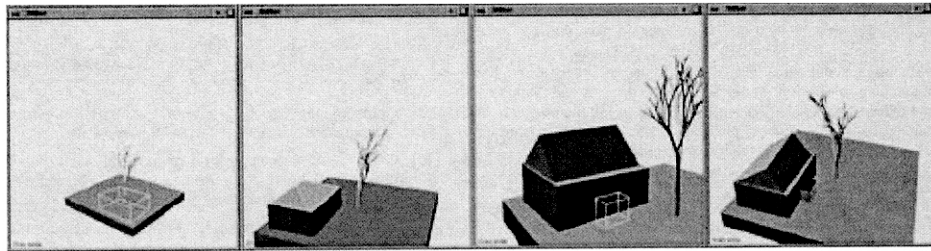
Working in a perspective view has a number of consequences for the reconstruction process. The most obvious factor is that parallel lines when drawn in

perspective no longer look parallel. To give the illusion of depth the relative distance between parallel lines slowly decreases until the lines eventually intersect at a vanishing point. Without a frame of reference as to where the vanishing points are, reconstructing the flat faces of the user's model becomes extremely difficult. Fortunately by allowing the user to draw within the context of an existing model Stilton can read a frame of reference from the surrounding objects and use that information to help reconstruct the user's drawing.

The reconstruction process itself constitutes Stilton's other unique feature. Like most of the other systems Stilton uses an optimization process to find plausible depth values for each of the vertices of the user's drawing. However rather than dealing with the large search space in a methodical fashion Turner *et al.* instead apply genetic algorithms, which generate possible solutions from random inputs, and then combine and regenerate the solutions over and over until a viable solution is found.

In their publication the researchers provide several examples of models created with their system. They also explain that Stilton can be used not only with VRML models but also photographs annotated with a minimum of positioning information. This allows designers and architects a great deal of flexibility in the kinds of context in which they can work. The major drawback to Stilton is the processing time required by reconstruction, which the authors' publication describe as on the order of ten seconds for even a simple cube or prism. However through their experiments working with the system the authors suggest that it's more effective to take an incremental approach to constructing a model through a number of discrete primitives than trying to draw the whole thing at once. Not only is this easier for the program to interpret, they point out, but it also gives

the designer more feedback about the progress of their work. An example of a typical construction process is pictured in Figure 5.7.



**Figure 5.7.** An example of the incremental construction of a house using Turner *et al.*'s Stilton modeling system [Turner *et al.*, 1999].

Although line labeling and reconstruction systems lend themselves to models that are primarily angular and contain planar surfaces, this does not mean that line labeling techniques can't be applied to domains that demand a measure of fluidity and smoothness. Davis *et al.* for example were able to apply a specialized version of line labeling to character animation to allow 3-D animators to quickly create motion sequences by drawing simple stick figures [Davis *et al.*, 2003]. By narrowly limiting the interpretation of a user's sketches down from any possible axis aligned 3-D shape to a specifically defined stick figure many of the issues with labeling such as efficiency and scalability are no longer real concerns. The user creates their animation by first defining a skeletal template, a stick figure image of their character parallel to the image plane that defines the definite length and orientation of each of the 'sticks' or 'bones'. The animator can then create his or her stick figure animation on paper and scan them into the system, or draws them directly in the program with a provided editor. The program then analyzes each frame, matching the components to the template and reconstructing plausible orientations for each joint based on the apparent foreshortening of the bones in



the drawing.

Given the inherent limitations of line labeling and related optimization techniques, the flexibility and creativity of the various sketch-based modeling application that use this process is astonishing. For some limited applications in which the desired models consist of planar trihedral volumes or other related shapes, line labeling might be a workable solution, however for general modeling the exclusion of curves and the heavy computational cost of these methods make them infeasible as a basis for a modeling interface. As the work by Davis *et al.* suggests, line labeling may find more use in tangential systems or as an abstracted interface for manipulating more complex attributes in an intuitive way.

## 5.5 Blobby Inflation

Most of the early efforts in sketch-based modeling used a stroke-based interface to augment the process of creating traditional CAD models for designers or architects. There is no question that CAD applications are an important class of design program and an obvious target for sketch-based modeling. However, as any 6-year-old can tell you, sketching and drawing are not just work-hour activities. One of the first sketch-based modeling research efforts to take a more whimsical approach to the process was developed by Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka at the University of Tokyo and Tokyo Institute of Technology. Igarashi *et al.* developed a prototype system for creating charmingly bulbous 3-D models from simple drawing input using an ingenious system of inflation. The models produced look like inflated balloons or stuffed plush toys, and so Igarashi and his team named the program Teddy [Igarashi *et al.*, 1999], even using a teddy



**Figure 5.8.** (left) A user creates a 3-D model of a teddy bear from silhouette strokes using Igarashi *et al.*'s Teddy modeling system [Igarashi *et al.*, 1999]. The system ‘inflates’ the user’s silhouette strokes to create a triangle mesh model. (right) Several example models from the system. These models have been colored with a commercial texture mapping software, however later versions of the Teddy system included the ability to color models directly within the program.

bear model created with the system as its mascot—see Figure 5.8.

The main feature of Teddy is its inflation system. The user begins by drawing a simple, 2-dimensional outline of the model they wish to create. The user’s stroke is collected as a closed polyline loop, and then analyzed by the system to find a central chordal axis or ‘spine’, a single branching line that passes through the middle of the closed shape like a skeleton. The closed shape is then converted into a series of triangles, and based on the average distance between the spine and the exterior, the spine vertices are elevated away from the plane of the outline. These points are used to create a triangular mesh dome that is mirrored to the other side and sewn together to create a symmetric watertight model. This inflation process produces the characteristically rounded and bulbous shapes of Teddy models.

Once a model is created Igarashi *et al.*'s system includes several modeling operations that allow the user to make further changes. Rather than a standard interface of buttons and menus, because the program has only a few simple oper-

ations, Teddy features a minimalist interface using a few simple gestures to signal commands. First the user can paint simple 2-dimensional features on the surface of the model. The user's strokes are actually painted onto a planar surface, and then projected into the model. The user can also add 3-dimensional features to the model by first drawing a closed stroke on its surface, shifting the view and drawing a bounding stroke. The closed stroke is extruded into the bounding stroke to create a protruding feature. The same method can be used to create digs or depressions in a model's surface as well. Finally the user can make straight cuts or curved 'bites' in a model by making a stroke crossing its surface, and 2-dimensional surface features can be removed with a simple scribble gesture.

The original version of Teddy met with a great deal of positive feedback in the computer graphics community. Working with a number of research colleagues Igarashi has continued to develop and improve the system. In 2001 Igarashi introduced a texture painting system with a simple, Teddy-style user interface called Chameleon that allows users to paint their Teddy models in a range of bright colors [Igarashi, 2006]. One of the drawbacks of the original system was that the mesh models created had wrinkled and bumpy surfaces. Igarashi and John Hughes addressed this issue in a 2003 publication that introduced a new version of Teddy combining the features of the original and the Chameleon painting program, along with a new mesh-smoothing algorithm [Igarashi & Hughes, 2003]. The smoothing system is based on Markosian *et al.*'s skin algorithm [Markosian *et al.*, 1999]. The algorithm reprocesses the original mesh, evening out its vertices so that it contains nearly equilateral triangles with a more uniform distribution, making the models appear much smoother. Finally Igarashi, Moscovich, and Hughes created a Teddy like simple interface for key-frame animation that allows users to create

simple cartoons with their Teddy models [Igarashi *et al.*, 2005].

When compared with the full-featured 3-D modeling systems most people think of, Teddy is severely limited. Only a single model can be created at a time, and the subject must have a simple topological structure with no holes. Furthermore many of the familiar editing tools such as Boolean operations or control point adjustment are simply not available. Finally models are restricted to objects like balloons and stuffed toys that can be described as bulbous inflations; they cannot contain detailed features or sharp edges and creases. However these observations are hardly shortcomings. Though the original publication mentions a variety of plausible applications for the program, Teddy is obviously designed primarily for entertainment value, and its feature set and simple interface make 3-D modeling accessible even to children.<sup>3</sup> The same features that make Teddy intuitive even to young children suggest that the system can tap into the same creative process as paper-and-pencil (or crayon) sketching.

Igarashi *et al.*'s Teddy papers proved to be inspirational to a wide variety of projects that broadened the horizons of sketch-based modeling to include not only the regular forms of CAD like design, but also more whimsical models. Igarashi, Matsuoka, and Tanaka's original paper suggests several avenues for extension and future work, specifically mentioning how the same interface techniques could be applied to other modeling representations. Perhaps the first project to explore this territory is described in a publication by Olga Karpenko, John Hughes, and Ramesh Raskar [Karpenko *et al.*, 2002]. Taking cues from Teddy, the team developed an inflation-based modeler built around an alternate surface representation that makes many of the modeling operations easier to compute.

---

<sup>3</sup>According to the author's web site Teddy has also been converted into a videogame for Sony's PlayStation2 gaming console [Igarashi, 2006].

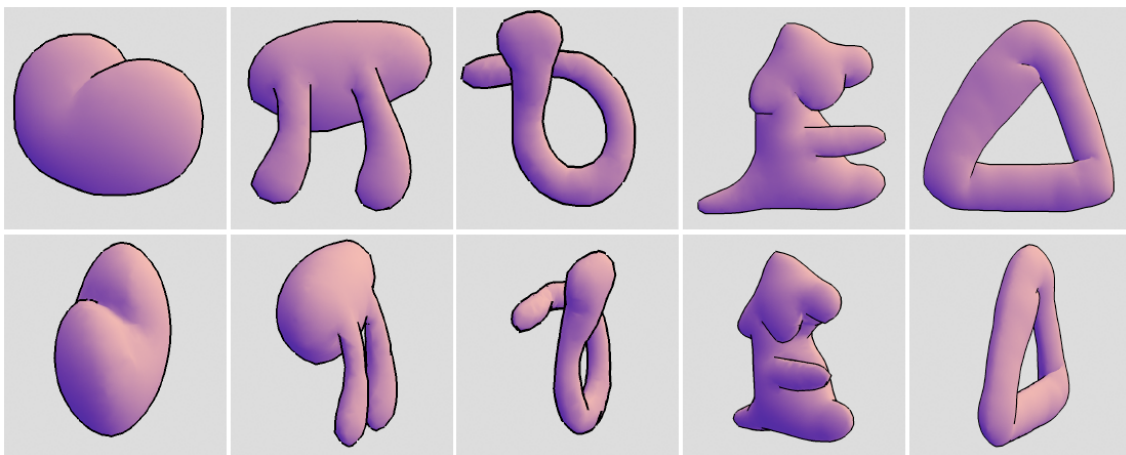
Karpenko *et al.*'s primary observation was the inherent limitations of Teddy's polygonal mesh surface representation. Because models in Teddy are composed using this structure, certain modeling operations, especially CSG operations like intersection and merging, are made computationally and logically complex. In addition, Teddy's inflation methods produced models with bumpy and craggy surfaces. To address these issues, Karpenko's research team utilized an implicit volumetric representation called *variational implicit surfaces*, a technique based on *radial basis functions*.

Within the system, a model is created by drawing a single contour curve representing the silhouette of the shape the user wishes to create. Like Teddy, this silhouette curve is inflated into a 3-D model, however rather than deriving the thickness of the model from the diameter of the silhouette alone, Karpenko *et al.*'s system analyzes the aspect ratio of the user's input and tunes the thickness of the model accordingly. Thus compact rounded objects are converted into more spherical 3-D forms while longer shapes are converted into cigar-like volumes with a circular cross section.

The initial silhouette must be defined as a single line, however once the shape has been inflated, the user can make minor alterations to its 3-dimensional exterior with a simple stroke-based editing method. Karpenko *et al.*'s system also leverages the ease with which implicit surfaces can model CSG operations. Whereas Igarashi *et al.*'s system was limited to a single model, user can create multiple shapes within the same modeling session and combine them into a simple modeling hierarchy. Once two shapes are adjusted to overlap, they can be merged, either directly, or through guidance strokes provided by the user, which create a smooth fillet between the two surfaces.

The research team also incorporated a number of innovative interface features. Users can use a symmetry operation to create, for example 4-legged models, more quickly. An innovative system of transformation and positioning is also provided. Users can rotate the model by adjusting a bounding sphere, but instead of dragging the model directly to change its position, users instead drag the model's shadow projected onto a ground plane.

In 2006 Karpenko and Hughes published an updated paper detailing further work on the system [Karpenko & Hughes, 2006]. The new program they describe—now dubbed SmoothSketch—allows users to define the silhouette of a model using multiple strokes. Where strokes overlap, creases are formed on the inflated model's surface, making it possible to create shapes with cusps or dents like a bean or heart—see Figure 5.9. Based on line labeling work by Williams [Williams, 1997], Karpenko and Hughes were also able to develop a method of constructing models with holes, t-junctions, and other shapes where the endpoints of a contour may be hidden.



**Figure 5.9.** A series of example models from Karpenko *et al.*'s SmoothSketch system.

Thanks to its alternative model representation, Karpenko *et al.*'s system addresses many of the issues with Igarashi *et al.*'s Teddy, however the radial basis functions are not without their own limitations. The most important is that models developed with the system cannot contain sharp or straight edges, a consequence of the underlying radial basis function's surface representation. The use of implicit surfaces also imposes certain performance penalties that preclude SmoothSketch from the construction of large or complex model. Generating the representation for each model requires the solution of large systems of equations, and once developed it is expensive to evaluate the radial basis functions in order to visualize the surface.

Ohwada *et al.* [Ohwada *et al.*, 2003] also investigated the use of volumetric model representations for inflational modeling. Similar to Teddy and SmoothSketch, models created with this team's system are defined by closed silhouette strokes, which are inflated into rounded models. Adding to this functionality, once a model is created users can extrude shapes out of or into the model's surface. The user first draws a profile shape on the model's skin, then from an orthogonal angle, defines a second profile shape indicating the path of the extrusion. The path can extend out beyond the model's current boundaries to create a protuberance, or can pass into the model, creating a dent, internal cavity, or even hole. To aid in this construction process, after a contour stroke is defined the system automatically rotates the model to provide the user an orthogonal view in which to define the silhouette curve.

An alternative implicit representation called convolution surfaces was proposed by Tai *et al.*, and demonstrated by their prototype modeling application ConvoMo [Tai *et al.*, 2004]. Similar to Igarashi's inflation method, which utilized a chordal

axis to construct an outer polygon mesh, the convolution surface representation analyses the user's closed boundary strokes to generate a skeleton. An implicit kernel is then convolved along this skeleton network to generate the model's volume and define its surface. The main advantage of this system is that it allows the user to draw his or her own cross sectional shape rather than being limited to the roughly circular cross sections common to the other inflation systems. By defining cross sections with tight curves, the user can create what the authors call 'semi-sharp' features, which help to expand the modeling vocabulary beyond rounded blobby shapes.

A limitations shared by many of the implicit and volumetrically based inflation systems is an upper bound on the complexity of models due to the computational or storage costs of the representation. Several projects have tried to address this issue directly. A good example is provided by ShapeShop. Developed by Schmidt *et al.* [Schmidt *et al.*, 2005], ShapeShop uses a hierarchical modeling representation called 'BlobTrees'. BlobTrees are similar to CSG modeling trees in which implicitly represented inflated modeling components form the leaves of a binary tree whose internal nodes are construction operators that blend, intersect, subtract or merge the components into a combined model. The system also supports simple linear and rotational sweeps to create volumes from the user's 2-dimensional curves. By combining these components with the BlobTree operators, the system is capable of creating models that contain truly sharp edges and flat faces, something that none of the other projects discussed thus far are capable of representing. Because of these features, ShapeShop provides an interface that is much more suitable for generalized modeling.

The inflation based modeling systems provide a prime example of how sketch



based modeling can be used in more whimsical ways. Although the modeling vocabularies of early systems like Teddy are limited, by focusing on simplified interfaces and drawing as a primary input method, users are able to look beyond these limitations and create fanciful and imaginative models. We can also see from later examples like that of Schmidt *et al.*'s ShapeShop that these blobby inflation techniques can still form the basis of a more advanced and well rounded system.<sup>4</sup>

## 5.6 Height-Fields and Shading

Up until this point we have considered sketching and computer sketching systems that revolve around linear or basically linear strokes almost exclusively. We focus on stroke for several reasons. First, traditional paper-and-pencil sketching is done primarily with lines because nearly any tool can make a line of one sort or another. Second, lines are also relatively straightforward to represent within a computer. Ranging from very basic lines to complex parametric representations, the mathematics of lines are well understood and primitive graphic elements to represent them are a standard component of every available graphics system. However, lines are not the only way in which artists define a 3-D shape within a 2-dimensional drawing.

Lines visually define a boundary or discontinuity in a figure, and so lines are a very effective means of dividing an object from its surroundings or specifying its features. This is why lines are used, for example, to form the silhouette of objects in most sketches, and also to show sharp features like edges and corners. These very properties that makes lines well suited for stark and contrasting visual

---

<sup>4</sup>No pun intended!

transitions can make them poor tools for defining gradual gradations in shape, depth, or texture. Consider, for example, how you could draw something blobby and amorphous like a lump of soft mashed potatoes. You could certainly draw its outline, and perhaps a few of the stiffer peaks, but it becomes more difficult to render the subtle surface changes using only lines. Don't feel bad. This is the same reason that drawing a likeness of someone's face is so difficult. Although our faces have definite features with clear elements those features aren't stuck together with definite boundaries like Mr. Potato Head. Instead our nose smoothly blends into our cheekbones and out over our lips and chin. Human faces have very few solid edges.

In order to describe the smooth organic transitions in subjects like faces, spheres, and mashed potatoes, artists use a technique called shading, a smooth gradation of values from light to dark that gives the illusion of a smooth surface. In fact, as most artists will tell you, there really are no such things as lines in real life. What we perceive as the line between, for example two faces of a cube, is in reality only the boundary between the two values on the cube's surface. Although artists who favor ink or pencil make use of lines, painters for example rarely use literal lines in their work, but rely upon the contrast between different colors to define the edges of objects or their features. A similar technique is also used in graphite or charcoal to render objects in a more photorealistic style. When artists use lines in a drawing they serve as a convenient and universal shorthand to describe these abrupt transitions in value.

From a physical perspective, the shading used by artists is a representation of two properties of an object: the material the object is made out of, and the distance of any part of that object from the viewer, relative to an ambient light

source. The gradual change in shade corresponds to the direction in which light bounces off of the object's surface. In the same way that a traditional artist uses shading to give his or her works a 3-dimensional quality, modern 3-D computer graphics systems shade 3-D objects to make them look more realistic. The mechanism by which the 3-D system determines how to shade an object is called a *lighting model* because it simulates the way light interacts with objects in a scene.

A number of different models have been developed, some focusing on efficiency for real time rendering and others on physical simulation to produce more realistic results, but in most cases the basic idea is the same. To each object in the scene the designer assigns a set of material properties that describe the way light should interact with the surface of the model. Each model is also equipped with one or more normal vectors, unit length vectors that for any point on a model's surface are orthogonal to the model and point away from its surface. The designer then positions one or more light sources within the scene, just as if they were physical elements. Once the scene is set, the 3-D system simulates light striking the surface of each model, and then adjusts the color and brightness of the surface at that point based on the surface normal, the direction of the light, the material properties, and the position of the viewer. Many researchers have recognized that the depth and surface normal information described by an artist's shading offers a potential source of information that could be used in reconstructing the 3-dimensional shape of an object from a 2-dimensional shaded image.

The original intention of the lighting model was to simulate as closely and efficiently as possible, the actual physical properties of light in order to make models appear more realistic. However, as lighting systems were developed and researchers began to experiment with their use a number of computer scientists

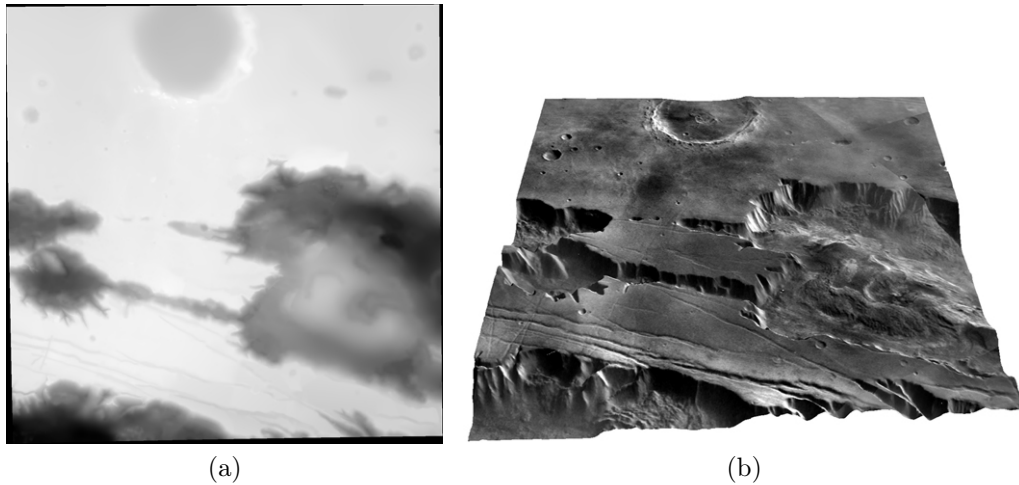
recognized that designers could use this additional layer of lighting and material information as a medium for creativity. Blinn for example recognized that by perturbing the surface normals of an otherwise smooth model it is possible to simulate small surface features [Blinn, 1978]. When combined with a lighting model these small alternations in normal vector directions create a pattern of light and shadow on the surface that resembles texture, but without the computational and storage expense of actually defining small surface elements in geometry. These *surface perturbations maps*, or simply *bump maps* are now widely used in 3-D modeling software to add interest to the flat surfaces of walls, vehicles, or scenery.

Because bump maps do not make any actual changes to the geometry of the model, their effect is only an illusion in the same way that something drawn on a flat piece of paper can be made to look 3-dimensional by shading it. This process can be taken one step further into displacement mapping, which rather than disturbing the surface normals actually changes the geometry of a surface based on a displacement map [Williams, 1990]. However, because of the complexities of altering the surface geometry, displacement mapping is more difficult to perform.

We noted above that shading is a result of two properties: materials and position. Bump maps exploit the material properties of an object—in this case its surface normals—in order to create an effect. It's also possible to adjust a model by working with its position, and specifically its depth. In the computer vision and modeling community, one way of representing this kind of depth information is called a *height-field* or *digital elevation model* (DEM). In the same way that a grayscale raster image assigns an intensity value to each pixel within the composition, a height-field image assigns a value to each pixel corresponding to the distance between the viewer and the surface of an object. This height-field

data can then be interpreted as a grayscale image, usually with lighter values corresponding to points closest to the viewer.

Height-fields have been used extensively by cartographers and geologists to create highly accurate topographical maps. Height data can be collected through remote sensing by projecting a laser or radar beam from highflying aircraft or satellites, but it is also possible to reconstruct height information from pairs of stereoscopic images, a technique used by NASA's Viking space craft to create maps of the surface of Mars [USGS, 2003]—see Figure 5.10. Closer to earth, researchers have developed 3-dimensional scanning machines that sit stationary or orbit around a small object such as a vase or a person's head, painting the subject with a laser beam, which records the distance from the scanner to the object. This depth information can then be loaded into a computer program to create a rough 3-dimensional model of the subject.

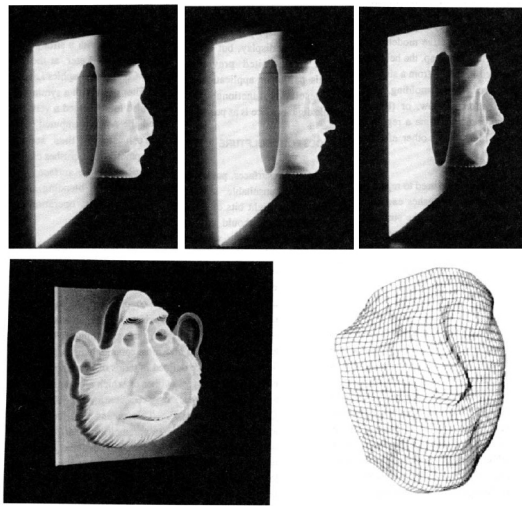


**Figure 5.10.** (left) Digital elevation model (DEM) of the West Candor Chasma region of Mars. (right) 3x vertically exaggerated 3-D surface rendering of the West Candor Chasma region of Mars. Note how dark regions on the left image become low points on the right image. Images generated from stereo image pairs taken by the Viking probe by the U.S. Geologic Survey for the National Aeronautics and Space Administration [USGS, 2003].

One of the researchers involved with developing displacement maps also took an interest in working with height-fields directly. In a 1990 paper, Lance Williams of Apple's Advanced Technology Group described a system called 3DPaint that allowed users to edit height-field data by loading the height-field image files into an image editing application [Williams, 1990]. In his publication Williams describes how various painting tools such as smearing brushes and lasso selection can be used as an easy means of manipulating the height-field data, enabling a user not only to edit existing data, but also to create a model from scratch, simply by painting into the image editor with light or dark values. Once the desired height-field is created, Williams's system converts the height-field into a 3-D model using square Coons patches, which describe the surface as a meshwork of spline curves.

In one particularly effective demonstration Williams shows a scan taken of a human face on which the nose has been flattened due to quantization. The face is

shown in profile on a specialized 3-D viewing system called a video raster surface display that depicts the depth of the model as a series of elevated scan lines. Williams described how the height-field image was loaded into an image editing application where the nose was corrected simply by painting the effected area with digital white paint. Then using features of the paint program like a smudging tool, Williams was able to quickly and plausibly reconstruct the nose.<sup>5</sup> In the same publication Williams also demonstrates how the technique can be used to create a model from scratch by simply painting a height-field into the image editor and processing the results with the 3DPaint system—see Figure 5.11.



**Figure 5.11.** Examples of the 3DPaint program. Models in black are displayed on the raster surface display. Top: (left) original scan of human face with blunted nose, (middle) high point added with paint program, (right) new nose sculpted with smudge tool. Bottom: (left) hand sculpted face model, (right) Coons patch model of human face. [Williams, 1990]

Williams’s 3DPaint system demonstrated the utility of working with height-fields, not only when adjusting existing models, but also of creating new geometry.

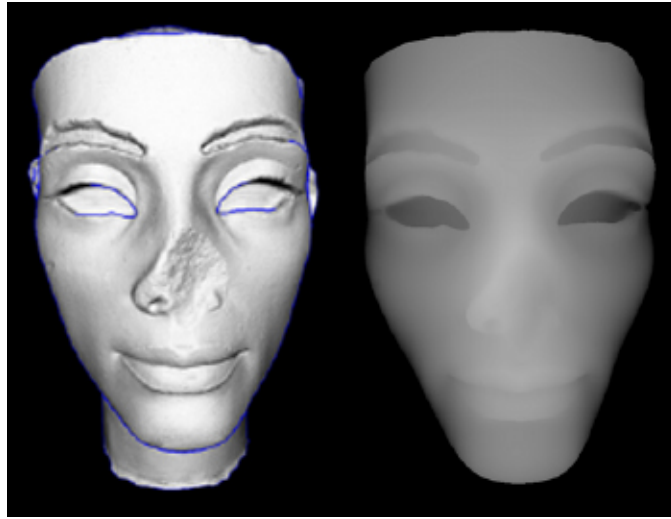
<sup>5</sup>In his publication, Williams dryly notes the obvious applications of the system for plastic surgeons.

However, as Williams himself points out, working directly with height-fields has some difficulties. A primary problem for the designer is the fact that, although height-fields resemble shaded artwork, their light and dark tones have a different meaning, representing the relative distance of the object from the viewer rather than the positions and intensities of light sources. Making this distinction and working in height rather than light can be a tricky conversion. As Williams puts it, “Our visual systems are adapted to interpreting surface shading as the local orientation or color of a surface, not its elevation.” Because humans, and especially artists, are so used to working with shading as shadow, it can be difficult to express the desired model in a height-field.

Rushmeier *et al.* point out a second issue with height-fields [Rushmeier *et al.*, 2003]. Unlike rendering an object with shading in which the direction of the light source can be adjusted to highlight the object’s features, the value scale of a height-field image is always oriented towards the viewer. Interpreted (incorrectly) as a shaded image this corresponds to a light source shining directly from the viewer’s position, a direction which tends to wash out many small details. You can see an example of this phenomenon in Figure 5.12. This is a problem we’re all familiar with. The effect is the same one we see when taking a flash photograph of a friend in a dark room. The brightness of the flash tends to wash out the subject’s facial features, making them look pale and flat. This effect makes it difficult to perform any fine adjustments to a model in the height-field.

Given the fact that height-field data and shading are both visual manifestations of some of the same physical properties, you may be wondering if it’s possible to reconstruct a 3-D model from its shaded image, just as the model can be recovered from height-field data. It turns out that it is, although with substantially more



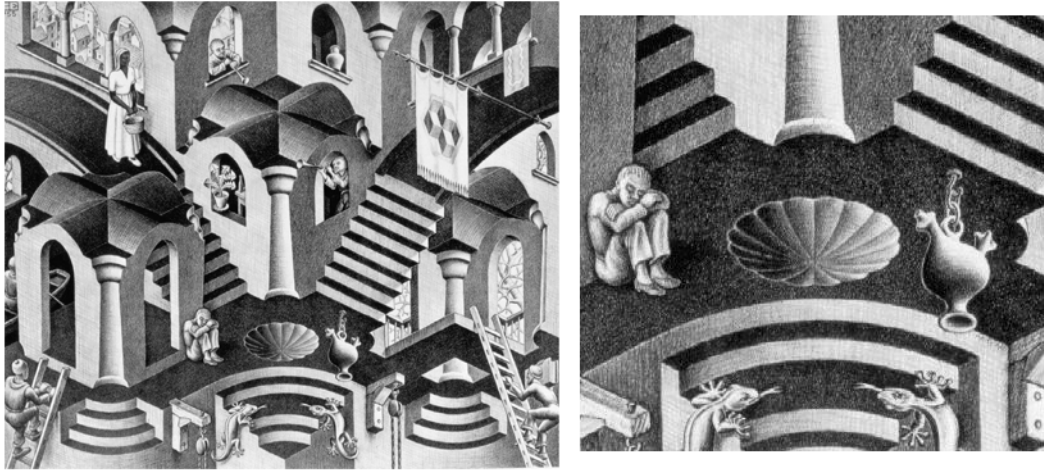


**Figure 5.12.** The face on the left is a shaded 3-dimensional model. The one on the right is the same head rendered as a height-field. Notice that although both faces have damaged noses, the damage on the height-field is difficult to see. Image taken from Rushmeier *et al.* [Rushmeier *et al.*, 2003].

work than height-field recovery. In the computer vision community there are a number of algorithms that attempt to do just this. In the literature the process is known as *shape-from-shading* [Rushmeier *et al.*, 2003]. These algorithms can be used to extract the 3-D shape of an object from its photograph. Unfortunately, the shape-from-shading problem turns out to be a very difficult one, and although these algorithms exist, like the line optimization methods from Section 5.4, most come with substantial limitations. Discontinuities in the photograph of a subject, or changes in the texture or reflectance properties of its surface can confuse the algorithm. Furthermore, because the shading is a property not only of the position and shape of the object but also the location of the light source, even under ideal conditions it is necessary to know or estimate additional information about the scene such as each light's position and intensity in order to make a successful reconstruction.

As if these issues weren't enough, trying to reconstruct a shape from a shaded drawing rather than a photograph creates a whole host of additional problems. Right off the bat there is an obvious issue in defining the position of a light source for a drawing. Artists certainly make every effort to correctly position shadows and shading within their sketches to suggest the position of some imaginary light source, but because this light does not really exist, or is only an approximation of ambient light cast on the artist's physical subject, it can be difficult or impossible to reconstruct. These inevitable mistakes and inconsistencies in the drawing make the task of reconstructing an accurate 3-D model nearly impossible. Kerautret *et al.* point out several other specific issues [Kerautret *et al.*, 2005]. First, not every pattern of value corresponds to a geometric shape. There is no single 3-dimensional shape for example that corresponds to a solitary black splotch couched on a field of light values. Any reconstruction system would need to somehow deal with 'impossible shadings' like these. A second issue is this: when illuminated by a single light source from the viewing direction it can be difficult to determine if the shading information describes a concave or convex object. This illusion, like the Necker cube, has been used by artists for centuries to create interesting visual effects—see Figure 5.13.

Despite all of this doom and gloom surrounding shape-from-shading, all is not lost. While it may be impossible to recover a model from an arbitrary sketch, if the input images can provide additional information or be constrained in some way, then the reconstruction process is viable. A particularly elegant application of the constraint method is a system developed by Rushmeier *et al.* to help archeologists reconstruct damaged artifacts as 3-D models [Rushmeier *et al.*, 2003]. The insight of the researchers was that if you could tightly control the circumstances



**Figure 5.13.** (left) Dutch graphic artist M. C. Escher’s *Concave and Convex*, Lithograph, 1955. (right) Detail from the center of the same image. Looking at this shading, it’s possible to interpret the scalloped shape in the center as either a depression on a floor plane—corresponding to the figure seated to the left—or a rounded shape pushing outward from a ceiling plane—suggested by the vase hanging to the right.

of creating a rendered image from an object, then many of the variables that make the reconstruction process so difficult become known quantities, significantly simplifying the task of reconstructing the model.

The system described by the researchers begins with an existing 3- dimensional model created from 3-D scanning technology, a common practice for archeologist studying fragile or rare items. Small statues or tools can be placed in a scanning device and recorded at very high resolutions to create an extremely detailed model of the object. This model can be freely studied or transported to researchers at remote locations without risking the fragile original pieces. These scans, however, present a problem. In order to record the fine details on the object’s surface the scans need to be taken at high resolutions, producing models that are very large and difficult to render or display. For archeologist studying only small portions of the model at a time this is not a problem, but many would like to be able to

adjust the geometry of the models in order to repair damage and reproduce the object as it might have once appeared. The sheer size and detail of the model makes importing data into any standard modeling software hard, but even then adjusting the model to maintain the same level of surface details in new areas can become almost overwhelming.

The system developed by Rushmeier *et al.* simplifies this process by changing the task from one of 3-D modeling, to one of photo manipulation. In this way the archeologist can use familiar photo editing software tools like clone brushes and filters to reproduce the same details found in undamaged parts of the model in their reconstructions. To begin the process the user views a simplified 3-D rendering of the model in a small browser, and selects a smaller portion of the model on which they wish to work. The system then re-renders that section of the model in full resolution and creates a standard 2-dimensional grayscale raster image that can be imported into any photo editing application. Within the editor the user can make whatever changes are necessary, and then re-import the adjusted image, which is used to augment the model.

The key to the reintegration process is that the original image was based on a rendering created by the system, meaning that the exact and unambiguous size and orientation of the model, and the positions of any and all light sources are a known quantity. While this information does aid in the reconstruction, it is necessary for the user to provide the some additional information to the system before reconstruction can take place. Along with the shaded image the user must also adjust a diffuse reflectance map, and in the case of larger scale adjustments the user can also edit the height-field to give the algorithm clues about larger changes in shape.

Continuing with our olfactory theme, in their publication Rushmeier *et al.* demonstrate how a statue's nose can be repaired by combining the shaded rendering with a standard photo of an actual person. In the same way that a late night comedian might mix the features of two movie stars to speculate on the appearance of their hypothetical offspring, a user can replace the damaged nose with the 'donor's' nose, and then reconstruct the 3-D model.

Rushmeier *et al.*'s approach has the advantage of making the reconstruction process straightforward for the user. However, because the system relies on the fact that the program itself produced the source image for its shape-from-shading algorithm, the application is limited to only editing existing models. Kerautret *et al.* in examining the approach point out that the Rushmeier method is effective for small-scale repairs, but is not suited for creating new geometry from scratch [Kerautret *et al.*, 2005].

So we return to our original question. Is it possible to construct a 3-D model from a shaded sketch? As we have seen, the inherent ambiguities in the shape-from-shading problem mean that a naïve approach does not provide the computer with enough information to perform the reconstruction, a property that prompted Rushmeier *et al.* to look for ways of reducing the level of ambiguity. Another option, and the approach taken by Kerautret *et al.*, is to explicitly provide the system with additional information. Although a single shaded sketch can be ambiguous, by combining a number of shaded sketches, each drawn with a light source from a different angle, the system can extract enough information to reconstruct a complete height-field, and from that height-field a  $2\frac{1}{2}$ D model.

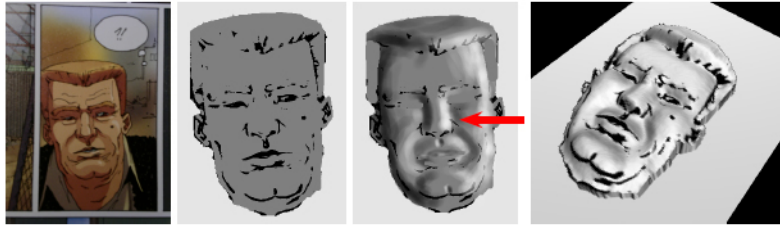
In order to create a model, the user first creates a contour image, basically a linear outline of the silhouette and features of the model they will be creat-

ing. The linear contour image serves both as a framework for the user on which they can create their shading designs, and a framework for the reconstruction algorithm, indicating shading discontinuities that will be maintained in the final model. After the contour image, the user must supply the system with one or more ‘shading solutions’ that simulate the play of light from a specific direction across the 3-dimensional features of the model. Kerautret *et al.*’s publication is not clear on exactly how many shading solutions are required, or the effects that varying amounts might have, but it alludes to the fact that the additional shading provides the system with the necessary information to deal with otherwise ambiguous reconstructions.

Kerautret *et al.*’s system can be used much in the same way as Rushmeier *et al.*’s to make adjustments or reconstructions of existing objects, this time from photographs and producing  $2\frac{1}{2}$ -D height-field models. However the real advantage of the system is that the contour and shading images can come from virtually any source. This includes photographs, scans, and 3-D renderings, but also freehand sketches and drawings. Users can literally begin with a blank canvas and create a 3-D model entirely from their imagination.

As a demonstration of this ability the authors apply their system to an outline image extracted from a comic book. By painting in a simple shading solution the authors were able to construct a more three dimensional face and then map the original image to the new model along with 3-D lighting effects to create a more realistic looking image—see Figure 5.14

One drawback to Kerautret *et al.*’s system is the fact that a user must usually supply an number of shading solutions in order to generate a model. The authors liken this process to an artist drawing preliminary sketches of a subject from sev-



**Figure 5.14.** Demonstration of Kerautret *et al.*'s shading design system. The image on the extreme left was extracted from a comic book. Using computer image tools, a global contour image (middle left), and then shading solution (middle right) were created and fed into the shading design application for processing. Based on these inputs, the  $2\frac{1}{2}$ D model on the right was generated. [Kerautret *et al.*, 2005]

eral angles in preparation for a finished work, however drawing and redrawing shading solutions could become tedious or error prone. More problematic perhaps is the fact that the reconstruction process is performed off line, rather than interactively. The authors report processing times of between two and five minutes for the example models they present, calculation times that would probably preclude most experimentation by the user. Still, if the necessary information can be reliably extracted from the user's input drawings then further experimentation may not be necessary.

This sort of  $2\frac{1}{2}$ -D / 3-D depth painting interface to modeling has also found its way into the commercial sphere. First introduced in 1999, ZBrush [Pixologic, 2007] is a sort of hybrid paint and modeling application that uses a unique pixel representation to store both standard pixel information (color and alpha values), as well as modeling information (depth, texture, and material properties). Using ZBrush, the artist can not only paint an image, but also push and pull the canvas surface in and out. This allows the artist to apply lighting effects and other traditionally 3-D modeling techniques to the creation of 2-D artwork. ZBrush also include more familiar 3-D modeling capabilities, allowing this sort of depth-

painting technique to be applied to both 2-D and 3-D substrates. However, the specialized pixel depth information used internally by ZBrush to create its effects are not available outside of the application, meaning that artwork or models generated with ZBrush can not be easily exported and used in other standard 3-D applications.

Although applications like ZBrush make effective use of this sort of modeling interaction, as we can see from the efforts of researchers like Rushemier *et al.* and Kerautret *et al.*, as the focus of a modeling interface this method of creating 3-D geometry has some serious drawbacks. Working directly with height-field data is effective for generating minor details, but for general modeling its visual interpretation is both unintuitive and uninformative to the artist. Working with shading information may be more familiar to the artist, but for the computer this mode of interaction complicates matters beyond the range of an interactive application.

On a more fundamental level, height-field and shading design systems have difficulty in creating truly 3-dimensional results. Instead, all of the examples—except for Rushmeier *et al.*, which can't be used to create new models—produce  $2\frac{1}{2}$ -D models with depth and detail in one direction, but a flat featureless back face in the other. Williams's publication describes how a fully 3-D model could be glued together from two  $2\frac{1}{2}$ -D pieces stuck together at a seam, but this limits the kinds of models that can be created. Furthermore, because the height-field is only defined in one direction, that is, away from a flat surface, it's difficult or impossible to define fine details that are not directly orthogonal to the viewing plane. For example, carving out the ridges and valleys of the nose and eyes of a face in portrait might work well, but there is no way to define the inner structure



of an ear, or the shape of the head behind them.

## 5.7 Deformation

As we saw in the previous section, one of the major drawbacks of height-field and shading design systems is the fact that their primary results are  $2\frac{1}{2}$ -D rather than fully 3-D models. This characteristic of the height-field and shading methods is the result of providing, in the case of height-fields for example, the distance from the viewer to the model in a single direction, like pressing a planar sheet of plastic over a 3-dimensional object. Although this is an inherent limitation of these modeling methods, the basic concept raises the question: if we can define a model as the height from a flat plane, why can't we define the model by adjusting height or depth on some other 3-D object's surface?

It turns out we can, in a process called *deformation*. In deformation-based modeling the surface of a 3-D object is interactively pushed in, puffed out, pulled, smudged, smoothed, or otherwise deformed to create the model's features. Deformations are not a new technology in the 3-D modeling arena, but it's their resemblance to physical artistic techniques that relates them to the field of sketch-based modeling. In many ways, deformation systems allow users to 'sculpt' the surface of a model by applying deformation operations like a clay sculptor squishing and manipulating the surface of a block of clay, or a mason or woodworker chiseling and cutting material away from a blank of wood or stone.

As you might expect, algorithmic methods for deformations depend a great deal on the underlying geometric representation in use by the system: implicit surfaces, parametric surfaces, polygon meshes, etc. However for most representations there are corresponding methods that manipulate the underlying data structures

in an appropriate way. Deformation operations can also be classified by the way in which they affect the model, and to what extent the user has control over those effects.

The most general form of deformation are global deforming operations. Global operations can be thought of much like image processing filters, making a single overall change to the model or modeling component in one fell swoop. In many cases these operations resemble the same sort of effects achieved in image filters. For example, a common algorithm used in triangle-mesh based modeling is a mesh smoothing. This operation is used to even out bumps and crinkles in a model's surface [Schroeder *et al.*, 2006], just like a photo editor might apply a Gaussian blur image filter to a photograph to smooth over grain or harsh edges in the picture.

More complex global deformation operations can often be expressed in terms of mathematical expressions. Because of this, modeling systems that utilize mathematical formulations to represent their models such as implicit surfaces make heavy use of this sort of deformation. One of the early example systems to put this sort of deformation to use was developed by Wyvill, Guy, and Galin [Wyvill & Guy, 1998]. Based on the researchers' earlier work in visualization and modeling implicit surfaces, their paper describes how deformations based on spatial warping functions can be used to stretch, translate, twist, taper, and bend the basic blobby shapes of simple implicit functions into interesting configurations. To control their deformations, the researchers borrowed a page from the field of constructive solid geometry, creating a hierarchical implicit volume model representation they call a BlobTree. Like a binary CSG tree, the blob tree stores at its leaves basic model shapes, and then combines them through interior nodes representing both CSG

style Boolean operations, and their own deformation functions.

The BlobTree representation was later combined with sketch-based modeling primitives by Schmidt, Wyvill, Sousa, and Jorge [Schmidt *et al.*, 2005], as discussed in Section 5.5. The resulting system, called ShapeShop, allows users create basic shapes from their simple 2-dimensional boundary sketches, and then combine and manipulate them as modeling components using CSG style operations. Although ShapeShop seems to support some of the same global style deformations described in Wyvill *et al.*, much more focus is placed on local operations controlled directly by the user’s drawing style input. This is likely a reflection of the fact that global operations do not provide the user with the sort of granular control one would expect from working with wood or clay. The desire for more localized control has led many researchers to local deformation operations designed around an idiom of carving or sculpture.

One of the first attempts at this sort of carving idea was presented in a 1991 paper by Galyean and Hughes [Galyean & Hughes, 1991]. The authors’ system used a voxel based volume representation they refer to as a voxelmap—similar to 2-D raster graphics pixelmap or bitmap image. The vertices of each voxel record the distance from the voxel to a model’s surface, and this data is interpolated by a marching cubes algorithm to generate a visualization of the model. This representation not only allows novel construction methods, but can also generate models with arbitrary shape and topology, something that can be difficult to represent in more traditional representations.

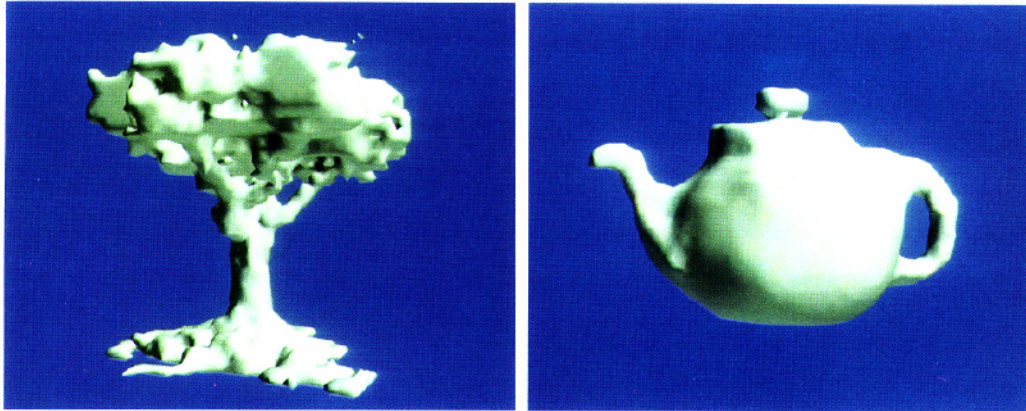
To create and manipulate the models, an artist uses a specially constructed 3-dimensional pointing device to move a tool through the modeling space. The tool is also represented internally as a voxel map, and when activated its voxel

values are combined in a tool function with the voxel values of the model's voxels it overlaps, thus distorting the model's shape. The authors' publication describes tools that can both add and remove material, as well as burnishing tools like sandpaper and a heat gun that can be used to smooth or alter surface properties.

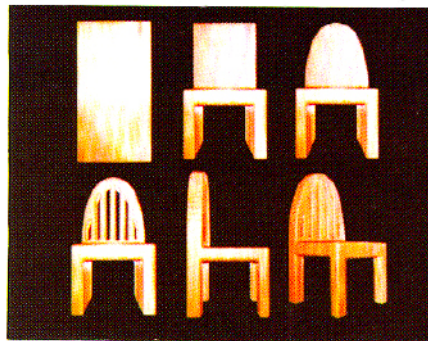
The major limitation of Galyean and Hughes's system was the roughness of the models it was able to produce. This characteristic was both a function of the modeling system itself, and due to limitations in rendering its volumetric representation with the hardware of the time. Galyean and Hughes focused primarily on developing a system that could function at interactive rates, a tradeoff that required models with lower volumetric resolution. Examples of models generated with the researchers' system can be seen in Figure 5.15.

Utilizing more advanced rendering methods on a similar volume raster representation, in 1995 Wang and Kaufman developed a similar system that vastly improved the quality of resulting models [Wang & Kaufman, 1995]. Called VolVis, the system used a more traditional 2-dimensional input from a mouse rather than a 3-D input source, and allowed users to carve or saw material away from a blank surface in order to create rough 3-D models in real time. An example construction using VolVis is depicted in Figure 5.16.

The size and detail of models in both the Galyean and Hughes system and the Wang and Kaufman system are ultimately limited by the voxel representation. In order to store more detailed information about the model's surface, it's necessary to divide the modeling space into higher resolution voxel grids. Unfortunately, doubling the dimensions of the voxel grid results in a cubic increase in the number of voxels, placing a substantial strain on system memory and display algorithms.



**Figure 5.15.** Example models produced by Galyean and Hughes’s volumetric sculpting system [Galyean & Hughes, 1991].



**Figure 5.16.** The construction of a chair from a material blank using VolVis, a volumetric sculpture modeling system developed by Wang and Kaufman [Wang & Kaufman, 1995].

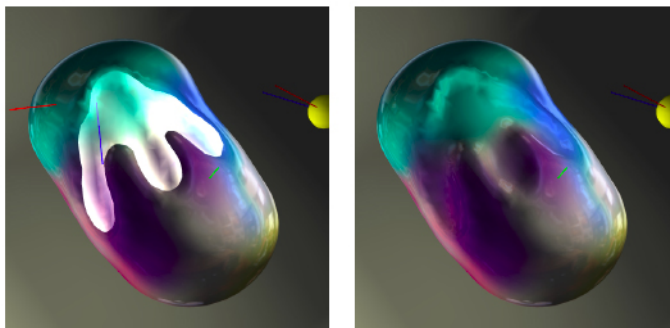
A team of researchers at Mitsubishi’s MERL laboratories proposed a novel solution to this inherent limitation in the voxel representation [Friskin *et al.*, 2000]. Rather than dividing the modeling volume into equally sized voxels, they proposed using an adaptive division method that only divides those regions where additional details are required by the model. This allows other portions of the volume containing simple model components or nothing at all to be represented by larger, coarser voxels that do not require as much memory or processing power. The researchers call their representation ‘adaptively sampled distance fields’ or

ADF.

Based on this ADF representation, Ronald Perry and Sarah Frisken developed a sculpture based modeling application called Kizamu [Perry & Frisken, 2001], named for the Japanese word *kizamu*, ‘to carve’. The modeling interface is built around a chisel tool that performs highly localized CSG style subtraction with a model’s surface to simulate removing material from a block of stone or hunk of wood. Thanks to the ADF representation, Kizamu allows very fine-grained details to be added to the model while still maintaining a reasonable user experience. Developed as a prototype modeler for the entertainment and animation industries, the research team also developed algorithms to convert to and from more traditional modeling representations like triangle meshes so that the application could be included in a production workflow with other industry standard modeling tools.

The sculpting systems discussed thus far have primarily focused on traditional techniques related to stone carving or woodworking. A number of efforts have also been directed at deformation modeling features that bare a closer resemblance to working with clay. An example of such a system is provided by Ferley *et al.* [Ferley *et al.*, 1999]. In this case, the researchers used a fixed voxel grid model representation similar to that of Galyean & Hughes and Wang & Kaufman, but utilizing a balanced binary tree structure to restrict the data to only those voxels in use. Within the modeling environment, users can create globs of material by directing a ‘toothpaste’ tool using a high-dimensional input device. Once a volume is created, other tools can be used to erase portions of the model, smooth out rippled or rough areas, or apply colors to its surface. The most interesting aspect of the system is the user’s ability to generate his or her own tools using the

same modeling interface, and then press or stamp the tool's shape into the surface of another model. An example of this functionality is shown in Figure 5.17.



**Figure 5.17.** An example of Ferley *et al.*'s clay-like deformation modeling system [Ferley *et al.*, 1999]. In this case, the user has created a white foot-shaped tool (left), which was used to make a footprint shaped impression in a model's surface (right).

Adaptively sampled distance fields like those used by Ferley *et al.* provide an effective representational framework to handle models with much higher resolutions, however as the user modifies the model more and more, the underlying mathematical representation stored in each of the adaptive grid cells begins to diverge from its original implicit surface representation. Furthermore, although the user can carve the surface of the model using CSG style operators, methods for more clay-like surface deformations have not yet been demonstrated with ADF's.

In response to both of these issues, Bærentzen and Christensen proposed a volume sculpting system based around the level-set implicit surface representation [Bærentzen & Christensen, 2002]. Tools in this system are represented by localized speed functions that, when directed near the surface of the model, cause its boundary to deform creating model details.

This sort of clay like deformation sculpting has also found favor in a number of commercial modeling applications. Many of the major entertainment and artis-

tic oriented modeling packages now include features to smooth, nudge, or smear portions of their models, usually represented as triangle meshes. One commercial application in particular focuses on this sort of deformation interface. Mudbox [Skymatter, 2007] was developed by a group of professional cinematic 3-D designers as a response to what they saw as the unintuitive modeling interfaces of the industry standard artistic modeling applications.

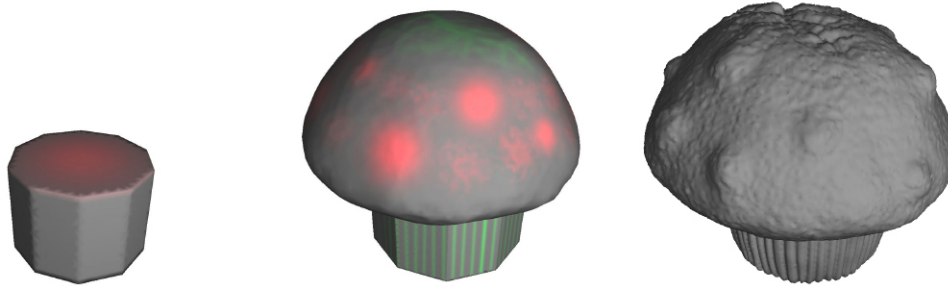
Unlike many of the research projects previously discussed, Mudbox utilizes a more standard representation based on triangle meshes and subdivision. Mudbox models routinely consist of millions of triangles, meaning that more standard point-and-click style modeling is out of the question. Instead Mudbox allows users to create basic shapes like blocks and spheres, and then carve, sculpt, and deform their surfaces using a variety of ‘brush’ tools. The application also provides several innovative features like a mirroring interface that allows the artist to work symmetrically on two sides of a model at once, masking tools to limit the effect of user edits, and a 3-D layer system akin to the 2-dimensional layers in image editors like Photoshop.

Deformation interfaces like Mudbox begin to suggest that perhaps a more intuitive means of interacting with deformation is not to think of physical deformation analogies like carving and sculpture, but instead focus on 2-dimensional interaction methods like painting that are already known to work well in a computational context. Using ‘brushes’ to create a sculpture may not make much sense in the real world, however artists and designers are already well equated with the behavior of technological representations of these tools in other graphic art applications. Thus, more recent research into deformation interfaces have begun to turn in the direction of painting, drawing, and sketch interfaces.



Princeton researchers Lawrence and Funkhouser [Lawrence & Funkhouser, 2003] presented a particularly novel deformation interface along these lines in their 2003 publication. Rather than deforming a model directly, Lawrence and Funkhouser demonstrated a system in which a user paints the surface of a model signifying areas he or she would like to be distorted. Each color of paint signifies a different distortion operation represented mathematically by a paint formula. Paint formulas consume local model information such as the surface normal and mean curvature and produce an instantaneous velocity in the form of a scaled vector. Once the user has applied paint, the model is simulated, allowing each model vertex to move a scaled distance along its velocity vector. Simulation is performed as an interactive animation that the user can observe, stopping the process whenever the model has changed to their liking. By selecting different weights of influence for each component of the paint functions the user can create different effects from sharp, discontinuous embossing to blobby, organic smoothing. An example of this system is presented in Figure 5.18.

Lawrence and Funkhouser point out several advantages their system has over other methods. First this sort of ‘velocity painting’ is conceptually and even intuitively easy to comprehend. Anyone who has squeezed Play Dough through a spaghetti mold can understand how the model will deform. More importantly however, because the changes take place as an interactive animation, the user can easily see what effect their edit is having, and take fine grained control over the extent to which the model will change. This method also has the advantage that alterations to the model are performed as 2-dimensional painting operations, which are much easier for the user to input and comprehend on traditional computer workstations.



**Figure 5.18.** Model of a blueberry muffin sculpted with the velocity painting system, constructed sequentially from left to right. Red paint signifies positive velocities and green paint negative. [Lawrence & Funkhouser, 2003]

Along with paint-like methods, several researchers have investigated sketch-based interfaces that use strokes as their control mechanism. A representative paper in this area, authored by Singh and Fiume, describes a sketch based deformation method called ‘wires’ [Singh & Fiume, 1998]. Inspired by the wire armatures that form the skeletons of stop motion animation figures, wires are parametric curves that a designer can bind to an arbitrary model. Each curve is associated with an implicit function that describes the curve’s influence on each point within the model in proximity to the curve. In effect, the curve becomes a proxy for the model’s features and a handle for manipulation. Once bound, any manipulations to the curves are in turn reflected as deformations of the model. By adjusting parameters of the implicit functions a designer can also control the radius of influence of each wire. This allows wires to be placed over the models surface where they can be used to animate effects like ripples in fabric or changing facial expressions.

Wires provide a means of associating deformations with sketch like input, however the authors’ publication does not describe an interface for manipulating the wires once in place. To see how this sort of deformation can be used in a sketch

based interface, we can look to the example provided by the prototype sketch-based modeling system of Cherlin *et al.* [Cherlin *et al.*, 2005]. Models in this system are represented parametrically as rotational and cross sectional blending surfaces constructed from planar parametric curves.<sup>6</sup> Because the constructive strokes are basically planar, simple models generated by the system also take on a roughly planar configuration. To address this, the authors provide a wire-like deformation system they call ‘orthogonal deformation strokes’. Once a model’s basic shape is defined, the designer creates an additional stroke roughly parallel to the model, but on an orthogonal plane. This stroke then deforms the original model out of its plane like bending a piece of paper—see Figure 5.19. This system allows the designer to bend, fold, curl, and otherwise mold the model’s original shape.



**Figure 5.19.** A planar leaf model from Cherlin *et al.*’s prototype modeling system [Cherlin *et al.*, 2005] is distorted by applying an orthogonal distortion stroke.

In situations like that of Cherlin *et al.*’s system where the model has a simple mathematical representation, the application of this sort of deformation can be straightforward. Characteristics in the deforming stroke can be directly re-

---

<sup>6</sup>The details of Cherlin *et al.*’s construction system will be discussed in more detail in Section 5.9.

flected in changes to the parametric representation of the model. Furthermore, because the model is generated from that representation rather than existing as a static structure, as the model's configuration changes its conversion to hardware compatible primitives can be updated to maintain the same level of visual continuity and smoothness. However for many modeling applications, especially in the commercial realm, 3-D models are represented not as parametric or implicit structures but instead using more traditional polygon and triangle based representations. Whereas volumetric, implicit, and parametric representations can adapt their surfaces to the changes described by deformation operations, polygon representations like triangle meshes directly contain the primitives that are used to display them. If deformation operations fail to consider this structure, alterations to the model can result in aliasing, tears, or the creation of other undesirable degeneracies.

In a 2005 paper, Kho and Garland [Kho & Garland, 2005] present a more fully featured sketch-based deformation system similar to those described by Singh & Fiume and Cherlin *et al.*, designed specifically to address the unique challenges of working with triangle meshes. To use the system, a user first draws a curve that is projected onto a supplied model and becomes bound to the model's geometry much like Singh and Fiume's wires. Then reminiscent of Cherlin's method, the user draws a second deforming stroke alongside the model, deforming the bound stroke and the model along with it. The user can also define a stroke orthogonal to the bound stroke, and then adjust its angle, causing that region of the model to twist in response. This allows an artist or designer, for example, to not only pose the body parts of a figure, but to turn the model's head or twist its arm into the desired position. Examples of these editing features are provided in Figure 5.20.

Coupled with this interface system is a dynamic mesh representation that can automatically adjust to the deformations, maintaining the model's structure. Triangles in areas that are stretched, bent, or twisted are subdivided and smoothed to prevent degeneracies and discontinuities in the model's surface. The result is a system that allows a designer to quickly manipulate the model as if it were made of pliant rubber using an intuitive sketch-based interface.



**Figure 5.20.** (left) The leg of a mesh model of a dinosaur is straightened to follow the path of a red deformation stroke. (right) The model's head and neck are turned to follow manipulations of a deformation of an orthogonal deformation stroke. [Kho & Garland, 2005]

Deformations provide artists and designers with powerful tools to affect the look of their models. However, just as a sculptor begins with a block of stone or a lump of clay, for the most part deformation operations must be applied to an existing model. It is far more difficult to create an entirely new model using these techniques. As these more recent research efforts suggest, deformation can instead be integrated into other modeling techniques, where it functions as one of a number of tools at the artist's disposal. In the case of sketch-based modeling, deformation techniques do not lend themselves to the construction of geometry, however the apparent success of sketch-based deformation interfaces when combined with other tools suggests that deformation can integrate into a

system designed around sketching.

## 5.8 Contour Curves and Drawing Surfaces

In the previous two sections we've examined two different approaches to introducing depth information into the modeling process through a traditional artistic idiom. Height-fields and shape-from-shading borrowed from traditional 2-D shading and painting techniques in an attempt to draw the depth information out of the user's drawings, or allow the user to paint the depth in by hand. On the other hand, deformation techniques were an attempt to apply traditional 3-D techniques of carving and sculpture to 3-D modeling. We can see that both of these methods have met with some limited success, however we have also seen that neither functions well as the sole component of an interface, but are better suited as additional tools added to a standard modeling toolbox.

Among the interface we've encountered thus far, the most successful at tapping into the rough and experimental feeling of sketching as been the blobby inflation methods demonstrated by projects like Teddy. By dealing directly with user's strokes, these systems seem to capture some aspect of sketching that has not translated as well in shading or sculpture. We can see a similar trend among the gesture systems, which are also based around strokes. This pattern suggests that 2-dimensional stroke input may be a more appropriate interface method. However, whereas the shading and deformation idioms directly provide a source of depth information, strokes by themselves are lacking in this regard.

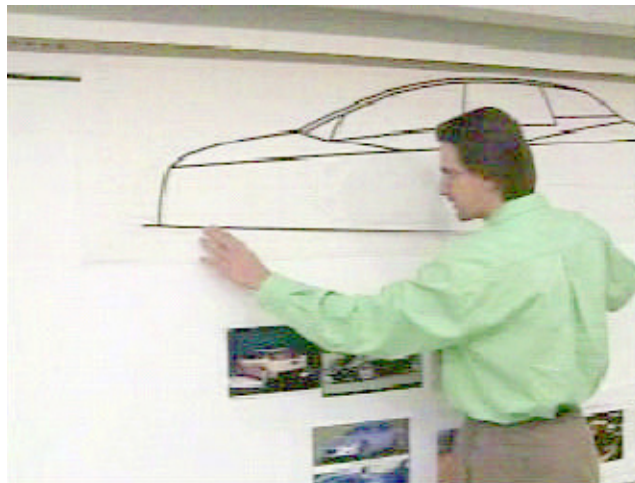
Although efforts like line labeling and artificial intelligence approaches have attempted to extract depth information from raw strokes, outside of very narrow constraints these methods are not robust enough to be used in a general appli-

cation. Projects using inflation techniques have addressed this issue by imposing their own depth information, making the assumption that all strokes are the silhouettes of rounded, puffy 3-D forms. This may be appropriate for entertainment oriented programs, but for a professional modeling application a greater degree of user control is necessary. This means that a successful general modeling system based around stroke input requires a method for the user to indicate this missing depth information more directly. Along these lines, a number of research efforts have directly focused on the creation and arrangement of lines, curves, and strokes within a 3-dimensional environment.

A number of research efforts have examined the use of 3-D input tools and other exotic hardware devices and how they can be used to define 3-dimensional strokes and surfaces directly. A 2001 paper by Schkolne *et al.* describes a system in which a designer can create surfaces by waving his or her specially gloved hand through space [Schkolne *et al.*, 2001], somewhat akin to 3-dimensional finger painting. As interesting as these systems may appear at first, from a practical standpoint, for designers and artists drawing is a 2-dimensional activity. Directly translating the drawing task into three dimensions is such a departure from the traditional physical activity that it's unclear the term drawing should even still apply. In any case, the physical motions and techniques of this new medium are so far removed from the original drawing task that one would expect no more transference of an artist's drawing skills to this new medium than using those drawing skills for clay sculpture. More relevant to the present discussion, some research has examined how curves can be placed in 3-dimensions from a 2-dimensional interface.

Automotive design is one industry in which contour and profile curve modeling is heavily utilized. In order to create a smooth flowing lines, and evocative curves

that give everything from sports cars to family sedans their distinctive appearance, automotive designers often employ a unique one-to-one drawing technique known as tape drawing. Unlike the more familiar pencil and paper sketches, tape drawings are created by literally applying long strips of photographic tape to large vertical work surfaces. As odd as this practice may seem to the outsider, the unique physical characteristics of drawing with strips of tape make it an ideal medium for working on such a large scale. By pulling long strips between the fingers of each hand an artist can easily create arrow straight lines without the aid of a ruler or straight edge. Similarly, to create smooth flowing curves the artist fixes one end of the tape to the work surface and following the attached edge with his or her finger leads the path of the curve by pulling a line of tape taut with the other hand. An example of tape drawing is presented in Figure 5.21.



**Figure 5.21.** An automotive designer creating a tape drawing.  
[Grossman, 2004]

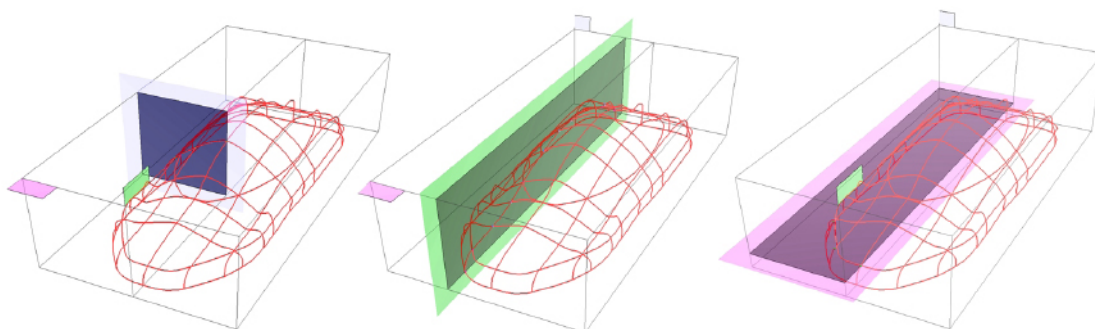
In an attempt to integrate this practice into a digitized workflow, a team of researchers led by Tovi Grossman developed a prototype curve modeling system for the automotive industry based on large displays and these unique tape draw-



ing methods [Grossman *et al.*, 2001]. Physically the system consists of a large projected display, and two 3-dimensional input devices that the user holds in either hand. The application of tape is simulated by positioning the input devices against the projection screen and directing a virtual strip of tape between them. The application's interface presents the user with a modeling stage represented by a rectangular bounding box that is intersected by three construction planes, one for each of the cardinal dimensions of the modeling environment. The user's virtual tape drawing gesticulations create contour curves that are projected onto one of these construction planes. Thus, although the input devices track the 3-dimensional positions of the user hands, all curves within the system are planar. By selecting the appropriate plane in the X, Y, or Z directions, and then moving that plane through the modeling volume to intersect the model in the appropriate location the user is able to place his or her contour curves 3-dimensionally into the environment.

To help the user visualize the model as it's created, Grossman *et al.* also include a number of convenient interface enhancements. The top, side, and front construction planes can be easily accessed or hidden using small tabs that appear on the axes of the drawing stage. The system also supports what its developers call an 'enhanced orthographic view', in which previously created construction lines are dimmed based on their distance from the current construction plane. Curves orthogonal to the current plane are highlighted at their points of intersection with the plane to help the user gauge their relative depth. Users also have full control over their view of the 3-dimensional environment. Not only can the camera's position be changed, but particular viewpoints can be saved with viewpoint markers allowing the user to return to a particularly important or helpful view

at a later time. When new views are selected, the system performs an animated transition from its current position to the new view, providing the user with a sense of continuity and helping to maintain their orientation within the 3-D environment. The modeling environment is depicted in Figure 5.22.



**Figure 5.22.** An example view from Grossman *et al.*'s tape drawing system [Grossman *et al.*, 2001]. Curves in red are created by projecting their shapes onto the movable top, side, and front planes.

A drawback to Grossman's original system was the fact that only planar curves could be created with the tape drawing system. Because curves were only projected onto the top, front, and side planes the system provided no method of dynamically varying the depth component of the curve. In general it is not possible to unambiguously define the 3-dimensional path of a space curve from a single 2-dimensional input, however some research has been directed at interface methods that allow the user to build the path of a space curve by providing additional input.

An influential work in this area was presented by Cohen, Markosian, Zelezeik, and Hughes [Cohen *et al.*, 1999]. Their publication describes a novel method of defining 3-dimensional curves using shadows. The system works as follows. First the user inputs a stroke, which forms a planar curve on a plane roughly parallel to the viewing angle. At the endpoints of the curve, the system draws a set of

guidelines that project a straight path to a ground plane. The user can then draw a second stroke on the ground plane connecting the ends of the two guide lines. The path of the ground curve is interpreted as the shadow cast by the first curve as it hangs in space. Once the shadow curve is defined, the system reinterprets the original curve to correspond to the path of the shadow and the path of the curve from the user's viewpoint. Essentially, a ruled surface is extruded from the shadow curve along the guidelines and the user's original stroke is then projected onto the resulting surface.

The authors claim that this system provides an intuitive interface for artists and designers because of its physical interpretation of an object and its shadow. However, their publication notes that it can be difficult for users to accurately judge the shape of shadows cast by complex curves such as spirals and loops. Furthermore, if the user's shadow deviates too much from the shape of the original curve then undesirable bumps can be created.

Based on the work of Cohen *et al.*, Grossman and his fellow researchers produced a second publication detailing an updated version of their prototype [Grossman *et al.*, 2002]. To allow the user to create truly three-dimensional curves the tape drawing mechanism was again utilized, this time in a two-step construction process. First the user creates a planar curve using the original tape drawing methods described above. This initial curve is then swept through one of the orthogonal planes to create a curving surface. The service then takes the place of the construction planes, allowing subsequent curves to be projected onto its surface. Notice that this system is basically the reverse of the shadow method described by Cohen *et al.* In effect, the shadow curve is defined first, and then subsequent construction curves are projected onto its resulting surface. This method has the

advantage of avoiding the need to reinterpret contour curves in order to adjust their depth. Instead, the depth information is provided unambiguously by the pre-existing surface onto which they are projected. The drawback is that this method prevents the creation of looping curves or spirals from a single input event. However, most of these shapes can still be defined through the construction of multiple curves.

The researchers' second publication also discusses the results of some preliminary user studies. They reported that, in general, the curve creation methods and user interface features went over well with users after brief testing sessions. However, in extended tests, test subjects noted difficulty in planning out where curves should be placed, or how best to construct them within the system in order to achieve the desired contours. Users also had difficulty working with the 3-D input devices due to instabilities in the tracking system, and a lack of haptic feedback usually present in traditional tape drawing.<sup>7</sup>

Grossman *et al.*'s system goes a long way to replicate the working conditions of its target market. However, as a general modeling interface, the large display and exotic input devices make the system impractical for more general applications. In response to this issues, Tsang *et al.* developed a contour curve modeling system with a very similar interface to that of Grossman *et al.*, but utilizing standard PC hardware and a digitizing tablet [Tsang *et al.*, 2004]. Like Grossman, Tsang *et al.*'s prototype modeling application is based around the creation of contour curves. Again modeling is performed on a three dimensional stage represented by a bounding box. Curves are projected onto construction surfaces aligned with the box's principal axes, or onto user defined 3-dimensional surfaces.

---

<sup>7</sup>In his 2004 thesis Grossman was able to addresses some of these tactile feedback issues using a specially designed hardware input device resembling a flexible strip of tape [Grossman, 2004].

To accommodate the transition from 3-dimensional input devices to the 2-dimensional digitizing tablet, the researchers' application also includes a number of tablet specific interface features. Tsang *et al.* developed a one-handed version of Grossman's tape drawing system to allow curves to be created with the digitizing stylus alone. The application also includes an integrated gesture system that allows the user to correct his or her strokes. Gestures are differentiated from stroke construction commands by reading pressure levels from the digitizing tablet; strokes drawn with pressure above a specified threshold are interpreted as gestures by the system, and those below the threshold form new curves.

Tsang *et al.* further extended Grossman's original interface by including a number of assisted drawing features. Users can, for example, apply constraints to curves causing them to snap, glue, or pin portions of their paths to match the features of imported 2-dimensional reference images. The system can also monitor the user's input and suggest, for example, closing an open stroke or re-creating something that was previously drawn.

Together, these projects demonstrate that it is possible to intuitively create and situate curves in three dimensions. They also provide an example of how general 3-dimensional space curves can be defined by the same system. However, indications from the researchers' publications suggest that the creation of completely general 3-dimensional space curves is difficult for the user. Although some of this difficulty can be blamed on deficiencies in the interfaces themselves or the limitations of 2-dimensional display devices, remarks by the authors suggest that to some degree users simply find it taxing to fully consider the 3-D structure of the curves. Gorssman *et al.* for example noted that the majority of modeling time by test subjects using their system was not spent creating curves, but care-

fully considering what curves to create in order to best define the desired model [Grossman *et al.*, 2002].

## 5.9 Stroke-Based Construction from Sweeps and Extrusions

The projects in the previous section prove that a 2-dimensional drawing system can be used successfully and intuitively to create curves in a 3-D environment. However each of the projects was limited to the creation of discrete contour curves. Although this may be appropriate for certain modeling applications such as automotive design, 3-D modeling is primarily concerned with models with a more substantial structure. Our question then becomes, is it possible to intuitively construct surfaces or solid models from 3-dimensionally positioned strokes?

An attempt at answering this question is provided by the parametric modeling system developed by Michalik *et al.* [Michalik *et al.*, 2002]. Models in this system are represented parametrically as b-spline surfaces. In most traditional modeling systems this sort of parametric surface would be edited through cumbersome control point manipulations. Instead, this system provides a sketch-based interface to both the creation and editing of parametric surfaces. To create a model the user draws strokes with a digitizing tablet or mouse into the 3-D environment, which are then projected onto planar drawing surfaces. To aid the user, most drawing surfaces are automatically generated based on the user's viewpoint and the context of their input stroke. The user can create simple swept shapes by defining a profile shape and an axis curve along which to sweep the profile in order to define a surface, however most of the flexibility of the system is derived from its sophisticated constraint-solving system.

The application can interpret a number of adjacent strokes as constraints and

generate a skinning surface that interpolates them. Once a surface is defined, the user can also redefine all or part of its shape using an overdrawing procedure. The user positions a planar drawing surface so that it intersects the model surface forming a contour curve. Once positioned, the user can then overdraw the curve, redefining its path and in turn the shape of the model. The system recalculates the path of the surface to include the user's changes using the same constraint solving methods, gently blending the edit back into the model's surface. By fixing the positions of adjacent curves, or adding additional constraints the user can adjust how much of the model will be affected by each edit.

The primary drawback to Michalik *et al.*'s system is its complexity. As the user provides constraints to the system in the form of strokes, each surface is created by solving systems of linear equations over those constraints. As one might expect, as the number of constraints increases, the difficulty of generating a solution surface also increases. Rather than rigid adherence to each constraint, the system constructs weighted constraint graphs that help to organize and prioritize the constraints impinging on a single solution. Despite these measures, the authors note that as the number of constraints increases, system performance is adversely affected. From an interface perspective, the authors also note that some users had difficulty working effectively in the 3-D environment. Especially when working with the tablet, users lacked effective means of adjusting their view of the modeling environment or other interface parameters.

From the artist or designer's standpoint, Michalik *et al.*'s application demonstrates that 3-D geometry can be effectively defined by directly by the user's positioned 2-dimensional curves. However, from a software standpoint, it would seem that the level of flexibility in the construction methods are perhaps too dif-

difficult for the system to manage effectively. It may at first seem counterintuitive to forgo additional functionality to arrive at a more functional system, but the foundation of sketching from which we are working are not based on the precision or quality of its output, but on the speed and ease with which that output can be created by the artist.

Of the systems we have examined thus far, those based on blobby inflation methods are probably the closest to realizing this dream of a sketch-like interface. Putting differences in interface and representation aside, each of these systems is based on the direct input of silhouette curves by the user. Those curves in turn directly define shape of the model. Even with their plump appearance, because of this direct correspondence, quickly sketched inflated models are a better representation of the user's thought process than could ever be achieved with a traditional point-and-click modeling system. To achieve this interaction, inflation-based systems make a tradeoff, forgoing user control of the 3-dimensional aspect of the geometry in exchange for rapid development from silhouette strokes alone. Although this tradeoff is successful in capturing the whimsical spirit of sketching, the lack of user control makes inflation systems inadequate as general modeling applications, even for preliminary modeling tasks.

Despite the drawbacks, the interface characteristics of inflation systems provide a good starting point from which to work backwards into more expressive modeling interfaces. A good first step in this direction was exemplified by Levet *et al.* [Levet *et al.*, 2006]. Starting from same basic interface and model representation as Igarashi *et al.*'s Teddy, rather than constructing 3-dimensional geometry from a silhouette stroke alone, Levet and his team's prototype modeling application requires two inputs from the user: a silhouette stroke, and a profile



curve. Profile curves replace the standard rounded cross sectional profile found in Teddy, allowing not only rounded shapes, but also more arbitrary cross sections. Profile strokes within the system are always closed shapes, however the user's profile shape is drawn drawn over either a half or quarter circle, and then symmetrically reproduced to form the closed curve. This profile is then used to form the surface of the model along a chordal axis developed from the user's silhouette curve. Levet *et al.*'s system is capable of creating a much wider range of models than Teddy, including some shapes with sharp edges or those containing holes.

Although Levet *et al.*'s system is based on a chordal axis construction method, careful consideration of the construction process, and comparison of sample models provided by the authors suggest that in many cases, very similar results could be achieved using generalized sweeps and extrusions of profile shapes. Examples of procedural modeling techniques, these methods provide a simple and computationally straightforward way to create many common shapes. What's more, a user's drawing input as profile curves and sweep paths can easily and intuitively define them. The versatility of sweeps and extrusions has not gone unnoticed by the sketch-based modeling community. We've already seen examples of other systems that use basic or simplified versions of sweeps to generate geometry. Michalik *et al.*'s system above, as well as many of the gesture-based primitive systems in Section 5.2 utilize similar operations to allow users to quickly and easily generate basic shapes.

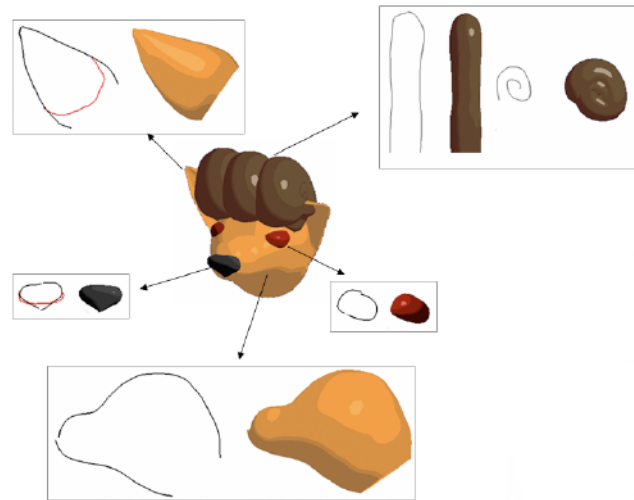
A more general version of the sweep concept was used by Cherlin *et al.* to create a sketch-based modeling system based entirely on this construction mechanism [Cherlin *et al.*, 2005]. Cherlin *et al.*'s publication derives its interface from an observation of techniques used by traditional sketch artist to define 3-dimensional

forms. The authors describe for example a common technique in which the artist draws tight spiraling shapes to help feel out a subject's 3-dimensionality. Translating this practice into modeling techniques, this traditional action is not unlike sweeping a cross sectional shape along a path to define a 3-dimensional form. Based on these observations, Cherlin and his team developed a novel modeling interface that generalizes concepts like ruled surfaces, generalized cylinders, and other forms of sweeps and extrusions as *rotational* and *cross sectional blending surfaces*, which can be defined directly from a user's drawing input.

To create a model, the user begins by drawing a pair of silhouette strokes, each of which is projected onto a planar surface. A rotational blending surface is then constructed such that its axis of rotation follows an average path between the two silhouette curves, and its diameter matches the distance between the silhouette curves. In effect, a circular cross section is swept along the average path of the two curves while simultaneously adjusting its diameter in response to their relative distance. Using this method alone, Cherlin *et al.*'s system is able to create a wide variety of common shapes. For a further degree of control, the user can also draw the cross sectional shape explicitly. The user's cross section then replaces the default circle, creating a cross sectional rather than rotational blending surface.

Once a model component is defined, the authors' system also provides several editing capabilities. We have already discussed Cherlin *et al.*'s orthogonal deformation stroke based deformation method in Section 5.7, which allows the user to deform the curves defining a model component off of their planar alignment, in turn adjusting the model as well. The system also provides a cross sectional oversketch feature, which allows the user to change the cross section at one end

of a blending surface causing its cross section to blend from one shape to another over its length. Once a number of modeling components have been constructed, the user can position them in 3-D space to construct a complete model.



**Figure 5.23.** An example of model construction using Cherlin *et al.*'s rotational and cross sectional blending surfaces [Cherlin *et al.*, 2005]. Note how each component is composed of strokes, and then combined into a finished model.

Based solely on these construction methods Cherlin *et al.*'s modeling system is able to create an astonishing variety of shapes and structures. Because each surface is generated from parametric strokes, the surfaces themselves have a parametric definition. This allows the surface to be evaluated at arbitrary levels of precision, and provides a readymade coordinate system for the application of surface techniques such as textures. From the user's perspective, models are created directly from input strokes, making the construction process fast and highly intuitive. Coupled with the fact that the constructions are derived from traditional sketching techniques, the system appears to provide fertile ground for the transplantation of traditional sketching techniques.

Although the system receives many high marks, there are some specific limitations in Cherlin *et al.*'s design. When constructing individual modeling components, the modeling interface appears to strike a comfortable balance between ease of use and expressivity, however, from the authors' publication it appears each component of a model must be constructed individually and then assembled into a finished design. Although the system provides a means of positioning objects, in describing the construction of example models for their paper the authors' state that placing each modeling component within the scene is difficult, noting that the preponderance of modeling time is not spent creating geometry, but correctly positioning that geometry once constructed. Cherlin *et al.*'s system is also limited to forms that can be defined as cross sectional blending surfaces with closed profiles. This prevents the creation, for example, of thin surface forms like walls, sheets, flags, or ribbons.

Among all of the sketch-based modeling methods we have examined, sweep and extrusion based constructions appear to be the most able to translate traditional sketching techniques into the 3-D modeling arena. In particular, the construction method described by Cherlin *et al.* seems well suited to defining 3-D geometry from drawn 2-D stroke. Because of this, a method very similar to Cherlin's forms the basis for the modeling system described later in this thesis. However, to create an intuitive, effective, and usable sketch-based modeling method requires more than a novel construction method. Cherlin *et al.*'s application provides a perfect example of this fact as well. Although the user can draw model components with ease, constructing those components into a complete design is a hindrance. Thus we must consider the modeling application from a multitude of angles. Can the user draw easily? Can he see what he is drawing? Can she predict the results of

her inputs to the system? What sort of tool—digital or physical—does the system provide to take that input? Each of these is as much a part of the modeling process as drawing a line.

## 5.10 Summary

In examining the modeling and interface successes and failures of these projects we can draw a number of conclusions. First, although the result of the modeling process is a 3-dimensional form, every aspect of the user's interaction with that form is through the lens of 2-dimensional interfaces devices. Mice, tablets, monitors—each of these devices places inherent limits on the user's expressivity and dexterity within the modeling environment. Although interaction methods that mimic 3-dimensional activities like sculpture may seem appropriate, the system must accommodate the user's 2-dimensional experience of the modeling process.

Second, because the practice of sketching spills the user's thought process into a physical (or virtual) form, layers of interface between the user and the modeling environment should be minimized, but also tuned to focus the user's ability. This makes gesture or interaction heavy interfaces inappropriate as sketching interfaces.

Third, no matter the quality or accuracy of an interpretation system, drawing is such an expressive medium that there are likely to always be situations too ambiguous for the computer system to divine on its own. That same expressivity and open nature of sketching means that rather than representing occasional troublesome conditions or the odd edge cases, these ambiguous conditions are perhaps more common than not. In confined circumstances an interpretive system can be appropriate, and even a boon, but at early stages of design it is the ambiguity

of sketching that both confounds discrete interpretation and fuels the creative process.

Finally, although the goal of sketch-based modeling is to translate traditional sketching and drawing skills into the modeling arena, the developer must accept that modeling is an inherently different activity than drawing. Although sketching and drawing may form the basis of a new interface, relying on developed traditional techniques like shading, perspective, or occlusion to provide admittedly related but impoverished spatial information is unsuitable. Rather than literally reproducing existing techniques, it is more effective to extract from those techniques the basic physical activates and mental processes that facilitate them, and then cater to those aspects directly in the modeling system.

# Chapter 6

## Requirements and Design

### 6.1 Aims and Objectives

The primary objective of this research effort is to investigate how the mental and physical aspects of traditional sketching techniques can be applied to the field of 3-dimensional modeling. Furthermore, we wish to examine how the features offered by digitizing tablets could best be utilized in this environment. This entails both the use of the tablet as a means of replicating traditional sketching techniques, and utilizing the dynamic physical information reported by the tablet as a more intuitive interface to 3-dimensional design.

To this end, this thesis focuses on the design and implementation of a prototype sketch-based modeling application dubbed *Hammerspace* (see Note 2). This application provides a test-bed for a sweep and extrusion based modeling method that constructs 3-dimensional models through a sketching interface. The application also implements a number of interaction mechanisms and idioms based around information provided by the digitizing tablet. The goal of this application is to provide a testing platform on which these methods can be both examined

individually, and qualitatively evaluated as a coherent system.

---

**Note 2** Hammerspace

---

**Hammerspace** /'hæməɹ · speɪs/—In the field of cartoon physics, an invisible dimension of space accessible to cartoon characters and occupied by a limitless supply of comically pertinent objects—specifically oversized blunt instruments such as mallets, hammers, barbells, anvils, pianos, safes, cannons, pies, seltzer bottles, and other assorted implements of mayhem. Objects can be transferred from hammerspace into normal space for comic effect. Characters within a cartoon can access this dimension by reaching into bags, cases, or other closed spaces. The existence of an access to hammerspace within these containers explains how objects much larger than would normally fit within them can be produced from their interiors. Characters within cartoons have also been observed to access hammerspace by deftly reaching behind their backs, or out of frame. In some cases, it is also possible for characters themselves to move into and out of hammerspace at will, although this effect can only be performed in the presence of a series of lidded pots, baskets, jars, or behind a series of strategically placed thin vertical structures such as trees or lampposts. [Contributors, 2007b]

---

Although the modeling application described here is intended only as a prototype and testing platform, the process of designing and implementing the system has been modeled after standard software engineering methods. As part of this process, the application was designed around a specific concept and feature-set. As the basis for a possible future software application, the project is targeted at designers and artists, both professional and amateur, who would like to create simple preliminary or experimental 3-dimensional models in a 'sketching' fashion. The models generated by the program are not intended as polished final products. Instead the application should provide a simple and expressive environment in which to explore ideas that can later be refined using more sophisticated conventional modeling tools. This application is not designed as a replacement for traditional 3-dimensional modeling applications, nor is it intended as a replacement for traditional paper-and-pencil or even 2-dimensional computer sketching.



Instead the application offers a 3-dimensional scratch pad or sketchbook to aid in the creative process.

## 6.2 Design Principles

In order to guide the design process, seven design principles were developed. These principles served two purposes. First, the process of explicitly enumerating these principles provided a framework in which to examine which aspects of sketching are most appropriate to translate into the 3-D modeling application environment. Second, the principles serve as a foundation for design and implementation decisions as the project was developed. The following sections enumerate each principle, provide an explanation of that principle's meaning, and offer justification for its importance.

### 6.2.1 Principle I: Sketch is the Basis of Input

This first principle indicates that the methods and operations used to interact with the application should derive from traditional sketching techniques. The hope is that by mimicking the familiar traits of the sketching activity, the application will make it easier for artists, or user of any background, to apply familiar skills. To foster this connection, stroke input to the application should give the user a similar experience to the physical activity of drawing. Furthermore, whenever possible the application design should strive to represent operations as analogues to the corresponding physical sketching activity. This also means that the application's feature set should cater to the spirit of traditional sketching and eschew features that do not directly serve this goal. Finally, even where other methods may be available, the application should prefer doing things in a 'sketching way'.

### 6.2.2 Principle II: Observe a Hierarchy of Sketching Actions

Although we often discuss sketching in terms of a single process, as noted in Chapter 2, sketching can be divided into a number of closely related but distinct activities. These include physically making or adjusting marks, changing the position of the pen or paper, and changing one's view of the work as a whole. In the context of traditional physical sketching this distinction is obscured to some extent by the coordinated and fluid way in which sketching tools, drawing surface, and the artist's hands and eyes are all manipulated together in order to adjust what is drawn, where it's drawn, and what is seen respectively.

In the context of computer based sketching these physical activities are replicated by the application, and so it is necessary to draw more definite distinctions between each of these operations. This allows the application to correctly interpret the user's intentions based on the context of input and the current state of the system. Furthermore, because the system is limited to performing a single operation at a time and mode switching between these operations represents a cost in continuity and user productivity, establishing a hierarchy of these operations will provide valuable guidance when assigning operations to user interface elements.

The primary operation in sketching is stroke creation. Strokes are a fundamental component of any sketch, and their creation is the focus of activity while sketching. This means that the interface distance between the user and drawing must be kept as small as possible. Drawing strokes should be the default operation, and should remain as unencumbered as possible.

Secondary to stroke creation is positioning the drawing medium. Repositioning the drawing is necessary because the range of motion within which the artist's

hand is most dexterous is limited. By repositioning the drawing, the focus of stroke creation can be positioned within this optimum range of motion. Although a modal switch will be required to move from stroke creation to positioning, every effort should be made to make this transition as accessible and fluid as possible.

Tertiary to the above two operations are viewing adjustments. Because traditional sketching is a 2-dimensional medium very little view adjustment is required. The artist's view of the sketch is fixed, and the only major movement is in repositioning one's head or changing one's gaze in order to draw into another area of the drawing—a physical complement to repositioning the drawing medium. However, when considering a 3-D modeling environment, especially one perceived through a 2-D display device, viewing adjustments become an important usability concern. Viewing transitions represent both a coarser version of the positioning operation above, and more importantly a means of understanding the 3-dimensional structure of the model as it is created. Similar to repositioning the drawing medium, it is necessary to make viewing adjustments readily available to the user.

These three fundamental operations represent the *core of sketching activity*. As such, it is important to ensure that the user can swiftly and easily switch between these operations.

### **6.2.3 Principle III: The Goal is 3-D**

By this principle we mean to emphasize that the resulting output of the application should be a 3-D model. In order to achieve this goal, it will be necessary to expand upon the traditional sketching idioms in order to allow the user to indicate to the system the shape of that model. By the same token, it must be clear to the user that creation of geometry, rather than 2-dimensional sketching

or stylistic embellishment, is the primary function of the application. This means that the user interface should focus on helping the user with this creation process, and avoid features or operations that do not directly relate to this goal.

#### **6.2.4 Principle IV: All Lines are Useful**

As we have seen in our discussion of sketching lines in Section 2.3.1.1, although the shape and form of a sketch is defined by the strokes that make it up, not all strokes are used to directly define the drawing. Many strokes serve as a frameworks or guidelines for later strokes, or represent a process or experimentation. This amalgam of lines is characteristic of a sketch, and fundamental to the mental process that underlies the activity. Therefore, the application must provide the user with a means of creating both strokes with an intended geometric meaning, as well as strokes with no explicit geometric interpretation.

#### **6.2.5 Principle V: Prefer Consistency Over Interactivity**

Creating 3-dimensional geometry from 2-dimensional input is an innately ambiguous process. In discussing related work in the field of sketch based modeling, we saw a number of systems that attempt to resolve this ambiguity through an interactive dialogue with the user.<sup>1</sup> While these interface designs provide a means of overloading interpretation of the user's input, allowing the interface to more closely resemble the high degree of dexterity found in pencil-and-paper sketching, this flexibility comes at a price. The process of continually interrupting the user in order to ascertain his or her intentions, or requiring him or her to step

---

<sup>1</sup>These dialogues took the form of so-called suggestive interfaces, expectation lists, calligraphic interfaces, or simply previews. References include: [Igarashi *et al.*, 1997b], [Igarashi & Hughes, 2001], [Cuno *et al.*, 2005], [Eggl *et al.*, 1995], [Tsang *et al.*, 2004], [Shesh & Chen, 2004], [Pereira *et al.*, 2000], [Pereira *et al.*, 2001], [Pereira *et al.*, 2003], [Pereira *et al.*, 2004], [Araújo & Jorge, 2003].

away from the drawing process in order to correct the system is distracting to the mental process of sketching. Furthermore, by overloading a single operation with multiple interpretations, it becomes difficult for the user to predict the outcome of a particular operation.

Although it may come at the cost of greater expressivity, the application should perform one and only one operation for any given input within a specific context. Furthermore, the system should provide the user with limited visual cues about the current context and input so that he or she can reliably predict the outcome of an operation *before* it's activated.

#### **6.2.6 Principle VI: Prefer Tablet Input Whenever Possible**

The tools required for traditional sketching are few and simple: nothing more than a smooth surface and something to make marks with are needed. The peripheral input devices standard to personal computers offer a much wider range of input options and it may be tempting to utilize them. However, especially for the core sketching operations outlined in Principle II above, this tendency should be avoided. The skills fostered by the traditional sketching activity are as much physical as mental, and those fine motor skills used by the sketch artist are closely tied to the use of a drawing implement. As noted in Section 2.4.1, these concerns were the basis for the development of a digitizing tablet in the first place. If we hope to utilize these same skills, it's important to offer the user a familiar physical experience.

### **6.2.7 Principle VII: Sketching Should Be Fun**

Finally, and perhaps most importantly, it is vital that the application provide the same feelings of exploration, experimentation, and enjoyment that are synonymous with traditional sketching. In terms of design, this means that the application should provide the user with operations that do not require special conditions or produce warning messages, but instead perform their action with what they are given. The user should also feel that he or she could safely make alterations or changes and quickly revert them.

## **6.3 Application Requirements**

This section provides an overview of the requirements and specification of the modeling application. In general, the program constitutes a sketch based modeling application that takes primary input from a digitizing tablet to create 3-dimensional models. The following sections enumerate the various requirements of each aspect of the program, and expound upon their required functionality.

### **6.3.1 Tablet Input**

A digitizing tablet will provide the primary input to the system. Interface events produced by the tablet will be used to generate drawing strokes in order to mimic traditional sketching techniques. The digitizing tablet will also take the place of traditional mouse input within the system, and be used as a basis for most other operations, both within the modeling environment, and in manipulation of user interface widgets. In addition to point, click, and drag style information, the digitizing tablet provides several other input channels including proximity information, transducer pen tilt, transducer pen pressure, hovering position, tool

type, and activation of barrel buttons. This additional information should be utilized by the application to provide the user with a means of controlling other aspects of the system.

### **6.3.2 Stroke Input**

The primary interaction between the system and the user should be the creation of strokes. The strokes input by the user are to be used as the basis for construction of 3-dimensional geometry. The application should make clear to the user through the interface, the system's interpretation of each stroke. Along with construction strokes, the user should also have the ability to create strokes that serve no purpose for the system, but provide the user a medium for experimentation or framework construction. These strokes are to be ignored by the system, but should be displayed in the interface in conjunction with the construction strokes. The interface should provide a means of differentiating between strokes that will be interpreted by the system, and strokes that the system will ignore.

### **6.3.3 Environment and Drawing Surfaces**

A common challenge for modeling applications targeted at standard personal computer hardware is providing a reasonable view of the 3-dimensional environment on a 2-dimensional display. To aid the user in comprehending the 2-dimensional display, the system should provide visual cues and environmental elements. Because the strokes produced by the user are inherently 2-dimensional, the environment also will provide the user with an adjustable set of drawing surfaces onto which the strokes can be projected. As the user draws, his or her strokes will fall onto these drawing surfaces, positioning the strokes in 3-dimensional space.

The system must provide a means of creating, adjusting, and visualizing these drawing surfaces, and an intuitive system for placing strokes in their proximity.

#### **6.3.4 View Adjustment**

The system must provide the user with an intuitive means to adjust his or her viewpoint of the 3-dimensional environment. To aid the user in working from multiple points of view, the system should also provide a mechanism where by a given viewpoint can be saved and then recalled at a later date.

#### **6.3.5 2-D to 3-D Conversion**

The most important feature of the application is the 2-D to 3-D conversion system. The conversion system will utilize strokes input by the user in order to create portions of the 3-dimensional model. The user interface to the conversion system should be based on traditional sketching techniques used to define depth and 3-dimensional form in 2-D drawings. However, the technique must be straightforward enough that the application's interpretation of users' input is unambiguous. Once a model component has been constructed from input strokes, the original strokes defining the model should be preserved to provide a visual record of the model's construction.

#### **6.3.6 Modeling System**

The output of the conversion system should be a 3-dimensional model component represented or readily convertible to standard geometric primitives for display or export. The model component must contain sufficient geometric complexity to faithfully represent the user's intentions, but at the same time, the component



should be simple enough to be displayed and manipulated by commodity personal computer hardware at interactive rates. The user should be able to create any number of model components in order to describe a complete model.

The system should be capable of creating model components with a variety of 3-dimensional shapes. This includes visually open and closed shapes, smooth rounded surfaces, and geometry with sharp edges. The construction of more complex shapes will be supported through the creation of adjacent modeling components. Although the system need not provide means of physically combining multiple models into a single structure, the user should be able to create modeling components in any position allowing components to combine through visual adjacency.

### **6.3.7 Graphical User Interface**

The user interface consists of at least the following components. First the user is provided with a main document window. The document window is the visual representation of the current file and displays the current view of the modeling environment. The document window also serves as the primary interface for editing operations. This window is the recipient of drawing strokes, view manipulations, and movement commands for the drawing surfaces.

Operations performed within the document window will be interpreted by the system based on a set of interaction modes. These modes correspond to the core sketching operations described in Section 6.2.2. They include a stroke creation mode, a drawing surface adjustment mode, and a view adjustment mode. The system should visually indicate to the user the current mode, and provide a means of quickly and intuitively switching between interaction modes.

## 6.4 Application Design

### 6.4.1 Target System

The target system for this application is an Apple Macintosh personal computer running the Apple OS X operating system version 10.4. The specific model used for primary development and testing is a Titanium PowerBook equipped with a 500 MHz G4 PowerPC processor, 512 megabytes of system memory, and an ATI RageM3 graphics chipset with 8 megabytes of total video memory. The system is equipped with a Wacom Intuos2 6x8 digitizing tablet. An photograph of the testing platform is provided in Figure 6.1. This particular system was selected primarily because of its availability and familiarity to the author. However, the specifications of this system have implications for the design process and were carefully considered in the early phases of design.

The decision to develop this application in the OS X environment is two fold. First, the Apple Macintosh is widely regarded in industry as a standard platform for media and design applications. Thus the Macintosh is a logical target system for applications in this area. Second, and more importantly, the OS X operating system has been developed around a BSD derived UNIX core. To the average user OS X's UNIX underpinnings are hidden behind a modern graphical user interface. However for application developers, Apple provides a standard suite of UNIX development tools and libraries. Furthermore, Apple's OS X specific API's and development tools can seamlessly interact with the lower-level UNIX components, making it possible to develop the majority of the system in portable and efficient UNIX code. These UNIX underpinnings also provide compatibility with a wide variety of standard UNIX libraries, including the OpenGL graphics



**Figure 6.1.** A photograph of the testing platform, running the modeling application. The system can work with a standard mouse, but is targeted primarily at input from the digitizing tablet.

API, used for processing and display of the 3-D components of the application.

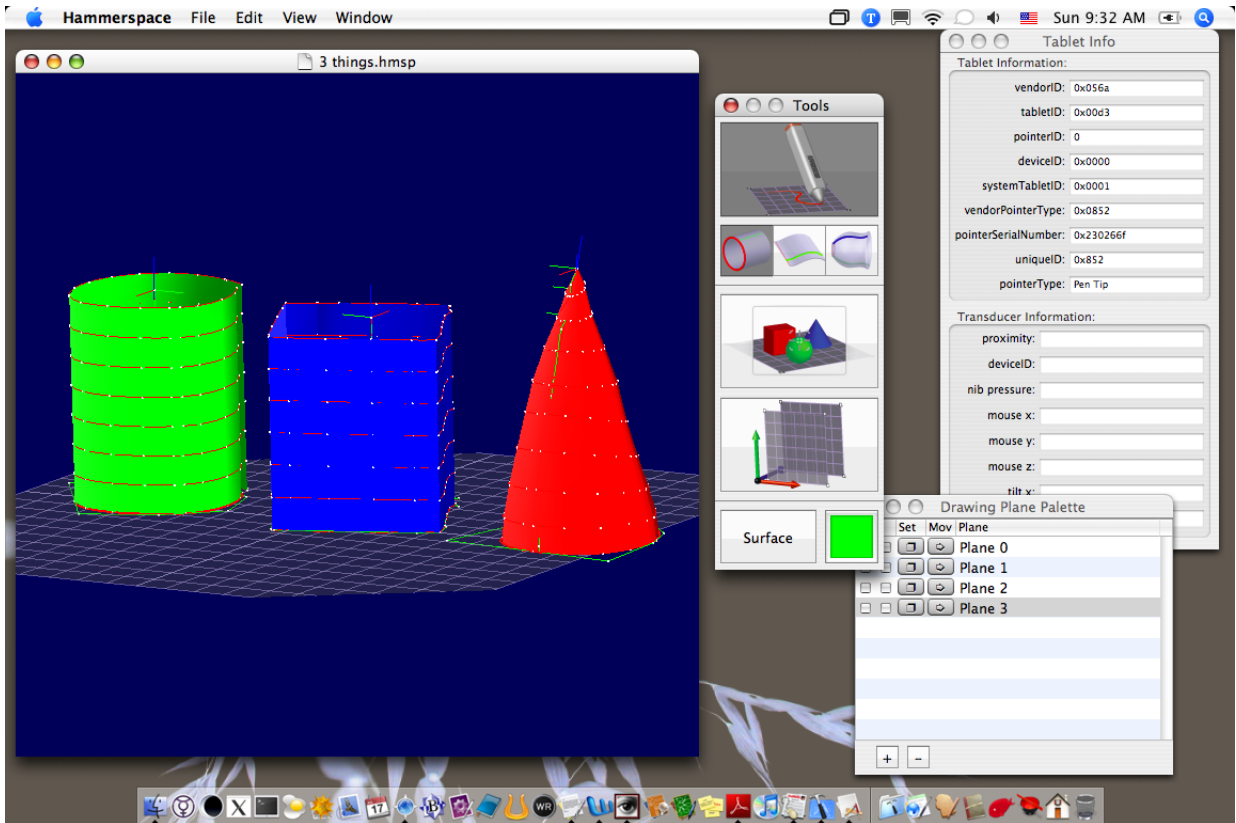
Many readers will note that the hardware specifications of the development system are, by current standards, fairly modest. This is especially true of the graphics hardware, which compared to modern 3-D graphics cards is woefully underpowered, even for a notebook graphics chipset. Although it may seem ill-advised to develop a 3-D modeling application on such an underpowered system, the constraining environment has its advantages. Because of limitations in the CPU and graphics hardware, processing and display bottlenecks within the program are much easier to detect and correct early in the development process. The result is an application with reasonable response characteristics, even on commodity hardware.

### 6.4.2 Graphical User Interface

A view of the entire application and its component and utility windows is depicted in Figure 6.2. As described in the program specifications, the modeling application centers on a main drawing window. This window depicts the user's current view of the modeling environment, and is the recipient of all editing operations from the digitizing tablet, including strokes. A single modeling window in the application represents a single document such that each document receives its own window when opened and any editing that occurs is confined to the file associated with the receiving window. This application format was chosen because of its resemblance to computer graphics applications that should be familiar to the target user group.

In initial design phases efforts were made to construct a system consisting solely of the main document window and devoid of the other panels and controls often seen in graphics applications like Photoshop or Paint. This design was intended to present the user with as simple an interface as possible in order to replicate sketching conditions. The intention was that users should explore the system to discover its features. However, as the interface began to take shape, it became clear that additional controls are needed as indications to the user of the capability of the application and the current state of the system. Again, because of its familiarity to target users, a document and palettes style interface was selected.

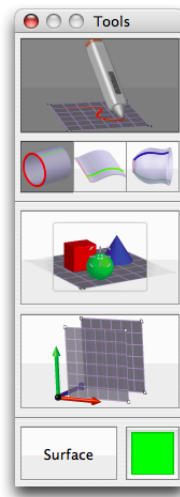
The system includes two palettes: a tool palette and a plane palette, both described in following sections, and pictured in figures 6.3 and 6.4 respectively. These palettes allow the user to explicitly activate operations or change the current



**Figure 6.2.** This screen capture shows all of the windows and palettes available in the application. Moving from left to right you can see the following. The main document window, here displaying a model of three shapes. The tool palette, in this case with the stroke tool and die stroke pen selected. The drawing plane palette, here showing four planes. And finally, the tablet info panel, a debugging panel displaying dynamic information about the digitizing tablet.

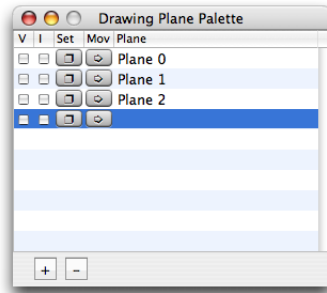
state of the system. Although the system provides these widget-based interfaces, most of the functionality that can be explicitly selected from each of the palettes is also available while drawing with the digitizing tablet via both modifier key combinations, and a number of gestures that utilize tablet dynamics. These tablet gestures are covered in detail in Section 6.4.4. In accordance with design principle VI, these alternative interaction methods are the preferred interaction paradigm for the application, and so initially the toolbars are hidden from view. However,

the user is free to activate the panels at any time from the application menu system. When the alternative methods are in use, the palette systems serve as an indication to the user of the current state of the system, as the appropriate tools and settings will dynamically update as the user calls upon application commands.

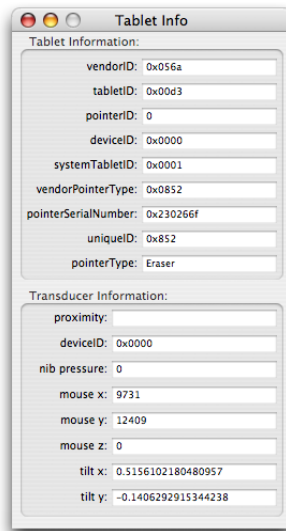


**Figure 6.3.** The application tool palette. The functions are as follows, beginning from the top. (1st row) Stroke Tool. (2nd row) (left) Die Stroke Pen, (middle) Path Stroke Pen, (right) Size Stroke Pen. (3rd row) Camera Tool. (4th row) Plane Tool. (5th row) (left) Activate Surface Construction, (right) Modeling Component Inkwell Widget.

The application functions in three distinct modes, described in the user interface as the *stroke tool*, *camera tool*, and *plane tool*. The primary tool and default operation mode for the system is the stroke tool, which is used to create drawing strokes. The stroke tool is designed to be the most straightforward and unencumbered interaction mode in the system. When stroke mode is active simply drawing on the surface of the tablet while the cursor is over the document window creates new strokes. Associated with the stroke tool are three sub-modes of interaction



**Figure 6.4.** The application plane palette. Each plane is represented by a row in the list. The items in each row are as follows: (1) Visibility Flag, (2) Infinite Plane Flag, (3) Save Camera Angle Button, (4) Restore to Saved Camera Angle Button, (5) Editable Plane Name. The current plane in the list is selected. Here the user is about to enter a new name for this plane.



**Figure 6.5.** The tablet info panel feature is used for debugging input from the tablet. The information it displays is updated in real time as the user's stylus moves in and out of proximity, and as the stylus moves across the tablet's surface. Information in the top division covers static data between proximity events while information on the bottom updates more regularly.










described in the interface as pens: the *die pen*, *path pen*, and *size pen*. These pen modes correspond to the way in which strokes drawn with the stroke tool will be interpreted by the modeling system for 3-dimensional constructions. The stroke tool is always used within one of these three modes, and the user can select the appropriate mode either explicitly through the palette interface, or using either a keyboard shortcut or tablet gesture.

The camera tool is used to adjust the user's current view of the modeling environment. Although the tool can be selected directly from the interface, no matter the current mode the camera tool can be activated from the tablet by drawing under a modifier key, or using tablet gestures. The plane tool is used both to create new drawing surfaces, and to reposition the current drawing surface. Similar to the camera tool, the plane tool can be selected explicitly from the interface, but is also activated by modifier keys or tablet gestures.







To accompany indications of the current interaction mode provided by the tool palette, or to provide these indications when the tool palette is not active, the interface system also uses a set of tool-specific cursors that are activated when the cursor passes over the document window. Each tool is assigned a specific cursor shape, which is then filled or augmented by an additional color to indicate different modes of operation within that tool. For example, the pen tool uses a small cross-hair shaped cursor, which is accompanied by an offset filled circle. The color of the circle indicates which of the three pen sub-modes is currently selected. The cursors for each tool and their respective meanings are depicted in Figure 6.6.

Accompanying the modal tools on the tool palette are two additional controls. A conversion button is provided that activates the 2-D to 3-D conversion function



		Translation	Rotation	Zoom
Camera Tool				
Plane Tool				 

	Die Pen	Path Pen	Size Pen
Stroke Tool	 	 	 

**Figure 6.6.** This table describes the various cursor shapes assigned to different tools in the application. Notice that the shape of each tool is associated with the particular operation it performs, while the color of the tool indicates related operations between tools.

of the application to generate 3-D geometry from the current state of the system. This functionality can also be activated from a keyboard command. Additionally, an inkwell widget is provided that allows the user to interactively select a color for the generated surface. This is the only control on the tool palette that does not have a corresponding alternative input method.

Within the modeling environment, 2-dimensional strokes are projected onto planar drawing surfaces situated at arbitrary positions in the 3-D modeling space. Within the interface of the application these surfaces are referred to as *drawing planes*. In order to aid the user in keeping track of and organizing the planes he or she defines, the plane palette provides a list of all planes defined for the current document. Because drawing planes represent discrete receptacles of drawing input, the drawing palette is designed to resemble the layers used in many 2-D graphics applications. Each listing in the drawing plane palette contains a user-editable name for the plane and two function buttons. The first button is used to associate

the current modeling view with the associated plane. When activated, settings for the camera are saved into the associated plane as a view marker. The second button allows the view marker associated with a plane to be recalled, adjusting the current modeling view to match the saved camera position.

Finally, the application provides a number of features standard to a design application. Models created by the user can be easily saved as document files for later editing or storage. The document file format is a simple text file and contains information to restore 3-D geometry including its position, color, and associated strokes; the location and interpretation of strokes, the locations and orientations of drawing planes, and which strokes belong to them; any saved viewpoint markers associated with those strokes; the visibility and extent of each drawing plane; and other user preference settings. Although the file format is unique to the system, the geometric information it contains describes standard geometric primitives. Straightforward alterations to the output system could be made to support the generation of file formats compatible with standard 3-D modeling tools or utilities. This proprietary format was selected for ease of development and debugging.

The application is also equipped with a fully featured undo/redo system, including a reversion history all the way back to the last save point. This feature is important in a graphics application, and especially so for one catering to a sketching mindset. The undo feature allows users the freedom to play and experiment with the system without the fear of making mistakes or irrevocable changes. After an undo operation, not only are changes to the model reverted, but features of the state of the system such as the designated interpretations of each stroke are also rolled back, meaning that the user can pick up exactly where he or she left off.

### 6.4.3 Tablet Input

As noted in Section 6.4.1, the digitizing tablet device target for this application is the Wacom Intuos2, an image of which is pictured in Figure 6.7. The digitizing tablet is connected to the test system through a universal serial bus, and is managed by driver software supplied by the manufacture. The device consists of two components. The first is the tablet itself, which is directly connected to the system. The tablet is a flat rigid housing resembling an undersized desk blotter. A 6" by 8" region in the center of the device is the active area of the tablet, and is covered by a thin plastic drawing surface. The tablet also features a small indicator lamp along its top edge. The lamp displays an amber color while the tablet is plugged in, but changes to green when the tablet detects clicks and drags from the user. This tablet model also features a region along the top most edge of the active area that functions as a set of user assigned buttons to access common commands such as save, undo, and exit, however these features were unused by this application.

The second component of the tablet device is the digitizing pen or *stylus*, referred to in Wacom technical documents as a *transducer*. The particular transducer used in this application resembles a standard pen with an ergonomic rubberized grip. The marking end of the stylus contains a small plastic nib, and the opposite end contains a larger eraser-shaped plastic nib, both of which function as input methods for the tablet. Along the barrel of the stylus at the grip location there is a 2-function saddle shaped barrel button that can be activated by the thumb while holding the pen. A close up image of two Wacom styli are presented in Figure 6.8. A special mouse is also provided that can work with the tablet



**Figure 6.7.** Photograph of the Wacom Intuos2 digitizing tablet used for testing and development of the present application. Although the mouse device is included with the tablet, its features were not used with the application.

technology, but was not used by this application.

Although not a feature of the particular model tablet used for testing and development, the current line of Wacom Intuos3 tablets, which have replaced the Intuos2, also feature a cluster of modifier proxy buttons referred to in Wacom literature as *ExpressKeys*. The ExpressKeys are chordable modifier buttons that can be assigned to program functions or standard modifier keys on the keyboard. Placing the keys directly on the tablet allows the user easy access to modifier keys that might be struck or held while drawing operations are taking place. Although the testing tablet does not feature these buttons, their use as modifier proxies is



**Figure 6.8.** Photograph of two different models of stylus style transducers for use with a Wacom digitizing tablet, along with a standard No. 2 pencil for size comparison. The purple stylus in the middle belongs to the Intuos2 tablet used for testing while the bottom stylus came with an older tablet model. Both feature a plastic drawing nib at their left extreme, and a larger plastic eraser nib at the right. Along the grips of both pens is a saddle button. The button rocks forward or backward, providing two functions. Notice how the ergonomics of the pen have been updated to provide a thicker, rubberized grip and more pronounced barrel button.

easily simulated by pressing the same keys on the keyboard, and so the application has been written to make heavy use of modifier keys to access input modes. The newer line of tablets also features a variable pressure strip, which can function similar to a mouse scroll wheel. However, because this functionality is not as handily replicated, these features are not utilized in the current design.

The tablet is activated when the transducer device moves within approximately  $\frac{3}{4}$  of an inch (about 2 cm) from the tablet's surface. At this point the tablet driver generates a proximity event indicating the position of the transducer, identifying

characteristics of the transducer such as serial number and tool type, and in the case of the stylus, information about which end of the stylus has entered proximity, pen tip or eraser. Once the transducer is within proximity, movements of the stylus are interpreted by the operating system just like mouse movements. Sweeping the pen tip above the tablet's surface moves the on-screen cursor as if the mouse had been dragged, touching the pen or eraser tip to the tablet surface generates a mouse down event, dragging the tip a drag event, and lifting the pen a mouse up event. After the pen is lifted, if its tip moves further than the proximity distance from the surface of the tablet then a proximity exit event is triggered, corresponding to the proximity entry event described above.

Although both a standard mouse and the tablet can be connected to the system at one time, only one cursor is available to interpret their input, so simultaneous use is not currently possible. Although tablet input is interpreted by the system as mouse events, the characteristics of using the tablet as a mouse are slightly different as described in Section 2.4.1. Briefly, whereas movement information collected from a standard mouse entails only a displacement from the current cursor position, information generated by the tablet provides an absolute position for the cursor on screen. This has the effect of mapping the screen area onto the active area of the tablet so that when the user places the pen down in the upper left-hand corner, lifts the pen away from the tablet, and then proceeds to make a mark in the lower right, the cursor follows the same movement on screen. To the user this mimics the feeling of drawing at different positions on a piece of paper.

Along with standard position information provided to applications by the system, tablet events include three additional parameters. The first indicates which end of the transducer stylus is in proximity to the tablet surface. This value is

static between proximity entry and exit events. The second is a set of parameters identifying the angle of the pen in relation to the x and y-axes of the tablet surface. This information is generated dynamically whenever the pen is within proximity, and is provided to applications by the system as normalized x and y tilt values in the range of -1.0 to 1.0. This range corresponds to the full 180° along each axis, however the tablet is in reality only capable of detecting the tilt within about 60° above or below zero along either axis. The final parameter supplied by the system is pressure information from the pen tip. Pressure information is only generated while the pen tip is in contact with the tablet surface. The tablet is capable of detecting 1,024 levels of pressure from both the pen and eraser tip. The user can adjust the sensitivity of the pressure system in the tablet's driver settings. Pressure levels are normalized by the operating system and provided to the application in a range of 0.0 for no pressure to 1.0 for full pressure.

Under normal circumstances tablet events are wrapped by the operating system into standard mouse events that can then be processed by applications with standard event callbacks. Applications that wish to access additional tablet features can then extract this information from the system at the time of the callback. In most cases this arrangement is beneficial, allowing ordinary applications to interact with input from the tablet without the need to be specifically written to take advantage of its features. However, for drawing applications that wish to take full advantage of the tablet, the mousing infrastructure of the operating system is limiting. The tablet hardware is capable of generating 100 events every second. This level of granularity is necessary in order to preserve the smoothness of strokes drawn very quickly by the user. Although the tablet hardware is generating events at such a high rate, the operating system coalesces multiple mouse events that

occur during a single event-processing loop into a single event, reducing the sampling rate of the tablet substantially. In order to receive the full granularity of tablet events, the operating system's coalescing facility must be disabled. This provides the application with access to every event generated while the stylus tip is in contact with the tablet surface.

#### **6.4.4 Tablet Gestures**

As discussed in Section 6.4.2, the default interaction mode for the system is the stroke tool, meaning that in the absence of explicit mode changes by the user, basic drawing input from the tablet is interpreted as stroke creation. In accordance with design principle I, which stresses the use of input methods analogous to physical sketching activities, and principle VI, which advocates a preference for tablet input, a system of additional tablet input methods were also developed. The development process involved a careful consideration of each tablet feature, its properties as an input parameter, and possible methods of utilizing the feature as an interaction mechanism. A summary of this examination process is depicted in Tablets A.1, A.2, A.3, and A.4 in Appendix A.

The result of this process was a set of tablet gestures that utilize the additional pressure, hover, and tilt information provided by the tablet to allow the user access to the camera and plane tools without the need to explicitly activate them from the tool palette. In order that each gesture relate to a physical analogue, each of the tablet gestures was developed to mimic a physical motion or technique observed during traditional sketching. Five tablet gestures were developed and are outlined in the sections below. Three of these five were ultimately added to the application.



#### 6.4.4.1 Brush-Off / Rehearsal

A common physical motion observed while sketching is a simple brushing or waving motion. To perform this action the artist lifts the drawing utensil from the drawing surface and then gently brushes or waves the heel and back side of the dominate hand above or against the paper, often in large sweeping arcs. This action is often performed after erasure in order to clear eraser dust from the vicinity of the correction. In other cases this action might be used to adjust the position of the drawing paper under the hand. A second but related rehearsal motion is also a common action among traditional artists. Here, the artist holds the drawing utensil in a standard grip and performs several quick passes just above the surface of the paper before committing to laying down a new stroke. This technique is especially popular with character artists and animators who place a certain value on the personality of their strokes. By taking a number of “practice swings”, they can generate the desired line with a single drawing action rather than building the line from an amalgam of overdrawing strokes.

The digitizing tablet offers a means of capturing these motions and utilizing them as input to the system. As noted above, the tablet has the ability to detect a transducer device, even when that device is not in direct contact with the tablet’s surface. When a transducer like the stylus comes within close proximity to the tablet surface, interested applications are alerted to its presence and subsequent movement. Within this state the application can receive information about the position and tilt of the stylus, as well as which end of the stylus (pen tip or eraser) is in use. We call gestures used to activate commands or enter data in this manner *brush-off* or *rehearsal* gestures. In the present application a version of the brush-off gesture called *lift-and-lead* is used as one method of controlling

the viewing system. See Section 6.4.10 on page 308 for a more detail description of lift-and-lead.

#### 6.4.4.2 Pounce

An intriguing capability of the digitizing tablet when used in conjunction with a stylus transducer is the availability of pressure information. As noted in Section 2.4.1, the pressure information provided by the tablet is usually used by graphics applications as a means of dynamically adjusting parameters of a drawing tool such as brush shape or paint opacity. In the context of the current application these features do not have an obvious application. However, pressure information can also be used in more abstract applications to adjust other program parameters or to affect an action [Ramos *et al.*, 2004].

A particularly interesting example is provided by Tsang *et al.*, who used pressure to differentiate drawing input from gestural command input in their prototype modeling application [Tsang *et al.*, 2004]. Strokes applied with pressure over a threshold were interpreted as gestures whereas softer strokes were used for drawing input. In their paper the authors note that this binary pressure mechanism was selected because users had difficulty controlling multiple pressure levels while drawing. Even when limited to this binary state it was necessary to adjust the pressure threshold for each user. Despite the obstacles encountered by Tsang *et al.*, their novel use of pressure as an interface component has a certain intuitive feeling, and deserves further attention.

As an attempt to improve on their original designs the *pounce* tablet gesture was developed. A pounce gesture is a quick, isolated, high-pressure event used to signal a modal transition such as a tool change or context switch within an

application. The user creates a pounce by placing the stylus on the tablet's surface and generating a brief pulse of pressure, similar to the motion one might use to punch a calculator key with the end of a pencil. After the pulse, the pressure is quickly released, but the stylus tip can remain on the surface to provide further input once the pounce is recognized and the context changed.

The pounce gesture is controlled by four parameters. The first, as in Tsang *et al.*'s formulation, is a pounce pressure threshold. This is the pressure value the user must exceed to activate a pounce. Second is the pounce duration, a range defining the minimum and maximum number of sample events from the time the user's input first crosses the pressure threshold until the pressure falls below the threshold again. This parameter is used to screen out false positives by ensuring that the high-pressure event is the result of a short isolated burst of pressure as opposed to a longer high-pressure stroke or contemplative pause. The third parameter, called the pounce distance serves a similar purpose to screen out false positives. This parameter describes a maximum distance between the first sample exceeding the pressure threshold and the first sample to fall back below that pressure value. The intention here is to screen out high-pressure events that may have been generated by the user while forming a particularly enthusiastic drawing stroke. A final optional parameter is a startup sample range. This range indicates a maximum number of samples between the time of pen down, and the first sample above the pressure threshold. This parameter allows the system to limit pounce events to the first second or two after the stylus touches the surface. If the pounce does not occur within the specified number of samples, even if it meets all other criteria, the pounce will not be activated and the operation will be ignored.

The pounce gesture has a number of advantages over the simple binary pressure switch described by Tsang and his team. First, because the pressure sample is isolated, there is no need for the user to maintain a pressure level over a long period of time. This means that once the mode switch is done, it's done. There is no concern that an operation will be accidentally canceled or disrupted part way through. Second, because the pressure action associated with the pounce event is fairly blunt, and is only maintained for a fraction of a second, there is less need to tune the pressure threshold for each individual user. Although users may need to find a comfortable threshold when asked to work at a pressure level for an extended period of time, everyone is capable of generating a single relatively high-pressure event no matter their usual working pressure comfort level. Additionally, because the pounce does not rely on pressure information over a longer range, and can even be isolated to the beginning of a stroke, the system is open to interpret variable pressure levels while drawing for other purposes, even including their standard use as dynamic adjustment to tool parameters.

The development of the pounce gesture has also revealed several issues that must be considered by a pounce implementation. First, just as Ramos *et al.* pointed out when describing other uses of pressure gestures [Ramos *et al.*, 2004], it's necessary to provide the user with some sort of visual feedback to indicate that the pounce was successful. This feedback is provided in the current system by the visual change of cursor shape and tool palette selection. Second, the application must be prepared for the fact that the user may fail to properly activate the pounce for whatever reason. Because the pounce operation requires a pen down action, which might otherwise be interpreted as editing input, the application should provide an undo or clear mechanism in the interface as a mean of removing or

correcting a pounce that is incorrectly interpreted.

The current application uses the pounce gesture as a means to activate portions of the drawing plane manipulation system. For further information see Section 6.4.9.

#### **6.4.4.3 Joystick**

The next tablet gesture does not correspond to any particular sketching-related physical activity or motion, but is derived from the capabilities of the tablet alone. When the stylus transducer is used in conjunction with the tablet, tilt information is provided continuously as long as the stylus is within range, even when the pen tip or eraser is not in motion. This feature provides an opportunity to utilize manipulations of the stylus's orientation over the tablet surface or while one end is planted and stationary.

One interpretation that is easily equated with a familiar input system is to use the stylus similar to a *joystick*. In this gesture, the stylus is grasped in an upright position with one end placed against the tablet's surface. The end of the stylus is kept relatively stable while the angle of the barrel is manipulated in a manner reminiscent of an aircraft joystick. The barrel can be grasped with the entire hand; similar to the way a child might hold a crayon. This grip provides greater stability, but limits the user's range of motion, especially in the positive x direction, and to low angles in all directions. This grip is most appropriate when used in conjunction with the eraser end of the stylus down to avoid accidental activation of the barrel buttons. Alternatively, the stylus can be held in a looser grip, lightly clenched between the thumb and first few fingers, and stabilized by resting the heel of the hand against the tablet. This grip provides much wider

range of motion, and when used with the pen tip, easy access to the barrel buttons.

Early testing with this interface suggests several implementation considerations. First, the protective plastic surface of the tablet is very slick, designed to let the pen tip slide easily over its surface. Because of this it's very difficult to accurately keep the end of the stylus firmly planted and prevent any motion of the tip. Consequently any implementation must be prepared to ignore this positional input, or be otherwise un-phased by its appearance. Second, although this or similar gestures can be implemented to read the tablet angle while the stylus is not in contact with the tablet's surface, proximity tilt information provided by the tablet driver while the pen is hovering has a lower fidelity than tilt information generated while the pen or eraser tip is down. For zero-pressure mouse events, (i.e. those generated while hovering or for events not associated with the left mouse button), the tablet driver coalesces the events when the tip movement is less than one pixel. This coalescence is separate and in addition to the event loop mouse event coalescing performed by the operating system and described in Section 6.4.3. Furthermore, orientation information in general becomes less accurate the further the transducer is from the tablet's surface. In examining the information provided by hovering tablet events, it appears that one or both of these factors can cause inaccuracy and instability in the angle readings. For example, as the pen is slowly rotated, angle values may fail to update in successive events, or the angle reading may momentarily jump several degrees backwards or forwards and continue updating from that position.

A simple implementation of this gesture was developed in the early stages of the project. However, because the related motion does not correspond to a traditional sketching analogue, it was decided that other methods were more in

keeping with the design principles of the application. From the limited testing it appears that the joystick gesture would best be suited for positional or rotational controls of modeling elements or the view as a whole. Because of their less accurate nature, angle related gestures tracked while the stylus is hovering are probably better suited for coarse-grained controls that only depend on the general level of tilt rather than its exact value. Examples of these sorts of gestures are provided by the flick and low-angle push gestures described in the next sections.

#### **6.4.4.4 Flick**

The *flick* tablet gesture, as its name implies, is designed to resemble a ‘flicking’ or ‘flip-through’ motion that one might make while counting off items in ones head, or mentally running through a series of options. The flick gesture is performed as follows. The user holds the tablet stylus in a standard pen grip in the dominant hand. The hand rests against the surface of the tablet or just above so that the pen’s tip is within proximity but does not contact the tablet’s surface. While held in this position, the user makes a brief flick motion, pivoting the pen at its grip once back and forth, forming at the tip a short vertical line in the air. This operation can be performed in two directions, either down (in the tablet’s negative y direction) towards the user and then back to the rest position, or up (in the tablet’s positive y direction) away from the user and then back. The motion is performed as a single swift action, exerting force to drive the pen away from the rest position and then allowing the natural elasticity of the hand to spring the pen back.

From an implementation standpoint, the motion of the flick gesture can be recognized in two different ways. The initial flick system developed for the present

application used tilt information from hover events generated by the stylus. Although this method was partially effective, instability in the tilt information from proximity events as described in the previous section proved to be a hindrance to correct recognition of the gesture. Especially disruptive were instances in which the angular reading would jump backwards or forwards, causing the gesture to register multiple ‘flicks’ when only one motion was performed by the user. An alternative method, and the one settled on for the flick system in the final version of the application, is to use instead the positional information generated by the end of the pen. Using either method, the implementation is straightforward.

When the flick system is activated it’s assumed that the user’s hand is in the rest position. The initial orientation or position of the pen is recorded, and will be used to compare with later readings to determine if a flick has occurred. The system also maintains two parameters, a flick-up threshold, and a flick-down threshold. These two values indicate the minimum distance (either angular about the x-axis, or Cartesian along the y-axis) through which the stylus must travel in order to activate the operation associated with the flick. The flick operation is triggered whenever the stylus moves outside the rest range defined by the threshold parameters, and the amount of the reading in either direction determines if the up- or down-flick was triggered. Note that individual up and down thresholds are used rather than a single parameter. This is because the flick distance in the vertical (positive y) direction is more limited by the dexterity of the hand than a downward deflection. By using two parameters the system can more accurately register upward flicks over a short distance while avoiding false positives in the downward direction.

Although it’s possible to track the movement of the stylus at all times, in order



to keep the recognition of the flick gesture as simple as possible and avoid confusing flicks with other movements of the stylus, its best to equip the flick gesture with some sort of activation. One viable option is to activate the flick detection system when a particular keyboard modifier key is depressed, and to deactivate the system once again when released. This method is quick and simple, but requires focus on both the keyboard and the tablet. A cleaner solution, and the method used in the current implementation, is to assign flick activation to one of the stylus barrel buttons. The upper barrel button in particular is situated at approximately the same location as the pivot when performing a flick, and is placed such that the thumb can easily depress it using the same force needed to comfortably grasp the stylus.

The flick tablet gesture has a resemblance to mentally flipping through choices or checking off items, and so the operation provides an ideal interface to cycling through a short list of mutually exclusive options, such as different versions of a tool. An added advantage is that the up and down variations of the flick can be easily detected and differentiated, allowing the gesture to move through the list of items in either direction. Because the flick relies on angular or position information on the vertical axis, its implementation is also ambidextrous. The current application utilizes the flick gesture to cycle through sub-modes of the stroke pen tool.

#### **6.4.4.5 Low-Angle Push**

The *low-angle push* gesture is similar to the brush-off gesture, but is designed to replicate the physical motion of sweeping the drawing surface with the front side of the hand. Someone might make this physical movement when skimming

over the words on a page of text while holding a pen. The hand is flattened against the drawing surface with fingers outstretched. The pen is held in a loose grip, with its point nestled between the index and middle fingers. The side of the tapered region around the pen's point rests against the drawing surface, but the tip of the pen itself does not contact the surface. Instead, the pen's tip along with the tip of the middle finger serve as visual guides while skimming over the page.

Translated as a tablet gesture, the low angle push can take advantage of the tablet's tilt functions for recognition. As was discussed in Section 6.4.3 above, the tablet provides tilt information in both the x and y directions over a 180° range normalized to fit between -1.0 and 1.0. However, in actuality tilt information from the tablet is only reported up to about 60° of deflection in either direction along the axis. This design is perfectly logical because when the stylus tip or eraser is in contact with the tablet's surface it's basically impossible to exceed this angle and continue to both grip the stylus and remain in contact with the surface. On the other hand, when hovering events generated by the tablet are tracked, it is possible when making motions like those described above to exceed the 60° angle of deflection. In these situations the tablet reports a capped value that does not increase, even as the pen's angle changes. This capped value provides a convenient threshold for detection of low-angle sweep input. From an implementation standpoint it may be necessary to institute threshold parameters, which could be adjusted to allow cutoffs at a lower angle than 60° degrees. Still, the hardware threshold represents a hard ceiling for this parameter.

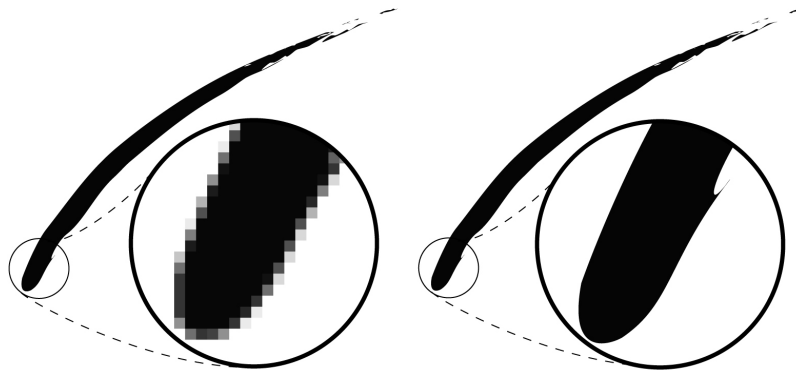
Like the flick gesture, because the low-angle push does not involve contact between the stylus tip and the tablet surface, some activation mechanism is required

to signal the application that the low-angle push gesture recognition should be detected. This can be achieved using one of the barrel buttons, but because of the position of the stylus in the user's hand, a keyboard modifier or application input mode might be more appropriate. Finally, the low angle push gesture is more easily performed using the eraser end of the stylus rather than the pen tip. The gesture requires sliding the hand and stylus against the tablet's surface, a motion that is hampered by the rubberized grip of the pen end of the stylus. From the eraser end, contact with the surface is limited to the smooth plastic barrel of the stylus, which slides much more easily. Because of the shape of the eraser end, it might also be possible to use this gesture with the eraser tip in contact with the tablet surface, avoiding the need for an activation or modifier key.

The physical motion of the low-angle push gesture suggests that it could be used most effectively as either a rough pointing interface, or for translation or other movement commands. The low-angle sweep was developed in response to the particular physical skimming motion described above, but translated into a tablet gesture, it's possible to consider holding the stylus in other low angle positions, for example with the opposite end of the pen pointing to either side, or away from the user. Each of these suggests a physical motion of edging or scraping something along the tablet's surface. In this capacity the low-angle push gesture was developed for the current application as a means of adjusting the visible size of drawing surface. The intent was to allow the user to perform a low angle push at the boundary of a drawing plane, extending or reducing a dimension of the plane depending on the direction of the stylus deflection. However, this feature has not been fully developed as of the time of writing, and is not included in the prototype application at this time.

### 6.4.5 Stroke Format

As the primary input to the application, the capture and manipulation of strokes is arguably the most important aspect of the system. Design principles I and II clearly indicate that the stroke system must replicate the fluid feeling of traditional drawing. Established computer graphics techniques provide two alternate representations for graphical data. The first method referred to as *raster graphics* represent graphical data in terms of the color and location of a fixed number of pixels—see Figure 6.9. Many standard 2-dimensional computer graphics applications such as Adobe’s Photoshop [Adobe, 2007b] use raster graphics as their representation because pixel level information is the format available for most image data. Working at the pixel level makes possible the creation of elaborate and expressive input tools such as the brush system in applications like Corel’s Painter [Corel, 2006].



**Figure 6.9.** (left) Raster graphics represent the visual image in terms of the activation and deactivation of individual pictures. This representation is most evident when the graphic is enlarged to the point that individual pixels become evident. (right) Vector graphics on the other hand represent the image using a mathematical representation that can be dynamically rendered to maintain sharp features at any magnification.

Raster graphics are well suited to these sorts of 2-D graphics applications, however for a 3-D modeling application raster representations have several drawbacks. First, raster graphic representations are directly tied to the pixels that define them, and so raster images are resolution dependent. This dependency is a limitation for a 3-D modeling environment in which the user will want to view the model from a variety of angles and positions both close up and from a distance. Second, strokes defined as sets of pixels do not have a definite geometric interpretation. It is difficult, for example, to say whether a particular point in space is on or off of a given raster stroke, or precisely where two raster strokes intersect. Even determining where one stroke begins and another ends becomes more and more difficult as strokes begin to overlap and pile up one on top of another. Because the modeling application will use strokes as the basis of 3-D geometry, we require a stroke representation that includes this definite geometric information.

Finally, in general it may seem preferable to choose a stroke representation that has the capability to simulate natural media. It is true that one of the stated design goals of the application is to strive for a sketch like user experience. However, design principle III clearly states that the application's primary goal is 3-D construction. Offering the user an interface that resembles traditional sketch too much will encourage the user to produce 2-dimensional sketches rather than 3-D models. By providing the user with an interface that allows sketch-like stroke input, but does not support sketching activities that are counterproductive to 3-D construction, the limitation of the interface will guide the user to utilizing the features that are available.

The alternative representation to raster graphics is referred to as *vector graphics*, a representation we have already mentioned in passing on several occasions.

Vector graphic images are represented in purely geometric terms as collections of graphic primitives and operations on those primitives, rather than as collections of pixels. This representation means that each time a vector graphic primitive is to be displayed, its visual representation in pixels must be generated. Although any particular display of a vector graphic image has a fixed resolution, because the image is generated each time for the specific display, the vector graphic representation itself is resolution independent. An added benefit of this representation is that all of the geometric information for each primitive is directly available to the system. This format is more compatible with the graphic primitive elements provided by the 3-D graphics sub system, and also provides a rich source of data to which algorithms can be applied. For these reasons, a vector graphic based representation was selected to represent stroke information.

#### **6.4.6 Drawing Feedback and Stroke Conversion**

As the user draws on the digitizing tablet, the tablet generates raw sample points that are passed directly to the application. These points are collected into a simple data structure and can be visualized by connecting the points in sequence using simple straight line segments. This representation is known as a *polyline*. Although the polyline contains the basic necessary geometric information describing the path of the user's stroke, the representation is inefficient. Some areas of the stroke that were drawn slowly are over specified by the large number of sample points, while others drawn more quickly have too few, producing a jagged line that does not approximate the path of the user's pen. To more accurately and efficiently represent the user's intended stroke, the raw sample point data from the polyline is subjected to a three-stage process of *filtration*, *emphclassification*, and

*conversion* into a parametric curve representation. This process is described in detail in Section 7.2 beginning on page 344. The end result of this fitting process is a new representation of the stroke as a chain of cubic Bézier curve segments. The stroke representation approximates the path of the user’s pen over the tablet surface, and is capable of capturing both smoothly flowing curves and multiple sharp, discontinuous features, all within a single stroke’s path.

From the perspective of the user interface, an important consideration is when the curve fitting process will take place. Most vector graphic software that support an interactive curve fitting algorithm take a batch processing approach, collecting sample points from the user as he or she draws and only performing the curve fitting process once the user completes a stroke [Adobe, 2007a] [Macromedia, 2005]. As the user draws, a temporary visualization of the raw sample points is provided to give the user feedback, but once the stroke is completed this temporary display is replaced by the results of the fitting process. Some researchers have noted this batch processing technique does not replicate the traditional drawing experience, implying that such a system can even be distracting to the user who might not expect their input to change. To address these concerns, Di Fiore and Van Reeth for example developed an on-the-fly curve fitting system for their Multi-Level 2-D sketching tool, which performs the curve fitting process continuously as the user draws [Fiore & Reeth, 2002].

Di Fiore and Van Reeth make a compelling case for an on-the-fly curve fitting system, and so early development of the current project also targeted this feature. However, the development process exposed a number of drawbacks to this method. It should be noted that the methods used in this project may differ substantially from those used by Di Fiore and Van Reeth. In their publication the researchers

do not provide implementation details for their curve fitting system. However, they do mention that curves are represented as cubic Bézier splines—roughly equivalent to the curve representation used in the current modeling system. The authors also note that a least squares fitting process is used to convert the raw sample points, which again roughly corresponds to the fitting algorithm presented later in this thesis.

First attempts at an on-the-fly fitting system took the direction of refitting the entire curve as new points came in. As one might expect, this method quickly proved to be far too inefficient for an interactive system. Because the stroke is represented by a series of individual curve segments, it is reasonable to conclude that once an early segment of the stroke has been properly fitted, later sample points should have little to no effect on that portion of the stroke, and so refitting those segments is not necessary. Although it's true that early segments may not need to change visually after they are established, as further points are generated it may be possible to refit longer sections of a stroke with fewer cubic curve segments. Thus partial refitting schemes represent a tradeoff between interactivity and efficiency. Still, the representation of the stroke as a sequence of inefficient cubic curve segments is likely to be more memory efficient than the original raw sample points.

A second version of the curve fitting system was developed in which points were collected until a sample threshold was reached. At this point the set of samples would be fit in a single operation, limiting the effect of the fitting process to a smaller portion of the total stroke. This process continued in a stepwise fashion as the user continued to draw. This alternative method proved to be efficient enough to function at interactive rates, however it exhibited several troubling character-



istics. First, the divide-and-conqueror algorithm used to fit curve segments to the raw sample points tends to become unstable as the number of sample points available decreases. When only a few raw points are provided, rather than finding a comfortable path between the points the system will occasionally produce a contorted curve segments with drastic loops and uneven paths that do not reflect the path of the user's original input. The deviations are the result of the limited number of sample points provided to the system. With so few constraints, the fitting algorithm gets caught up in accounting for natural deviations in the sample points which would otherwise be averaged when larger numbers of constraints are provided. Although this situation only arose occasionally in testing, the visual effect is quite disturbing.

A second problem inherent in the sequential fitting system is the visual indication of the fitting process as the user draws. The stroke fitting process lags behind the user's cursor as he or she draws, and several times a second, the short set of unfit points directly behind the cursor are fit and redisplayed. This causes a distracting visual movement or jump as the path of straight lines connecting the raw sample points is slightly adjusted to a fitted curve segment. Although the effect is subtle, because the user's attention is focused on the location of the cursor this small movement catches the eye very easily. This problem is exacerbated if the size of the sample point set is increased. Increasing the sample point size has the effect of reducing the incidence of fitting errors because more raw data is available to the algorithm, however this also increases the length of the trail of unfit points behind the cursor and makes the visual jump of fitting those points more obvious.

Careful consideration of the results of the incremental fitting system suggested

several possible solutions. First, rather than filling a buffer with unfit points and then fitting the points all at once when the buffer fills, it could be possible to use a sliding window containing the last  $n$  or fewer sample points and refitting these each time. Some method or heuristic would need to be developed so that the fitting process considers a limited number of previous points, but still consider all points up to and including the end of the last fit curve. Another possibility might be to perform a basic fitting immediately as points are read in, as well as an optimizing algorithm concurrently on a separate thread of execution, which could follow behind the original algorithm and refine the chain of curves. Each of these approaches offers a possible solution to fitting errors, however because of the least-squares nature of the fitting process, any change to the sequence of curves used to represent the stroke will almost certainly result in slight movement of the changing portion of the curve.

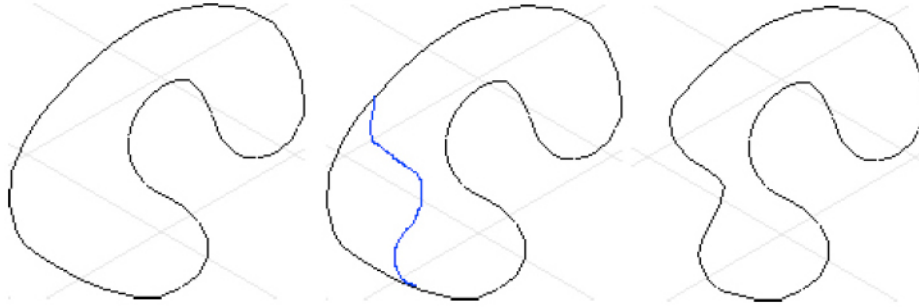
Given these issues and considering the design principles outlined in Section 6.2 above, it was decided that the most important aspect of the curve drawing system should be the comfort of the user. To this end, a batch processing system was selected as the least disruptive to the user experience. While the user draws, his or her stroke is visually depicted as a series of discrete (i.e. un-connected) white sample points representing the filtered raw input points. Once the pen is lifted and the stroke is finished, a single fitting operation is performed and the visualization is reconfigured as a single smooth stroke. The reasons for this particular visualization are two-fold. First, the dotted arrangement makes clear to the user that the visualization they are seeing is temporary. Second, by leaving the points unconnected, the user's visual system will naturally impose a smooth curve connecting the points, an effect referred to as the gestalt principle of *continuation*

or *closure* [MacEachren, 1995]. A similar method is used by the curve fitting system in Adobe Illustrator [Adobe, 2007a], and is an option for Macromedia FreeHand [Macromedia, 2005]. Although a visual jolt must occur when the pen is lifted and the sample points are replaced by the fitted stroke, by providing the user with a static display while drawing is still underway, and by using the temporary visualization, the jolt is both foreshadowed by the visualization and postponed until a natural transition in activity.

#### **6.4.7 Curve Correction**

Another stroke related feature under careful consideration for the application was a means for the user to make corrections to his or her strokes. As discussed in Chapter 2, sketching is a process of continual refinement in which the artist draws and redraws strokes to find the desired form—see Figure 6.10. Although the application is targeted at creating 3-dimensional models, some measure of this interaction is an important feature for the stroke input system in order to give the system a feel of sketching. As discussed in Section 5.1, a number of researchers have included stroke-based editing systems into their 2-D and 3-D sketch-based applications. Although realized through a number of different means, in general these systems allow the user to correct the path of a stroke by redrawing portions of that stroke, but leaving other areas intact. In the initial planning stages of this project a similar facility was considered as a feature of the program.

Along with research applications, a number of commercial vector graphics applications also include similar functionality. The pencil tool in Adobe Illustrator for example allows a user to create a stroke and then edit portions of that stroke by drawing a second path in proximity to the first. In researching these systems,



**Figure 6.10.** A demonstration of a stroke-based correction system utilizing overdrawing to replace a segment of an existing curve. Original figure from Pereira *et al.* [Pereira *et al.*, 2000].

examining examples of traditional pencil and paper sketches, and in experimentation with Illustrator, it quickly became apparent that determining the intentions of a user’s editing strokes is extremely difficult. First of all, it seems rarely to be the case that a stroke is created, partially edited a number of times, and then considered to be correct. Instead, as can be seen in Figure 2.4 on page 33, traditional sketch artists generally start with a series of possibly exaggerated strokes that approximate their subject, and then begin to build further “sketchy” strokes on that foundation. Once the desired shape comes into focus, it is not necessarily defined by any single line, but is instead the result of the build up of strokes in their totality that defines the final shape of the sketch, a visual phenomenon Henzen *et al.* termed a line hypothesis [Henzen *et al.*, 2005].

Even in situations where more discrete lines are to be edited, determining which portions of the line the user is intending to edit is non-trivial. Experimentation with Adobe Illustrator’s line editing system led to a similar conclusion. Although clearly Illustrator must employ some sort of heuristic to determine which portions of a curve should be replaced and which spared, user experience shows that the systems actions are routinely incorrect, and even frustratingly unpre-

dictable. As described in Section 5.1, the Pentimenti research application developed by John Alex [Alex, 2005] provides a means of making the editing process more explicit with an editing threshold widget. However, from the standpoint of the sketching experience, adjusting widgets and thresholds while drawing does not seem conducive to the creative process. Furthermore, the distance adjustment does not entirely disambiguate the user's input.

A further problem with curve editing systems is the fact that repeatedly editing a curve requires splicing new curve segments into the existing curve. Fitting a smooth sequence of curve segments to a single sample of points is effective because the points on which the resulting stroke will be based were created from a continuous physical action. However, after splicing together sample points from numerous discrete sampling events, the resulting curve no longer represents a single unified motion, and so it can be difficult to represent that path smoothly. The process of splicing and re-splicing the curve creates wiggles and bumps that can only be removed through smoothing operations, which simplify the curve by reducing its fidelity to the original sample points even further. Based on design principle V, an admonition against a suggestive interface, and the fact that this editing method does not reflect actual sketching practice, it was decided that a general stroke-based editing system was inappropriate for this application.

To develop an alternative method of sketch-like editing, further examination of traditional sketching practice was informative. This provided two additional observations. The first involves a common method of defining strokes in a sketchy fashion. In traditional sketching, an artist will often open with an initial stroke that defines only a short portion of the intended line and then ends in a quick motion. The artist then begins again some distance down the original stroke and

creates a second stroke that extends the original along the intended path, but again ends with a quick flourish. This process continues until the entire length of the desired stroke is defined. This technique—we will call *casting strokes* because of the distinctive casting motion used to describe the stroke’s path—is often used when defining long smooth curves, especially closed, circular curves for which the artist must continually reposition in order to bring the focus of drawing into alignment with the most dexterous range of his or her hand.

This casting behavior is of note because, unlike the more general stroke-based editing described above, the viable portion of a casting stroke should be much easier to determine. The casting correction starts at a point on the original stroke in proximity to the beginning of the new stroke and proceeds in a single direction replacing the remaining extent of the original line with the new stroke. The drawback to this system is that, although it may aid the user in extending an existing stroke, it does not provide a means of making more drastic corrections. A further detraction stems from the fact that the correction replaces the original path of the line. For the sketch artist the paths of previous lines provide a framework and structure on which to base further refinement. In addition as discussed in Chapter 2, although extraneous strokes may not reflect the artist’s original intentions, they serve as fodder for the mental feedback system that drives the sketching process. By replacing portions of the line completely this record is lost. Because of these issues, although some consideration in the design process was directed towards the casting concept, it was eventually decided that a different solution was needed.

The second observation of traditional sketching technique involves the stages of a traditional sketch. As described in Section 2.3.1, Huot *et al.* in examining

sketching architectural students noted that the sketching process tends to progress in stages [Huot *et al.*, 2003]. As the drawing is developed, early strokes are predominantly used for defining a general shape. These strokes then give way to more controlled and detailed overdrawn strokes that refine the shape of the sketch. A similar pattern can be seen in work by character artists. Henzen *et al.* for example [Henzen *et al.*, 2005] describe how an animator or illustrator will create and refine a sketch, and then pass his or her work off to a cleanup artist. It is the cleanup artist's task to redraw the original sketch by removing the nervous and sketchy characteristics of the existing strokes and replace them with a single clean line that captures the shape and character of the original, in essence extracting Henzen's line hypothesis from the original drawing. The cleanup stage may involve erasing or replacing the original sketch, but in other situations the new strokes are placed directly on top of the old. To facilitate this process, the original sketch may be drawn with a special photo-blue or red pencil to help distinguish the old lines from the new, or the cleanup artist's work may be performed in ink ovetop of the original pencil sketch. An example of these techniques can be seen in Figure 6.11.

The concept of a cleanup system provides a much better technological analogue to traditional sketching. Both Henzen *et al.*'s 'nervous hand' attractor based editing system [Henzen *et al.*, 2005] and Alex's use of history strokes [Alex, 2005] represent efforts to capture aspects of this technique in computer-sketching systems. For the current application the following design for a cleanup artist stroke system was devised. When producing sketch lines, the system maintains a current pen type, which reflects the system's interpretation of how strokes created with that pen will be utilized by the 2-D to 3-D conversion system.<sup>2</sup> Each line

---

<sup>2</sup>The conversion system will be explained further in Section 6.4.11.

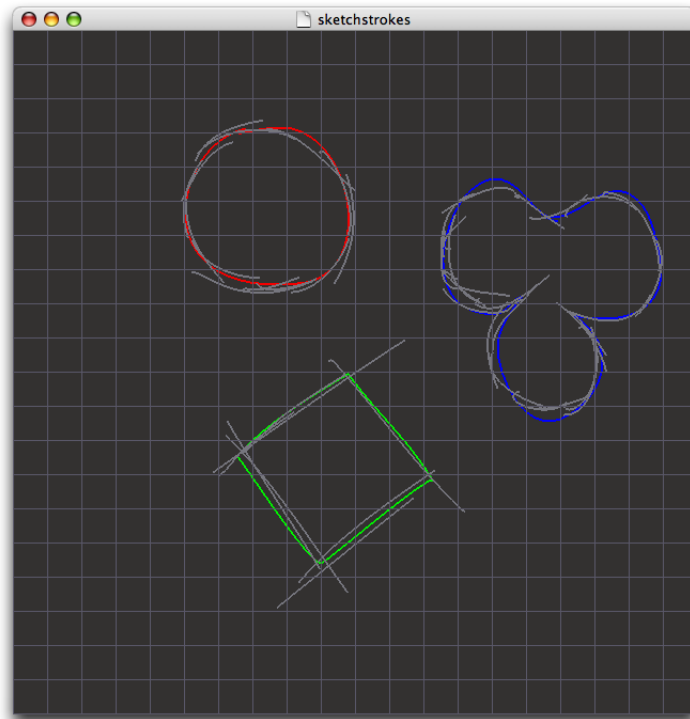


**Figure 6.11.** A cleanup artist takes the rough, sketchy original drawings generated by the animators (left), and extracts from that amalgamation of strokes the ‘line hypothesis’—individual clean lines (middle). This ‘cleaned’ drawing then moves along to later stages of the animation process (right) such as ink and paint.

entered by the user is color coded with one of three colors to visually indicate the application’s interpretation of that line as one of three types of modeling strokes. For any given pen type, the conversion system can only accept a single stroke, and so as the user draws multiple strokes with a single pen, only the last stroke entered is coded with the color of the current pen. Previous strokes, once replaced, are not removed, but instead are re-colored with a neutral grey tone to indicate to the user that the stroke will no longer be considered by the system. However, although the system will subsequently ignore these past strokes, they remain completely visible to the user, acting in the same manner as overdrawn sketch strokes in a traditional paper-and-pencil sketch—see Figure 6.12.

By including all of the strokes that are entered, the user can utilize the same feedback and refinement mechanisms used in traditional sketching. Once a shape has been sufficiently refined by the user, he or she simply redefines the shape with a single coherent cleanup stroke, guided by the framework and history of strokes





**Figure 6.12.** An example of the use of extraneous and contmeplitive strokes with the sketch system. Here the user has made a number of sketchy strokes (grey lines), as a basis for solid clean final lines (colored strokes).

built up in the sketching process. This final stroke automatically becomes the active stroke for the current pen type, meaning both that it will be utilized by the system for modeling operations, and that the color of the stroke is changed, visually separating this refined stroke from the collection of overdrawn strokes used to develop it. The effort to preserve and utilize these non-modeling strokes reflects the goals expressed by design principle IV. In addition, because this update process is automatic, it requires no explicit mode switching or parameter alteration on the users part. This allows the user to create stroke after stroke without further interactions with the application, following design principles I, II, V, VI, and VII. Furthermore, because the resulting stroke is defined as a single input event, the raw

sample points used to construct it represent a single coherent sample. This means that the subsequent fitting process can produce a smooth curve that precisely reflects the user's intentions without editing artifacts like bumps and ripples.

#### **6.4.8 Drawing Interface**

The user interface of the application provides a number of controls over the way that strokes are accepted by the system. As noted in the interface overview in Section 6.4.2, the stroke tool is used to indicate to the system that input from the tablet should be interpreted as stroke drawing. Because stroke creation is the primary interactive mode of the application, no specialized interface settings or alternate commands are provided for the sketch tool. It's instead assumed that the user will spend most of his or her time in the sketch tool, and utilize special commands, keyboard modifiers, and tablet gestures to temporarily access other tools within the program. This design is in keeping with the design principles laid out in Section 6.2.

The user interface provides several features and parameters that the artist uses to manipulate the behavior and interpretation of the stroke tool and drawing input. The first feature of the stroke system is the ability for the user to close an open stroke. It's possible to create a visually closed stroke simply by completing a stroke in close proximity to its initial point or possibly overlapping some small distance with the initial path of the curve. This is the method used in traditional sketching to draw what appears to be a closed path. While this method is sufficient for 2-dimensional sketching, when sketch strokes are used to generate 3-dimensional geometry, a visually closed path is occasionally insufficient. What may appear to be closed in 2-dimensions could create unsightly gaps or overhangs

in generated 3-D geometry. To remedy this problem, the user has the option of forcing a stroke to be geometrically closed by pressing and holding the `OPTION` modifier key after a stroke has begun and until the stroke is finished. This has the effect of adding a single additional sample point to the end of the raw points generated by the tablet. This additional point is coincident with the initial sample point and causes the fitting system to join the two ends. This point is always interpreted as a corner for algorithmic reasons, however the junction can be made visually smooth by ending the stroke in close proximity to the initial point, just as one would in a traditional sketch.

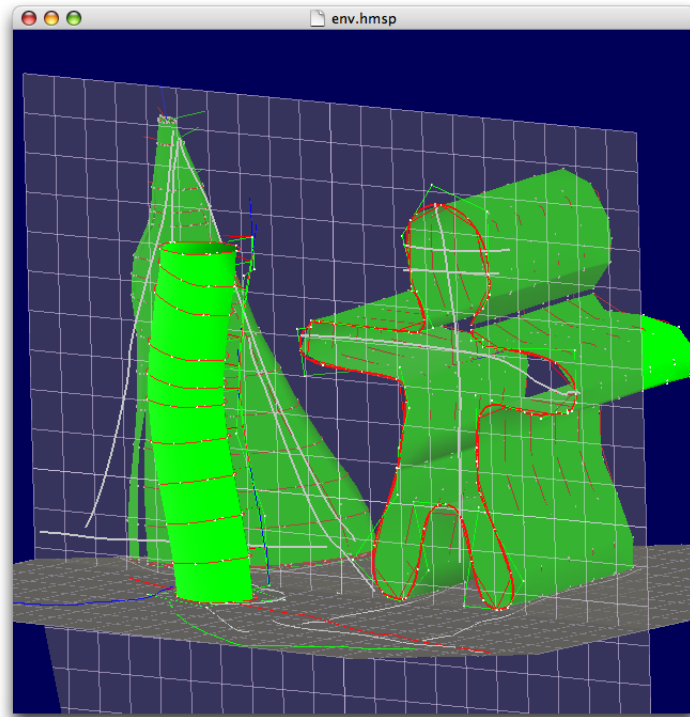
Another aspect of the stroke system is the current pen settings. As described previously, the current pen indicates to the system the interpretation of any newly created strokes. These interpretations are used by the 2-D to 3-D construction system in order to create 3-dimensional geometry. Because most constructed geometry will require several strokes, each with a different interpretation, switching between input pens is a common activity for the user. The tool pallet provides buttons for each pen to allow the user to select the current pen type explicitly. In addition, two other methods provide more direct access to the current pen setting that can be accessed more conveniently while drawing. First, by striking the left and right arrow keys on the keyboard, the user can cycle through the list of pens. This same functionality is also offered from the tablet by using the flick gesture described in Section 6.4.4.4. The user takes up a flick rest position above the tablet's surface, and then depresses the upper barrel button to activate the flick system. Upward and downward flicks will then be interpreted to cycle through the list of pens. The flick recognition system remains active until the user releases the upper barrel button, at which point he or she can continue with the drawing.

### 6.4.9 Environment and Drawing Surfaces

The modeling environment is the 3-dimensional space visible to the user through the document window. When a fresh document is first opened, the only content of the environment is a single drawing plane situated parallel to the viewing plane and positioned such that it fills the entire viewing area. In the environment drawing planes are depicted as flat 2-dimensional squares floating in space.

Initial designs called for each plane to have a white color to simulate the look of a paper drawing surface. However early visual tests showed that using a background with such a high brightness obscured details of strokes and other geometric elements. Instead, a dark, low saturation color scheme was selected for all interface components including drawing planes. This has the effect of pushing these elements into the background. Geometric elements like strokes and model components on the other hand are depicted with bright and highly saturated primary colors. This contrast visually separates constructed and user supplied elements from the dimmer, darker background, and makes small details easier to perceive. A view of the environment is provided in Figure 6.13.

In accordance with this color scheme, drawing planes are drawn in a dim purple-gray background and covered with a network of lighter but de-saturated purple-gray grid lines. These lines serve two purposes: first, because the camera angle may be situated arbitrarily in the 3-D environment, the grid lines help to orient the user in relation to the drawing surface; second, because a typical construction will require multiple planes to hold each defining stroke at the appropriate position in 3-D space, the grid lines help to highlight the ways in which neighboring drawing planes intersect or overlap each other.



**Figure 6.13.** A view of the document window of the application containing a model in progress. In the image you can see two drawing planes. The plane along the bottom of the view is active, whereas the more vertical transparent plane is not. Several modeling components are shown in green. The components are adorned with various additional lines in red and green. These are the strokes that constructed the surfaces as well as geometry generated during the conversion process. In a normal application these would be hidden. Finally we can see a number of strokes not connected to the modeling components. Strokes in grey are sketch lines ignored by the system. Those in red, green, or blue, are the current active strokes for each stroke pen.

One additional property determines the visual appearance of each plane. At any given time one of the planes is designated as the current drawing plane. The current drawing plane is the recipient of any strokes drawn with the stroke tool, as well as any plane adjustment commands issued by the plane tool. To differentiate the current plane from the other planes, all non-current planes are drawn with a 50% transparency. This both indicates to the user which plane is the current

plane, and allows the user a clearer view of the entire selected plane and the modeling environment as a whole.

Switching between planes in the interface can be achieved by two separate means. The most direct is to explicitly select the desired plane from the list of planes displayed in the plane palette. The currently selected plane is always highlighted within the plane palette's list. Although this is the most explicit interface, it is far from convenient. One additional method is also provided that allows the user access to other planes while drawing. The user can strike the TAB key, which will cycle through the list of planes according to their order in the list displayed in the plane palette. This interaction is designed to simulate the COMMAND+TAB application switching function familiar to most users.

Although each drawing plane is visually depicted to have a finite square extent, each plane is internally represented as an infinite expanse that divides the viewing area in two. When the user draws, strokes that extend beyond the visual boundaries of the drawing plane land in mid air, projected onto the invisible surface of the infinite drawing plane. The user can optionally instruct the application to draw any given drawing plane so that it extends to fill the visible region of the modeling environment by activating or deactivating a checkbox for that plane in the plane palette. Similarly, any plane can be marked as completely invisible using a similar method. When a plane is invisible, any strokes drawn onto that plane are also invisible, however modeling components derived from those strokes will still be rendered.

Original designs for the application also called for a second set of modeling structures called the *modeling stage*, similar to the modeling stage designs of Grossman *et al.* [Grossman *et al.*, 2001] [Grossman *et al.*, 2002] [Grossman, 2004]

and Tsang *et al.* [Tsang *et al.*, 2004], consisting of the back three faces of the viewing area, also to be drawn with a network of grid lines. The intention of this design was to provide a further orienting landmark for the user, as well as an indication of the extent of the visible modeling environment. Although this design is entirely viable, it was decided that instead the background should be depicted simply as a solid color, with no indication of the extent of the visible area. Thus surrounding the focal point of the view is a solid dark blue background with no discernible features. This arrangement has the effect of suggesting a much larger open environment, similar to painting clouds on the walls of a cramped room. As portions of the model or drawing planes extend beyond the actual viewable area of the modeling environment their extent is simply clipped by the graphics hardware. This more subtle indication of the visible area provides the user with similar spatial feedback to the stage environment without enclosing the view in a confined space. As for the orienting aspects of the stage environment, because the solid background does not provide a single standard orientation, the user is more comfortable viewing the model from rotated angles that may not align with the original up and down directions.

When considering the design of drawing surfaces for the application the striking differences between 2-D and 3-D sketching begin to become apparent. When sketching in 2-D virtually no consideration needs to be given to the drawing surface beyond “will my drawing run off the page?”. For the proposed 3-D environment however, creating and positioning drawing planes becomes extremely important. Because the drawing process is inherently 2-dimensional, it is the position and orientation of the drawing planes that will provide the 3-dimensional information to the modeling process. Despite the importance of the drawing planes, this is

also an area for which the user is likely to have little previous skill. No traditional sketching techniques correspond to this activity. Therefore, a goal of the drawing planes' design was to develop a system that is both simple to use, and can integrate with the more familiar sketching aspects of the program.

The first task is that of positioning and orienting a plane. The plane tool provides the user with three distinct operations that can be used to adjust the position of a plane: *rotation*, *translation*, and *zoom*. The first operation is rotation, which serves as the default operation when the plane tool is selected. This default was chosen because the most common operation when dealing with the plane is the need to rotate its position. The extrusion-like method of the 2-D to 3-D conversion system involves defining a cross-sectional shape with one stroke, and then applying additional strokes in relation to the cross section in order to define the model's extent. This requires the user to position two planes roughly perpendicular to one another, which is achieved through rotations. The rotation mode is indicated to the user by the green color of the delta cursor used by the plane tool.

Without further input from the user, the standard operation of the plane rotation tool is to rotate the current plane about a central point. This point is indicated by a small white dot that appears on the surface of the plane. As the user drags the mouse, the plane rotates about this point in response to the mouse's motion based on a standard 3-D trackball idiom. The user can easily set the center of rotation to any point on the plane's surface by holding the SHIFT modifier key and clicking at the desired location. To aid in identification, the rotation center point is drawn on top of any other interface or modeling elements, however its position is always in contact with the plane's surface.



The free rotation interface provides the user with complete control over rotation of the plane along the current viewport's x and y-axes (pitch and yaw). However, in some circumstances it is beneficial to constrain the rotation to a single axis. This feature is provided by an interface idiom called a *crease line*. When holding the SHIFT modifier key, rather than entering a single click the user can click and drag to create an arbitrarily oriented straight line on the surface of the current plane. This crease line, which is drawn in white on the plane's surface, will serve as the axis of rotation for further rotational adjustments. Subsequent clicks while holding SHIFT will replace the crease with rotation center points again, whereas subsequent drags will replace the crease line with a new crease. As with the rotation center point, the crease line remains visible once defined by the user as long as the plane tool is active, and is rendered in front of all other interface elements even through its position on the plane's surface might normally be occluded. Once the crease line is defined, succeeding input from the plane tool will no longer freely rotate the plane, but instead perform rotations confined along the crease axis.

The two remaining positioning operations have more straightforward interfaces than rotation. The second positioning operation allows the user to translate a plane parallel to the current viewing angle. Whereas rotation is a single click and drag while the plane tool is active, double clicking with the mouse or double tapping with the stylus accesses the transition mode. Once active, translation is indicated to the user by the red color of the plane tool's delta cursor. Similarly, a triple click or triple tap with the tablet accesses the zoom function, which is indicated by a blue cursor color. The zoom function is used to translate the plane along the depth axis of the current viewing plane, in effect pulling the plane

forward or pushing it back. The degree of movement is controlled by the y-axis displacement of the mouse while the zoom mode is active.

At the start of the application a single default plane is automatically created. This plane provides a good starting point, but in order to generate interesting 3-dimensional models, additional planes must be created. To create a new plane coincident with the current plane, a small button is provided at the bottom of the plane palette. However, this is not the most efficient means of creating new planes. Just as 2-dimensional sketching is an evolutionary process that builds upon previous strokes, the plane system is designed to aid the user in constructing the necessary series of planes in an incremental fashion. Rather than planning out exactly how many planes might be necessary and positioning them before the drawing process, plane creation is integrated into the plane tool's positioning capability so that new planes are created as the user works.

The standard pattern of behavior for the modeling process involves creating constructive strokes on one drawing plane and then positioning a new plane in relation to those strokes. This means that, from the user's perspective, it is natural to think of the position of the new plane as a function of the position of the current plane. Furthermore, whenever the user creates a new plane, that operation will routinely be accompanied by a subsequent positioning operation. In fact because of the constructive and visual nature of the design process, it is most likely that the positioning operation is the focus of the user's intentions and forefront in his or her mind.

To aid the user and integrate the plane creation process into the standard workflow the following design was developed. If a plane does not contain any strokes, then the plane is considered to be clean, and any positioning operations—rotation,

translation, or zoom—will affect the plane as described above. However, if the plane does contain strokes, then it is designated as *active* by the system. Active planes cannot be repositioned because their positional information is needed by the strokes they contain in order to provide them with a 3-dimensional location. Instead, when positioning operations are applied to an active plane, the system automatically generates a clone of the plane situated in the same position as the original, but containing no strokes. The cloned plane is automatically selected as the current plane, and becomes the recipient of succeeding positioning commands. In this way, rather than requiring a separate mode of operations or additional tools to create new planes, planes are instead created in relation to the user's current focus, and are generated automatically as part of the positioning operations that is the focus of the user's intentions.

Each of these plane adjustment operations can be accessed explicitly by selecting the plane tool from the toolbar interface. However, during the modeling process the user will frequently be switching between the tools related to the core sketching operations outlined in design principle II. Rather than requiring the user to frequently break concentration on the modeling task to switch tools in a separate display, the plane tool's operations can be accessed using the keyboard and tablet or mouse alone while any other tool is in use. Whenever the OPTION modifier key is depressed, the plane tool is activated and input from the tablet or mouse will be interpreted as plane adjustments. This allows the user to quickly perform a plane alteration without changing focus from the document window. By using the program with one hand on the keyboard to activate this and similar modifier keys, or by using a tablet equipped with modifier key proxy buttons, the use of the tool palette can be avoided all together.

In addition, the operation to define a new crease line can also be accessed from the stroke tool using the pounce tablet gesture described in Section 6.4.4.2. By generating a quick high-pressure event at the beginning of a stroke, the stroke will be canceled and the mode will be temporarily switched to define a crease line. When the stylus is again lifted from the tablet's surface, the interaction state will return to the previous tool automatically. This operation is designed to be reminiscent of the physical activity of setting a crease tool into the surface of the drawing medium before drawing out the path of the crease.

#### **6.4.10 View Adjustment**

Another aspect of the modeling system that stands in stark contrast to the traditional practice of sketching is the issue of viewing the model. It is true that the traditional sketch artist must consider the viewpoint of the subject of a sketch before putting pencil to paper. However, once a viewpoint is selected, that view is static. A 3-dimensional modeling application affords the designer the ability to consider the model from any arbitrary viewpoint at any time during the construction process. This is a boon in the sense that the user has a clear picture of the 3-dimensional structure of his or her subject, and can define the subject from all angles in a single work. At the same time, this raises the complexity of the sketching process because the artist is required to consider the model from all sides.

For the most part, although traditional sketching is a 2-dimensional medium, sketch artists have trained themselves to consider the 3-dimensional form of the subject. By using visual cues and drawing techniques, the artist creates a sketch that suggests the full 3-dimensional form of the subject even though the sketch

lacks information about the depth of the subject or its appearance from alternate angles. The modeling application allows the user to define these details more directly, and in a straightforward manner instead of relying on visual cues and suggestions of depth information.

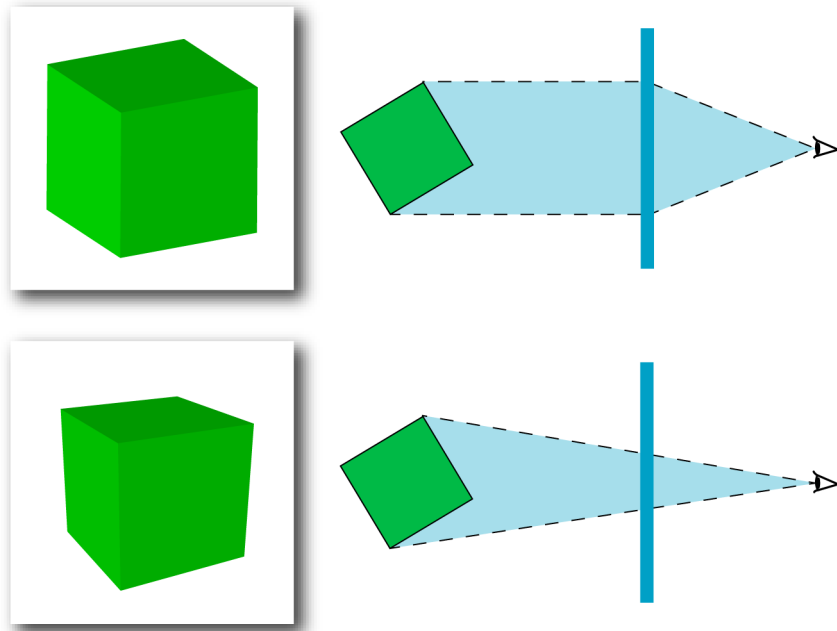
A drawback to the modeling system however, is the fact that the computer hardware only offers a 2-dimensional representation of the 3-dimensional modeling environment and its contents. Whereas a sketch artist working on a flat piece of paper, or a sculptor working at a physical model can easily and naturally change their perspective on the work by moving their physical bodies or adjusting the direction of their gaze, the computer must be given explicit instructions on the viewpoint of the 3-D environment. Barring more exotic display systems and high-dimensional input devices, this is a basic limitation of modeling software targeted at commodity personal computer platforms. This situation is unfortunate because view changes are a common operation, as indicated by their status as one of the core operations defined in design principle II. The design challenge then is to develop methods of adjusting the view that are both expressive enough to allow the user to position the viewpoint where he or she is most comfortable, and at the same time make those operations simple and intuitive enough that they do not disrupt the creative process.

The user's view of the 3-dimensional environment is displayed in the document window as a 2-dimensional projection. For any 3-D application, there are two standard methods of transforming the 3-dimensional geometry into this 2-dimensional depiction: an orthographic projection, and a perspective projection [Angel, 2003], as represented by the diagram in Figure 6.14. The first and most straightforward is an *orthographic* or *orthogonal projection*. In this view, each

point in the 3-dimensional environment is projected against the viewing plane along parallel paths, which are in turn perpendicular (orthogonal) to the viewing plane itself. This sort of parallel projection is commonly seen in traditional engineering drawings and schemata because the projection preserves important geometric features of the 3-D subject such as relative size, length, and parallel faces.

The second method is referred to as a *perspective projection*, or *perspective view*. In this model, each point in 3-dimensional space is projected onto the viewing plane along paths that converge to a single point at some distance behind the viewing plane. This arrangement causes objects further from the view to appear smaller in relation to closer objects, an effect referred to as *diminution* or *foreshortening*. By projecting the points in this manner, geometric properties such as size and the relation of parallel lines are lost. Despite this, perspective projections more closely model the physical properties of our eyes, and so especially for larger scenes containing many objects, the resulting projection appears more realistic.

The current modeling application is configured to use a basic orthographic projection. This selection was made for two reasons. First, because of its more straightforward mathematical representation, the orthographic projection partially simplified the early design and testing process. Second, because the application is targeted at the creation of small, individual models or components at the early stages of design the orthographic projection provides a clearer view of the artist's work and is more appropriate for the desired output. Perspective effects are more appropriate for later stages of the modeling process when multiple components are combined in larger virtual environments, and where visual realism for final output is an important consideration. Although the current de-



**Figure 6.14.** The top diagram shows a conceptual model of a parallel or orthogonal projection. Notice that the lines of sight directed off of the subject are parallel until they strike the viewing plane, where the projected image is seen by the eye. In contrast, the perspective projection depicted in the bottom example projects the subject's features along converging lines of sight. To the right of each diagram is an example rendering. Notice that although the edges of the cube in the bottom perspective rendering are not parallel, the image overall appears more natural, whereas the orthogonal rendering above it looks distorted in comparison. The viewer should note that the perspective effect in the bottom image has been exaggerated to emphasize the difference between these two projection methods.

sign does not support perspective views, the application's implementation is not incompatible with perspective projections, and this feature will likely be included in future releases.

View alterations in the present application are performed using the camera tool, which, like the plane adjustment tool, offers three modes of operation: *translation*, *rotation*, and *zoom*. When the tool is selected from the tool palette, the default operating mode is translation. Note that this default is in contrast to the

default rotation operation used for the plan tool. This design decision was made after initial testing with the program revealed that translation operations were more intuitive as the default operation for the camera tool. When the user clicks and holds the mouse, or places the pen tip on the tablet surface, the translation mode is activated and a small red-filled anchor point icon is displayed at the location of mouse or pen down. As the user drags the mouse or pen away from the location of the anchor point, the view begins to continuously translate parallel to the viewing plane in the same direction as the cursor deflection.

The speed of the translation is determined by the distance between the cursor's current location, and the location of the anchor point. To provide an intuitive physical response to this input mechanism, the speed of the animation is modeled after Hooke's law of elasticity, which is the physical law approximating the ratio between distance and force in a linearly elastic system such as a spring [Resnick *et al.*, 2002]. Hooke's law is expressed as:

$$F = -kx \tag{6.1}$$

in which  $F$  is the restoring force of the elastic body,  $x$  is the displacement distance, and  $k$  is the elasticity constant. In the context of this application the force variable of Hooke's Law is interpreted as the rate at which the view's position should change. Taking the place of  $x$  is the distance between the cursor and the anchor in world-space coordinates. The elasticity constant is taken as negative to cancel the negative component of the equation above, and was tuned during application development to produce a reasonable response rate from the system. Translation of the view continues as long as the mouse button or pen remains down, regardless of further mouse or pen movements. When lifted the operation ceases.



The rotation function is accessed by double clicking or double tapping and then dragging the mouse or digitizing stylus while the camera tool is active. At the point of the double click or tap, a green anchor point icon is displayed. As with the translation function the distance between the cursor's location and the anchor point controls the speed of rotation. The icon also serves the additional purpose of marking the center point of rotation. When the user double clicks or taps, the location on the viewing plane is projected into the modeling environment, striking the forward most object. The location of this intersection is taken to be the center of rotation until the mouse or pen is released.

The zoom function works along similar lines. When the user triple clicks or triple taps a location on the document window a blue anchor point marker is displayed and the zoom function is activated. While the mouse or pen are down, the distance between the cursor and marker location is used to control the speed of a uniform scaling of the modeling environment and its contents, centered at a point projected from the anchor point's location. Negative deflection along the y-axis corresponds to a positive scaling and a zoom-in while positive deflection causes a negative scaling and zoom-out.

As with the plane tool, although the camera tool can be accessed explicitly from the tool palette, the application also features a set of keyboard and tablet commands that can be used to access the same functionality with less disruption to the drawing process. First, the view alteration tool is assigned to the second mouse button, allowing easy access when using the modeling application with a standard mouse rather than the drawing tablet. Furthermore, right clicks and drags can be simulated in the system by holding the Control modifier key while using the standard left mouse. These two alternate methods provide slightly more

convenient access to view alteration functions, however to truly integrate view alteration into the drawing process a more tablet-centered solution was needed. Once again, the inspiration for the design of the viewing system was taken from an examination of the physical process of sketching.

Although, as we have discussed, most movement operations in traditional sketching are performed by natural body movements, there are situations in which the artist will adjust his or her work directly. The best example of this behavior is when an artist repositions the paper under his or her hand. Although the joints of the fingers, wrist, elbow, and shoulder provide an amazing degree of dexterity to the human hand, there are certain ranges within the full range of dexterity in which the hand is most comfortable, and its owner has the most fine grained control over its movement. This fact becomes intuitively obvious if you consider how difficult it might be to write your name holding the pen in your clenched fist, or draw a picture with your arm immobilized in a cast. Taking another example, imagine writing a letter longhand on a tablet of paper. As your words fill the length of each line, you must continually lift your writing hand from the surface of the paper and reposition it further down the line so that you can comfortably reach the location where the next letter is to be written.

The letter shapes of English orthography make it possible to form each letter from a roughly consistent angle with the page, however, for the larger, more intricate, or more dynamic shapes used in drawing, it is often necessary not only to reposition your hand, but to reorient the paper as well so that the target of your strokes aligns with the most comfortable range of your manual dexterity. The behavior used to achieve this motion is a well-coordinated and fluid movement. The artist first lifts his or her dominant hand from the surface of the work, pulling

back or forward in the direction of the new target region. Next, the artist's non-dominant hand deftly repositions the page by sliding or spinning the paper while the other hand is held just above. Finally, when the correct alignment is achieved, the dominant hand returns to rest on the surface of the paper, where it is now positioned to continue working. This sequence of movements is performed as a single fluid motion and may take place dozens or hundreds of times during a drawing session.

The digitizing tablet is designed to replicate the feeling of working with pen and paper by mapping the area of the screen to the tablet's surface. In the case of writing, the tablet can be used with much the same motion of repositioning one's hand while writing a letter. In the case of repositioning paper while drawing, the physical activity while using the tablet is somewhat different. The tablet, because of its physical features—the power cable, sculpted corners, rubberized feet, and its general thickness and heft—is not designed to be repositioned but is instead intended to remain stationary. It is important however to recognize the disconnect between the tablet, which serves as a drawing surface, and the computer display, which provides the visual feedback of the drawing. The orientation of the image on the computer's display can be arbitrarily positioned. This means that the user's hand can remain in the same general position while the screen display is reoriented to align input focus with the range of maximum dexterity.

The application's view alteration system is designed to utilize similar physical movements in the user's dominant hand to control the position and orientation of the view. This gesture is referred to as *lift-and-lead*, and is derived from the brush-off tablet gesture described in Section 6.4.4.1. The lift-and-lead gesture functions as follows. The user begins by holding the digitizing stylus above the

tablet's surface; close enough to remain in proximity to the tablet sensors, but far enough so as not to touch the tablet's surface. From this position, the user activates the view adjustment system by pressing and holding the lower barrel button of the digitizing stylus. This has the same effect as a mouse down event, setting the view anchor point and beginning a view adjustment. The adjustment continues until the user releases the button, triggering the corresponding mouse up. The intention of the gesture is to simulate the psychological movement of lifting the dominant hand while the drawing paper is repositioned beneath. The motion of the paper is simulated by the animated effect of the positioning system as previously discussed, while the motion of the user's dominant hand above the tablet directs the view adjustment similar to the way the sketch artist's dominant hand tends to deflect in the direction of the movement of the paper. The user directs the view alteration by hovering the stylus over the tablet, and when the operation is complete the desired location has been repositioned to coincide with the area on the tablet to which the user has directed it.

Making further use of the tablet information, selection of either the translation, rotation, or zoom tool is made according to the angle of the digitizing stylus when the barrel button is activated. When the button is first depressed, the system records the angle of deflection of the stylus along the x-axis (about the y-axis). A deflection angle of between  $18^\circ$  and  $90^\circ$  (a comfortable pen-style grip with the right hand, or a pointing style grip with the left hand) corresponds to the translation mode. A deflection angle of between  $-18^\circ$  and  $-90^\circ$  (a comfortable pen-style grip with the left hand, or a pointing style grip with the right hand) is assigned to rotation. Angles of deflection in the range of  $-18^\circ$  to  $18^\circ$  (holding the pen roughly parallel to the sides of the tablet) selects the zoom tool. Once

the initial determination of angle is made, the pen can be reoriented during the operation without changing the mode. Positional motion while using any of the above modes has an identical effect to the translation, rotation, and zoom view adjustment operations described previously.

#### **6.4.11 2-D to 3-D Conversion**

Stroke input is the primary interaction method for the application, however, in terms of functionality, the core operation of the application is to take that stroke input and develop from it a 3-dimensional representation. This section describes in detail how 2-dimensional stroke information is interpreted by the system to create 3-dimensional geometry.

##### **6.4.11.1 Conversion System Design**

As was discussed at length in Chapter 2, from the physical perspective sketching is a 2-dimensional activity, but to both the artist and viewer, sketching contains a wealth of 3-dimensional information. Some of this information is conveyed using simple tricks and visual cues, which are effective shorthand for the viewer but do not necessarily reflect the true 3-D form of the subject. Other elements of the sketch are derived from careful visual observation of the subject's 3-D shape. These details are a reflection of the artist's attention to depth and physical space, and replicate in 2-dimensions the sample visual phenomena that make something 'look' 3-D.

As projects demonstrated by other authors in the field attest, a common avenue for exploration in sketch-based modeling has been the attempt to capture and interpret these 2-dimensional artistic cues and use them to control the for-

mation of 3-D geometry. Methods along these lines include line labeling systems, which attempt to literally interpret the geometric relationships between geometric primitives; inflation systems that rely upon an assumed 3-dimensional shape; some gesture systems, which try to capture the action of drawing these artistic cues and reinterpret them as modeling commands; and shadow systems, which try to extract from artistic shading and light effects an estimation of the 3-D object that would cast them. Each of these methods has given rise to interesting new modeling interfaces and even viable commercial products, however from an artistic point of view, each suffers from the same limitations. These methods all take as input a 2-dimensional drawing and attempt to reconstruct or extract 3-D information—information that is inherently impoverished.

These artistic techniques and depth cues are not the result of the physical characteristics of the subject, but only approximations and artistic interpretations of those features. This means that data drawn from shading information or apparent occlusion in a hand or computer drawing make extremely poor input to algorithms that assume they are receiving a whole and coherent description. The data is instead incomplete, and even possibly contradictory, and so the algorithm is left to try and sort out a best guess, a process that is both computationally complex and prone to error. The situation is exacerbated by sketch style input, which is by its very nature inexact, unplanned, and generally messy.

Gesture based systems attempt to address these issues by cataloguing and co-opting common drawing idioms as direct commands to the construction of 3-D shapes. This approach is an improvement from the perspective of model construction because the user's intentions can be addressed directly rather than vaguely inferred. Targeted at a full-featured modeling application intended for

later stages of the design process, gesture systems have a great deal of potential. However for the other end of the market catering to the early design stages, heavy use of gestures hampers the creative process by discretizing the full variety of strokes expressed in a drawing into a limited number of interpreted commands.

More effective as a medium for early design stages are inflation-based systems, which work directly with the user's drawing input to produce 3-D forms. The inflation method allows even inexperienced users to quickly and easily create 3-D models, however in exchange for this ease of use the system must make assumptions about the desired 3-D shape, leading to distinctly rounded and bulbous models. Inflation systems are well suited for entertainment applications, but are simply too limiting to be an effective professional tool.

An aspect of the drawing process that most of these other methods fail to consider is this: although the final product of a sketching session is 2-dimensional, the thinking process used to render the sketch calls upon the 3-D information perceived and processed in the artist's mind. In other words, the artist pictures the 3-dimensional shape in his or her mind, and then translates that information into the 2-D sketch. We can tell that this is the case because of the very same depth cues and artistic techniques the artist uses to embellish the drawing. Without perceiving or imaging the actual 3-D form of the subject, the artist would have no information on which to base the use of those cues and techniques. The information in the artist's head is translated into 2-D not for any aesthetic reason, but because this is all that's available with pencil and paper. If an application can provide a tool to directly capturing this 3-dimensional information, then the brief layover between concept and model as a 2-dimensional sketch becomes unnecessary.

Historically standard point-and-click 3-D modeling applications have offered this direct connection, but wrapped in a cumbersome interface that is far removed from the drawing process. The goal of this project is to provide a construction method that strikes a balance between the ease of use and resemblance to standard drawing of the inflation based systems, with the greater level of expressiveness found in gesture systems.

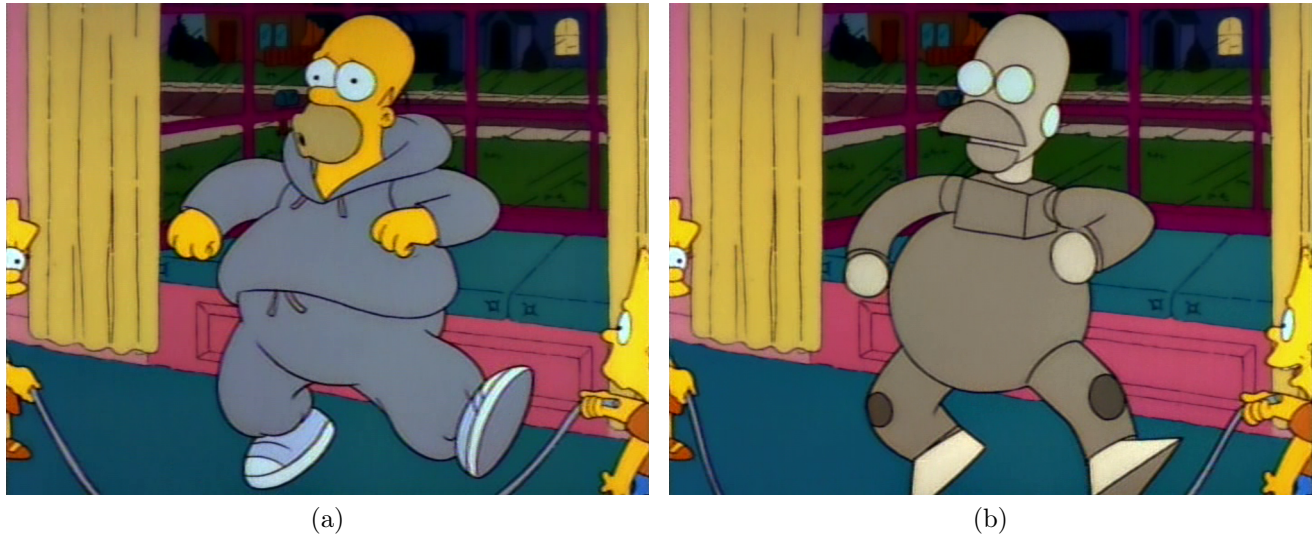
#### **6.4.11.2 Artistic Basis**

Artists are trained, when looking at a subject, to deconstruct its physical shape and form into more simple constituent shapes. These shapes include such basic building blocks as boxes, balls, tubes, and cones. Most objects are obviously far more complex than these simple shapes, however cones and boxes are much easier to draw than a person's hand or a sitting dog, and provide generalizations of those more complex forms that allow the artist to focus on high level relationships before low level details.

Most artists do not draw complete literal boxes or cylinders with clearly delineated tops and bottoms, but the gentle suggestions of these shapes on paper form the first strokes of the drawing. By starting out with these basic shapes, the artist can construct a very general representation of the model's overall structures, focusing his or her effort on attaining the proper proportions and relationships between different parts of the model. Once these basic forms have been established, the artist can then use the collection of shapes as a framework or scaffolding on which to hang more detailed descriptions of structure. As we discussed in Chapter 2, a similar and related technique is the basis of sketching. The artist begins with simple shapes and through a process of feedback and incremental refinement, builds



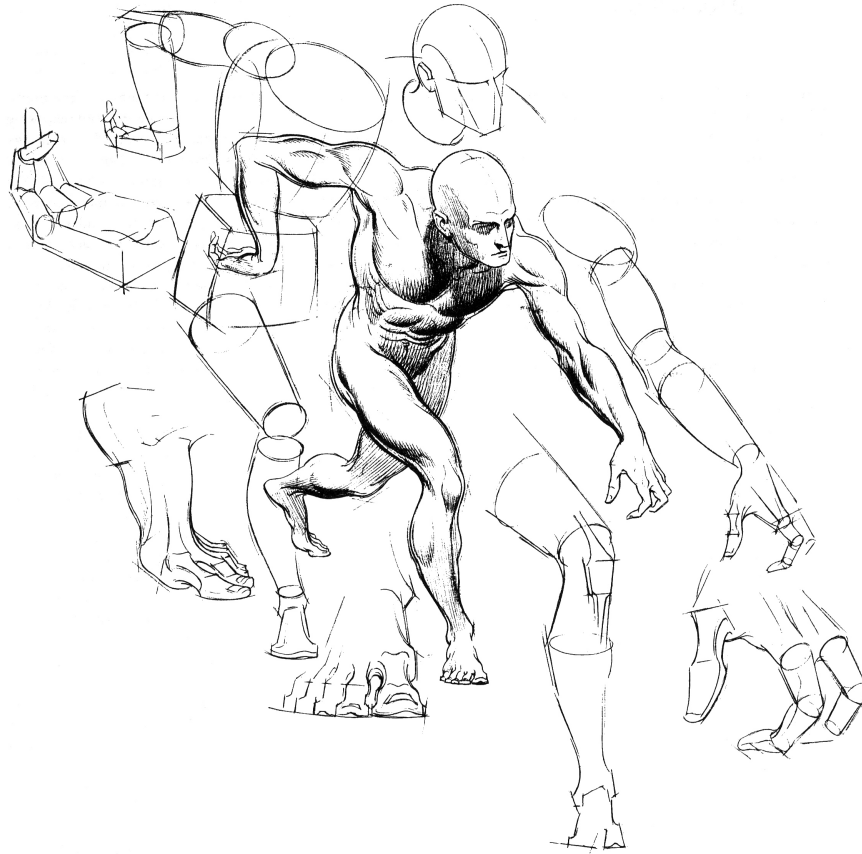
on that foundation to drive the creative process and produce a finished design. An example of this sort of deconstruction into primitives is depicted in Figure 6.15



**Figure 6.15.** This image from *The Simpsons* comically demonstrate the common technique used by artists to deconstruct a complex form such as the figure into an amalgamation of simpler basic shapes. In this case, Marge Simpson is practicing the “patented Lombardo Method” of visualizing her subject, here her husband Homer, in preparation for a painting.[Roberts & Reardon, 1991] (continued on page 322. . .)

To inform the design of the 2-D to 3-D conversion process, the focus was placed on how the artist conceives and draws these simplified shapes to construct his or her drawing. From the artist’s perspective the most visually striking characteristics of 3-D shapes are their outlines. This is why so many artistic depictions of objects or people are drawn with a solid black outline, even though in real life these outlines do not exist.

An artistic depiction of, for example, a simple box begins by drawing the shape of one of its ends. The artist draws a rectangular or square shape, but skewed to



**Figure 6.15.** (continued from page 321) A deconstruction of the human figure into solid forms by illustrator Burne Hogarth [Hogarth, 1996]. Although the shapes used to represent the components of the body are much more organic and irregular than the solid cylinders and blocks in Figure 6.15(b), each component is still composed of generalized versions of these basic shapes.

adjust for the 3-D position of the box in space. Next, the artist draws a series of extension lines. As outlines these extensions trace important visual features of the solid such as its silhouette or sharp edges on its surface. On a structural level, they indicate how the shape defining the end of the box is swept through space to form a 3-dimensional structure. Even though back-facing portions of the solid are occluded and out of view, the viewer extrapolates the most likely form as a simple extension of the end shape along the extension paths. This same

technique is the basis for drawing nearly any basic 3-D shape, from simple boxes cones and cylinders, to more complex organic variations of these simple forms like vases, tubes, torsos, leaves, or ribbons. Keep this example in mind as we cover the construction system in the following sections.

#### **6.4.11.3 Conversion System Interface**

The present application uses these same basic drawing steps as the basis for 3-dimensional construction. Within the system, the stroke defining the end shape that is to be swept through space is referred to as the *die stroke*, indicating its resemblance to an extrusion die. To construct the volume, the die stroke is swept along a path defined by the *path stroke*, which is drawn roughly perpendicular to the plane of the die stroke, and, just as the traditional artist would draw an extension outline, defines the path and extent of the sweep. Finally, the variation in size along the path stroke is derived from a second sweeping path called the *size stroke*. The variable distance between the path and size stroke define a dynamic alteration of the thickness or diameter of the 3-D shape, just as the two silhouette edges of a vase move in and out from its center to create its distinctive curves. These three components contain all of the information necessary for the application to generate a 3-dimensional model component. Based on one, two, or all three of these input sources, the system can generate three distinctive constructions, explained below.

The user indicates to the system how each stroke he or she creates should be interpreted by drawing each stroke with the appropriate version of the stroke pen tool. The current or active die stroke is the last stroke entered by the artist using the die pen, and is colored red in the modeling environment to indicate its

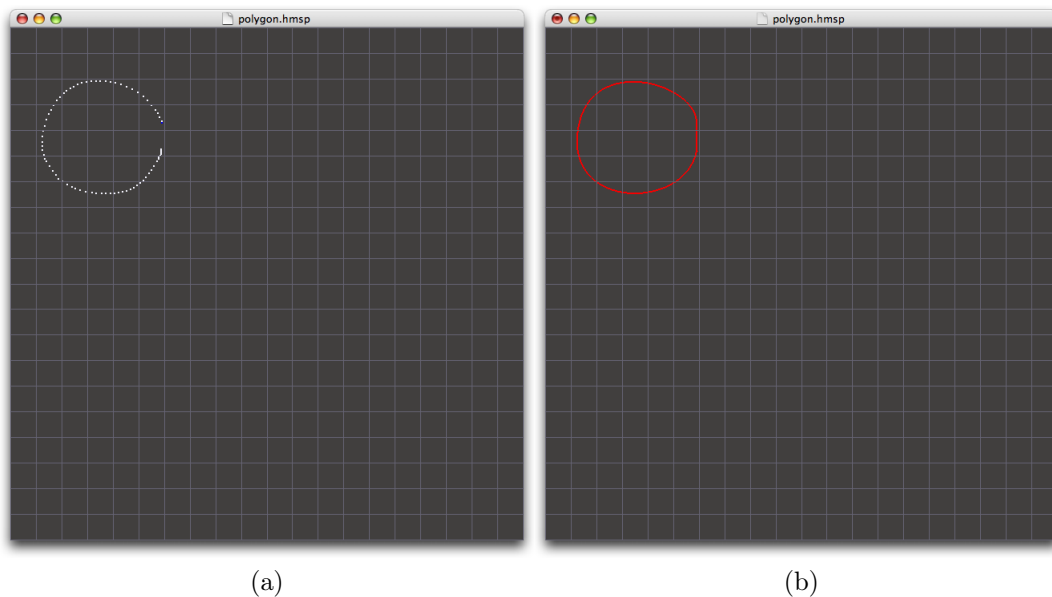
interpretation by the system. The active path stroke is similarly the last stroke generated with the path pen, and is colored green. Finally the active size stroke is created with the size pen, and is colored blue. Any strokes in the interface that are not marked with these colors will be ignored by the construction system.

Once the user has defined the appropriate input strokes, the conversion system is activated either through a keyboard command or a button on the tool palette. The system will process the input strokes and generate the appropriate kind of surface or volume. By default, volumes are rendered using a primary green color, however the user can select alternate hues using the inkwell widget on the tool palette.

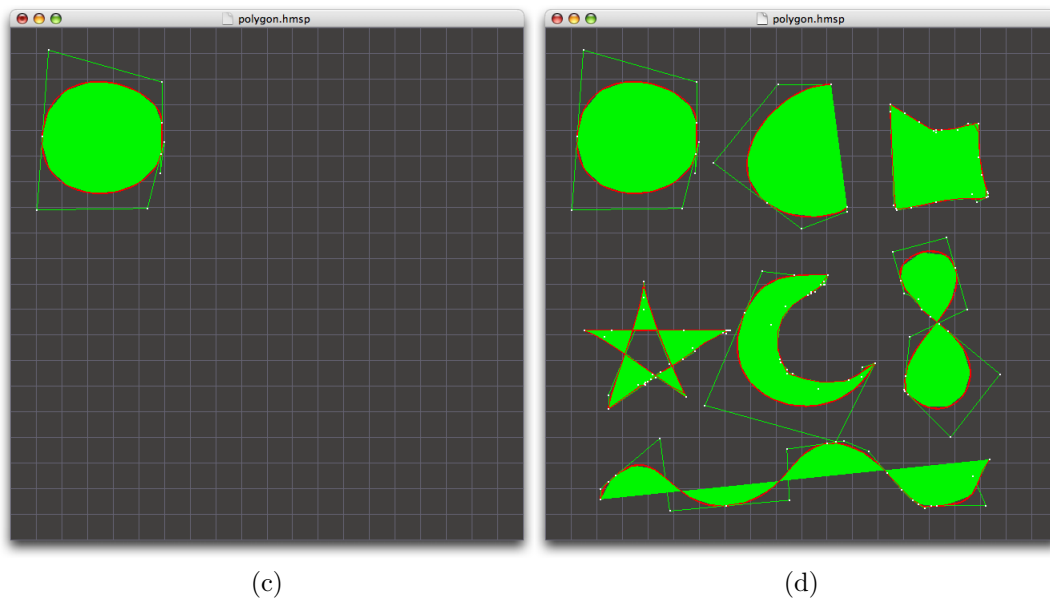
The first construction that can be formed by the conversion system is a simple 2-dimensional closed polygon. This model component is useful for placing solid caps at the ends of other components, or can be used to represent any generally flat shape. Because the polygon is 2-dimensional, only a single die stroke is necessary to define it. The polygon is constructed by connecting the beginning and ending points of the current die stroke, and then filling the closed region with a surface. In most cases constructed polygons are intended to be used with simple closed strokes. However, there is no restriction on the type of die stroke accepted by the system. Strokes that are not closed will be closed automatically by the algorithm. The system can also handle strokes with multiple self intersections or highly non-convex shapes.

To ensure that the closed region is rendered correctly by the system, rather than generating the surface from a single graphic primitive, the region is instead automatically tessellated, filling the concavity with a variable number of efficient triangles. This arrangement has several benefits. First, because only three points

define the triangles, each triangle is guaranteed to be planar. Second, by rendering the polygon as a set of triangles, the system is able to fill even the most oddly shaped region completely. This is not always possible with a single geometric primitive because primitives are required to be convex. These properties of planarity and convexity are important for many 3-D graphics architectures, including OpenGL, whose rendering algorithms require planar, convex polygons to function correctly. Examples of polygons generated with the system are visible in Figure 6.16.



**Figure 6.16.** Here we can see the construction of a 2-dimensional polygon. (a) The user begins by drawing a stroke with the digitizing tablet. The stroke is represented by a series of discrete white dots. (b) When the stylus is lifted, the dots are processed by the fitting system to generate a solid stroke, here colored red to indicate its interpretation as the active die stroke. In this case the user activated the `OPTION` modifier key, causing the stroke to be automatically closed. (continued on pg. 326 ...)



**Figure 6.16.** (continued from pg. 325) (c) Once created, the user activates the construction system, which creates a solid polygon filling the space of the user's stroke. In this view we can also see the control points and control polygon defining the chain of Bézier curves forming the perimeter of the stroke. (d) Using this method, a wide variety of polygon shapes can be created.

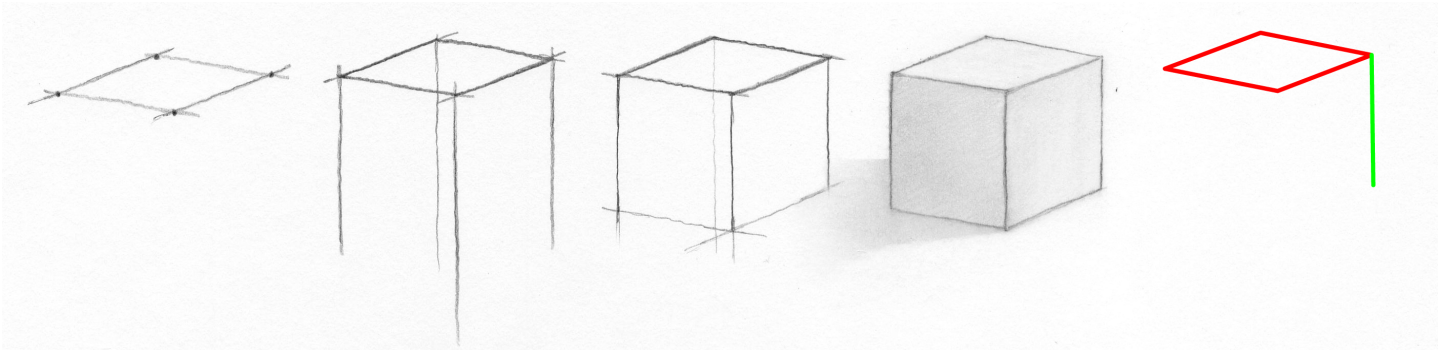
The second construction that can be formed by the conversion system is a generalized extrusion or sweep. For this volume, a contour shape is generated from the user's die stroke, which is evaluated at regular intervals along a course define by a path stroke. As the contour shape is replicated along the length of the path its orientation is also adjusted to match changes in the curvature of the path stroke. This has the effect of aligning each contour with the apparent orientation of the path stroke. Because no alteration of the die stroke's size is involved, only a die stroke and a path stroke are necessary to form this construction.

Under most circumstances, a path stroke entered by the user is drawn on a drawing plane roughly perpendicular to the plane of the die stroke, and the initial point of the path stroke is drawn to connect with an outer edge of the die shape.

Although the path stroke should usually be drawn in proximity to the die stroke, the exact position of its starting point can be varied to control the center point of contour orientations in relation to the original die shape. Drawing the path stroke from the perceived center of the die shape will also center orientating rotations as contours follow the path stroke. Drawing the path starting point to the edge of the die shape will instead center rotations to that location, enlarging or decreasing the radius of rotation depending on the direction of curvature.

This construction technique is designed to mimic the traditional practice of drawing a silhouette stroke extending from the edge of a face to define a 3-dimensional form. To understand how this works, let's consider our box example again—see Figure 6.17. As you will remember, to draw the box at an arbitrary angle a traditional sketch artist begins with the end shape, a rectangle, which is skewed and reoriented to reflect the off-angle view of face. The artist then draws a number of strokes extending the die shape through space to indicate its 3-D form. Although the number will vary with the particular end shape and viewing angle, for a box this is likely to include two silhouette strokes to either side of the end shape, and probably a stroke from the central corner of the rectangle to suggest the sharp edge of the box. After extending these three lines, the artist finishes by redrawing the visible image of the original end shape, indicating where the extension of the volume has stopped.

From a visual standpoint, all of these lines are necessary to fully define the shape of the box. However, in terms of geometric information, far less is actually needed. Once the shape of the end of the box is established, assuming that the shape is maintained through the entire volume unchanged, the only additional information necessary is to indicate the path along which the shape should pass



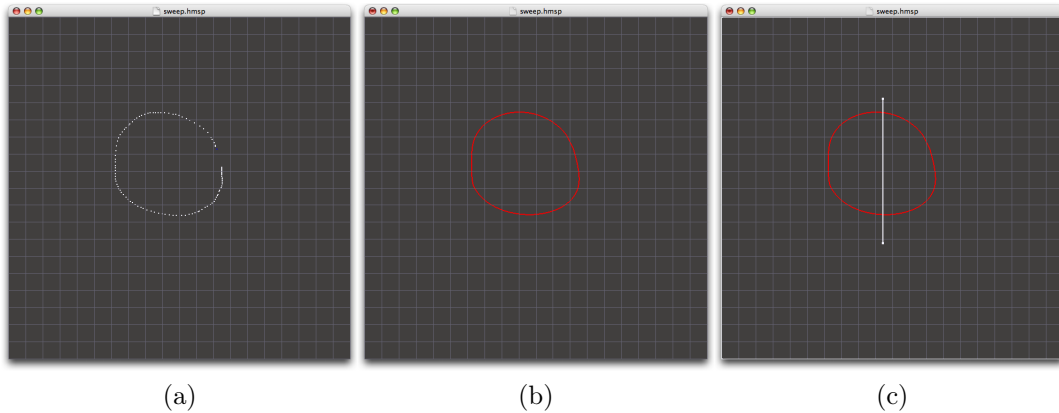
**Figure 6.17.** This figure demonstrates a common technique used by traditional artists to draw a box or any similar volume. Working from left to right, the artist begins with a skewed end shape and then extends that form through space using silhouette lines to define the entire volume. On the far right are the essential strokes defining this volume.

through space. This can be accomplished with a single path stroke, which indicates the translational aspect of the sweep, and where the sweep should stop. These two pieces of information, the end shape and the sweep path, are exactly represented by the die stroke and path stroke input to the conversion system.

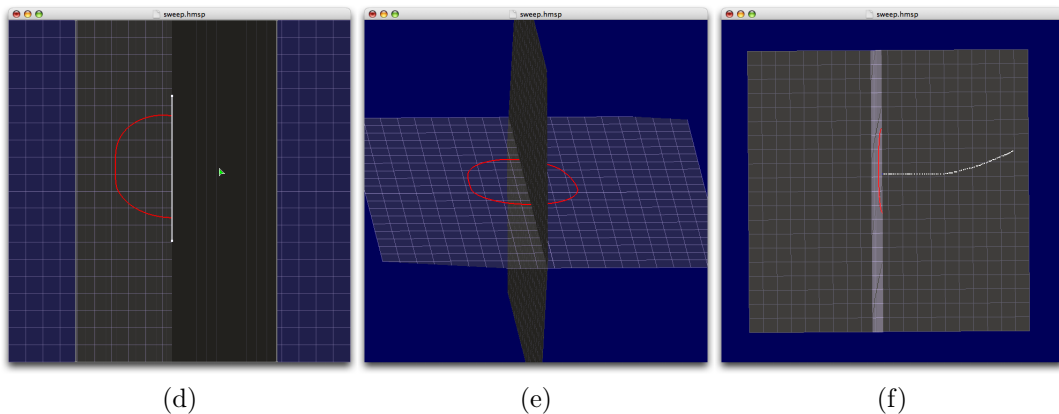
This construction method is useful for creating any number of 3-dimensional structures that maintain a consistent cross section along their entire length. When used with die strokes of standard closed 2-dimensional shapes like circles and rectangles, this can include such common 3-D forms as tubes, boxes, cylinders, and ducts. Because each contour is oriented to the path curve, the 3-D shape of the sweep does not collapse as the path stroke turns corners, but instead remains intact. This produces tube or pipe like structures that pass through both gentle curves and tight angles. Using unconnected die strokes, it is also possible to make sheets, ribbons, walls, hulls, flags, and other interesting shapes. Just like the polygon construction method above, any die stroke shape or path stroke shape that the user can draw will produce a 3-D structure. An example volume created



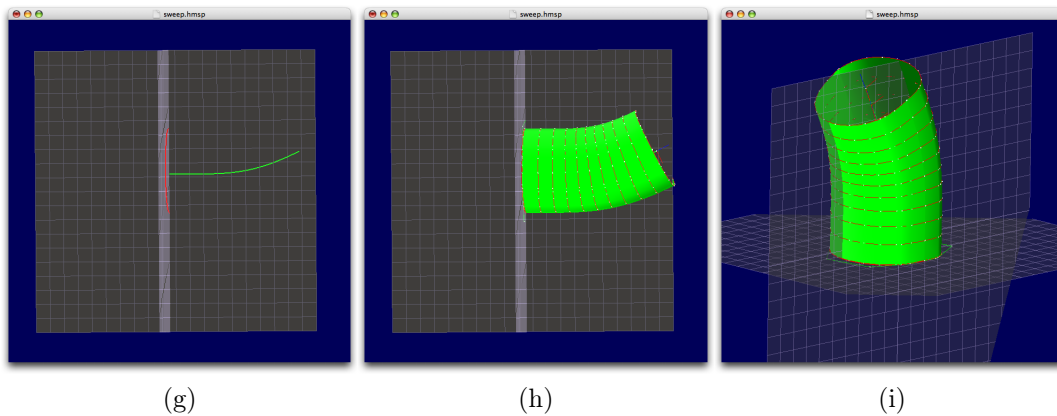
with this method is provided in Figure 6.18.



**Figure 6.18.** This series of images demonstrates the use of the sweep construction method. (a) The user begins, as before, by creating a die stroke, (b) which is colored red by the system. (c) A new plane is created by creasing the current plane with the crease tool. The crease is shown as a white line dragged out by the cursor. (continued on pg. 329 ...)



**Figure 6.18.** (continued from pg. 329) (d) The current plane is then rotated about this axis, creating a new plane and causing the old plane to become transparent. (e) Here we can see the relative positions of the two planes from a different angle. (f) The user next repositions the camera roughly parallel to the new plane, and draws an extrusion stroke. (continued on pg. 330 ...)



**Figure 6.18.** (continued from pg. 329) (g) The path stroke is colored green by the system. (h) The construction system is activated, and the original die stroke is swept along the curve to create a 3-D modeling component. (i) In this view, each contour curve oriented along the sweep path is also displayed in red. The final image depicts the resulting shape from an alternative angle. Looking inside, it's possible to see the extrusion path, which in this view is adorned with three-pronged Frenet triads indicating the local alignment along the curve at each position. There is one triad for each contour curve along the sweep.

It should be noted that with this construction method it is possible to create a 3-dimensional form that intersects itself. This occurrence can result from two situations. First, if the path stroke drawn by the user contains instances of self-intersection, then the 3-D model component generated from that path stroke will also self-intersect. This condition is fairly straightforward and easily identified by the user. The second condition occurs when the radius of a turn in the path stroke is smaller than the general size of the die stroke entered by the user. In this case, the contours generated from the die stroke will intersect along the region of the extrusion path where the curvature falls below the die stroke size.

In some modeling situations, such as the design of models for physical objects like consumer products or machine parts, models containing self-intersecting components are to be avoided. However, in other circumstances, such as the design

of character models or models for artistic use, self-intersection may not, in and of itself, be as much of a concern, or may in fact be what the artist desires. As a sketching application the purpose of this program is to faithfully represent the input provided by the user. Just as it is entirely possible to draw self-intersecting shapes with pencil and paper, the application makes no attempt to avoid or prevent the creation of self-intersecting geometry.

For the user, a basic understanding of the construction process should provide enough intuition about the behavior of the program to allow him or her to create self-intersecting forms when that is the desired result, and to avoid them otherwise. For the first type of self-intersection, it suffices to avoid using path strokes that self-intersect or pass within proximity of themselves. For the second type of self-intersection, the user can observe that the radius of curvature must be larger than the radius of a circle centered at the point where the path stroke begins, and extending large enough to encompass all portions of the die stroke in the direction of curvature. By adjusting where the path stroke begins in relation to the die stroke, the user can create very tight path curves that still avoid self-intersection. In any case, the system is designed to encourage experimentation and development. If a set of strokes is produced that generates a model with undesired self-intersections, it is a simple matter to undo the operation and try again.

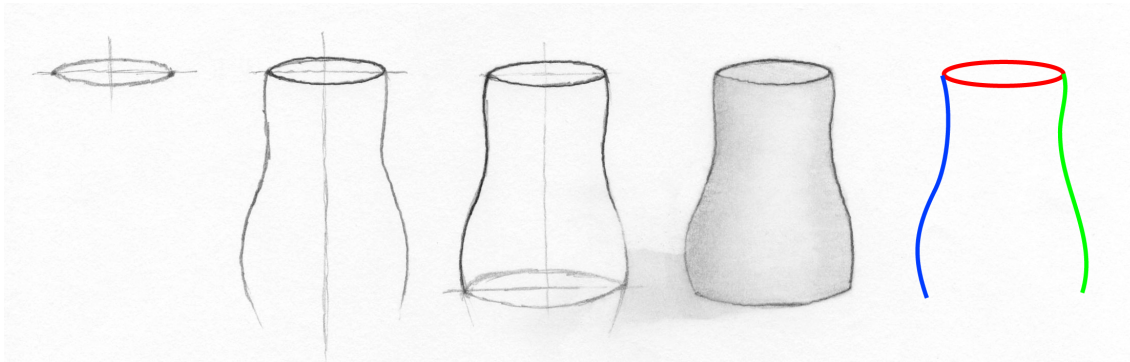
The third construction method available to the user is similar to the generalized sweep or extrusion, but offers an additional degree of expressivity by allowing the user to dynamically vary the size of the die shape as it's swept along the path stroke. Similar to the previous construction method, input consists of an arbitrary die stroke and a path stroke, usually drawn on a drawing plane roughly

perpendicular to the die stroke plane. In addition to these two strokes the user also enters a size stroke to indicate how the die shape should be scaled. The size stroke is also customarily drawn on a drawing plane roughly perpendicular to the die stroke; this can be a new plane, but it is often convenient to draw the size stroke on the same plane as the path stroke.

Just as the path stroke serves as a 3-dimensional analogue to an outline or silhouette stroke drawn in a traditional sketch, the size stroke also acts as a silhouette stroke, adding additional information to the silhouette provided by the path stroke. To understand how this works, this time lets consider a traditional sketch representation of a cylinder with a circular cross section. Just as in the case of the box shape, for a traditional sketch of this volume the artist would begin by defining a biased circle to represent the end shape, and then two silhouette lines, one to either side of the circle to indicate the straight sides of the cylinder. The drawing would then be completed with the final stroke to indicate the image of the end shape at the other end of the cylinder. Once again, as with the box example, from a geometric perspective only the end shape and a single sweep path are needed to fully describe this volume, and so the generalized sweep construction method is sufficient to describe the cylinder in its entirety with only a die and path stroke.

Now consider that the artist would like to draw a vase, again with a circular cross section, but this time featuring sides with rolling curves—see Figure 6.19. Although the shape is very similar, in this case the die stroke and path stroke do not provide enough information to correctly construct the volume. In addition to the path along which the die shape travels through space, we also need some indication as to how the die shape’s diameter changes along the sweeping path. In

a traditional sketch, this information is conveyed by the relative distance between the silhouette curves at either side of the end shape. As the curves move farther apart the diameter of the vase appears to increase, and as they move closer together the diameter diminishes. In the modeling system, this same information is expressed by the relationship between the path stroke and the size stroke, which act as analogues to the silhouette strokes of the traditional sketch.

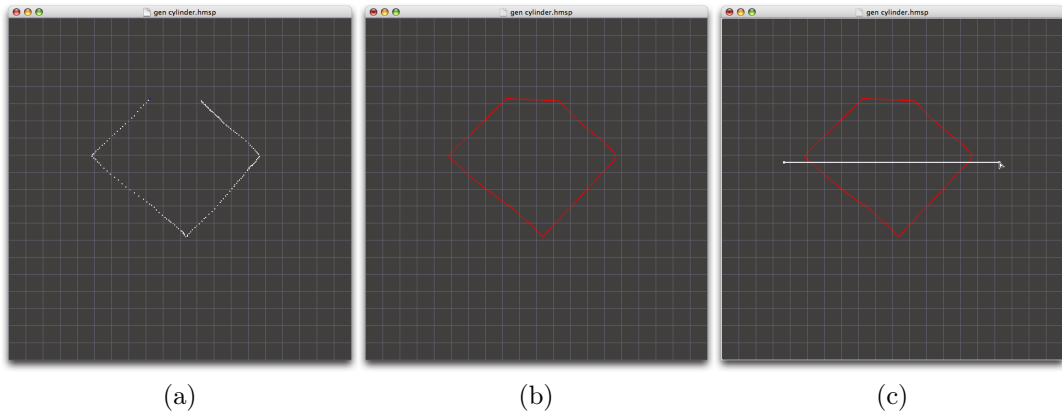


**Figure 6.19.** This figure demonstrates a common technique used by traditional artists to draw objects with a variable width. Again working from left to right, just like drawing a box the artist begins with a skewed end shape, which is then swept through space to define a 3-D volume. Notice how the change in the object's diameter is suggested by the relative distance of the silhouette strokes on either side. The final image shows the essential strokes defining this form. In this case, two lateral silhouette strokes are required to describe the changing shape.

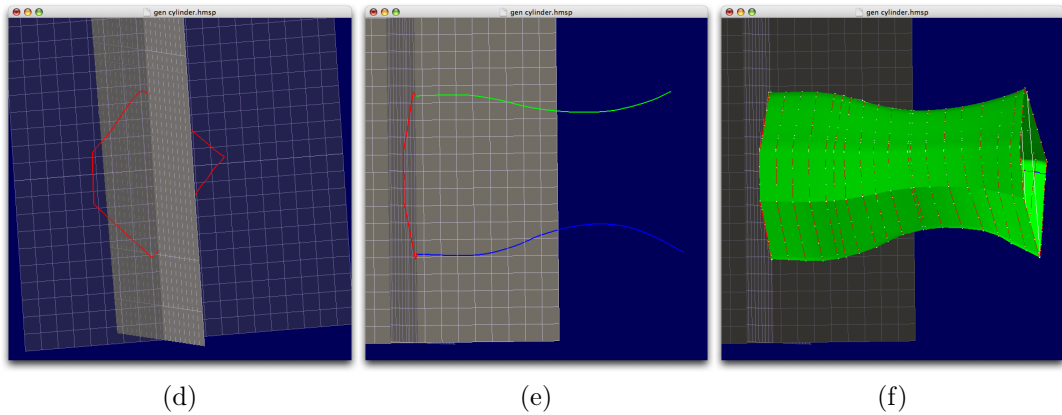
For this construction, rather than sweeping the die stroke shape along the path stroke directly, a new sweeping path is automatically generated by averaging together the path stroke and size stroke. The die stroke shape is then swept along this generated path. As contours are create from the die shape along the sweep, they are both oriented and scaled by comparing the path and size strokes at the corresponding parametric value. An orientation frame is generated from the vector perpendicular to the generated sweep path, and in the direction of a

line connecting the path and size strokes at the current parametric value. The scaling factor is determined as the ratio of the distance between the path and size strokes at the current parameter value to the same distance at the start of each stroke.

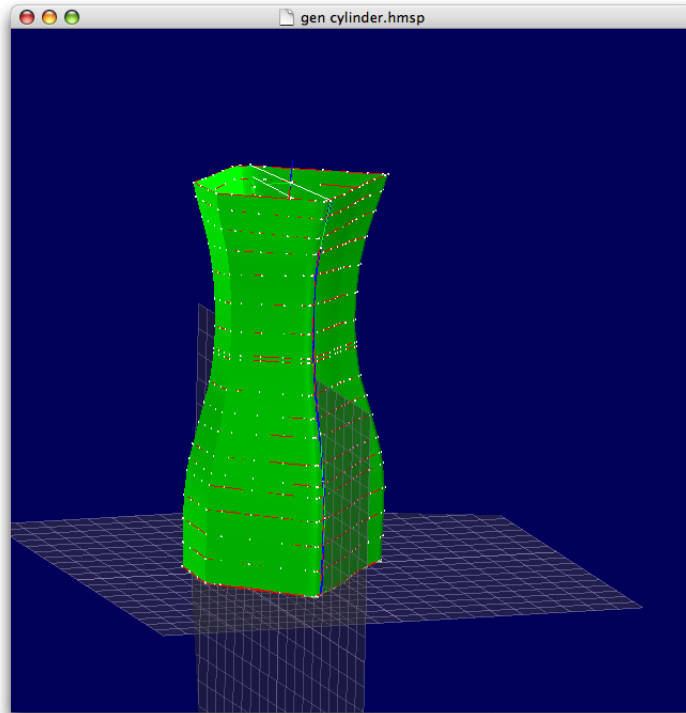
Using this method the user can create a wide variety of interesting shapes. With a basic closed die shape, common volumes like cones, pyramids, balloons, lampshades, dishes, and yes, vases can be drawn quickly and easily. With more open and abstract die strokes leaves, sword blades, fruit, fish bodies, character heads, plant stalks, beveled letters, mechanical parts, and any number of other interesting forms can be described. Although standard shapes are the easiest to think about, as with the other construction methods, there is no limit to the kinds of strokes that can be provided to the constriction system. The user is free to experiment. An example construction is given in Figure 6.20.



**Figure 6.20.** In this example we can see the construction of a modeling component using the generalized cylinder method. (a) As before, the construction begins with a die stroke. Here at each corner of the shape, among the white dots marking the path of the user's curve, blue point indicating that the system has detected a corner can be seen. (c) Just as with the sweep method, the next step is to create a new plane roughly perpendicular to the original that will contain further strokes defining the shape. (continued on pg. 335 ...)



**Figure 6.20.** (continued from pg. 335) (e) Once a new plane is created and the user's view adjusted, two additional strokes are drawn. One with the path pen, indicated in green, and one with the size pen, indicated in blue. Notice that although these strokes extend beyond the visible portion of the current plane, they continue to be projected onto its invisible surface. (f) Once these two strokes are defined, the construction method is activated, and a 3-dimensional modeling component is created. (continued on pg. 336 ...)



(g)

**Figure 6.20.** (continued from pg. 335) (g) An alternative view of the model. Looking at the top of the component, we can again see the alignment frame triads, as well as a series of white lines. These lines indicate matching positions along the path and size strokes that are used to derive scaling information for the contour curves. As with the triads, there is one line for each contour. Finally, notice that the sharp corner features in the original die stroke have been preserved to create a sharp edge in the final 3-D component.



# Chapter 7

## Implementation

This chapter describes the implementation details of the Hammerspace modeling application. This includes a general overview of the system architecture, as well as detailed descriptions of the data structures and algorithms used to create, represent, and display the system's features and components. We will begin with a general overview of the system architecture and codebase. The next section describes in detail the stroke system, including stroke representation, the stroke fitting process, and other algorithms used to process the user's drawing input. The following section gives an overview of the 3-D construction infrastructure, including a description of the internal representations of the 3-D models. Following that, the next two sections describe the sweep based and generalized cylinder based constructions systems respectively. Finally, the last section covers several topics related to the visualization of the 3-D models.

## 7.1 System Architecture and Codebase

As described in Section 6.4.1, the modeling application is designed to be used on a target platform running Apple's OS X operating system. OS X provides a number of application programming interfaces to allow developers to create native Macintosh applications quickly and easily. For the purposes of the present application, Apple's Cocoa API was selected. Cocoa—which was derived from NeXT Software's NeXTSTEP programming environment—provides access to a full set of libraries and tools to generate a GUI based application. Cocoa and its associated libraries and frameworks are based on the Objective-C programming language, a Smalltalk like object-oriented extension of the C programming language.

The selection of the Cocoa API for this project was based on two factors. First, because Cocoa is the *de-facto* standard for new application development on OS X, the Apple development tools are designed specifically to work with Objective-C and the Cocoa API's. Using these tools, it was possible to quickly generate the basic user interface elements required by the program, and to integrate the program's features into standard application tasks like saving and loading files. Second, because Objective-C is a true super-set of the C language, working in Objective-C provides direct access to a large number of standard C-based UNIX libraries and tools.

Because Cocoa can interact with standard libraries, it was possible to utilize the OpenGL 3-D graphics API, a standard C-based 3-D graphics API, which was used for processing and display of the 3-D components of the system. This particular API was selected for three reasons. First, OpenGL is an open cross-platform standard, meaning that graphics code created for this project should be portable to other system. Second, OpenGL's open nature and portability have

fostered a strong and active developer community and many print and on-line resources to aid in the development process. And finally, OpenGL is the graphics API with which the author is most familiar.

One drawback to the Cocoa programming environment is that code generated to work with Cocoa and OS X, unlike OpenGL components, is not in general portable. Although the current application is targeted specifically at the Apple platform, it was desirable to generate a codebase that could, in the future, be transferred to other platforms with a minimum of retooling. To achieve this goal, an effort was made to limit Cocoa and Objective-C specific code to the user and system interface components of the program. At the same time, all of the data structures, algorithms, internal and back-end components of the application were written in portable C++.

To bridge the gap between this Objective-C front end and the C++ back-end, the GNU compiler collection provides an Objective-C++ compatibility mode, allowing Objective-C and C++ code to be mixed within the same source file (observing a number of restrictions—see Note 3. Because both Objective-C and C++ share the same C foundations, this allows data to pass back and forth between the two parts of the system seamlessly. However, every effort has been made to limit the degree to which the Objective-C and C++ code intermingle. This ensures that if the application were to be ported to another environment, only the user interface portions of the program would need to be rewritten.

The overall architecture of the application follows the Model-View-Controller design pattern. This design not only facilitates the separation between the Objective-C interface components and C++ back-end, but is also the preferred design pattern for document-based applications under Cocoa. By organizing the program

---

**Note 3** Objective-C++

---

Generally, the Objective-C and C++ components interact on the level of their shared C heritage. Thus, all of the primitive types—int, float, double, char, etc.—are identically stored and manipulated by either language. By the same token, pointers, both to structures and C++ or Objective-C classes can be passed around with ease. Method calls to C++ and Objective-C objects can be freely intermixed in the code, even within the same compound statement. However, the structure of class hierarchies and the semantics of calling conventions between the two language’s class systems cannot be intermingled. C++ pointers cannot respond to Objective-C messages and vice versa. Neither can classes from one language inherit or derive from classes of the other. Objects from one language can serve as member variables of a class in the other language, however they must be represented as pointers in the class description and manually instantiated and de-allocated in a constructor and destructor respectively. For more information, see Apple’s Technical Documentation on Objective-C++ [Apple, 2001].

---

into controller, view, and model components, it is easier to utilize the user interface design, event loop, and document architecture of the Cocoa API. A diagram of the system’s components is provided in Figure 7.1.

The view portion of the program is primarily represented by the `OpenGLView` class, which represents the main document window. The `OpenGLView` is responsible for interacting directly with the OpenGL subsystem. The `OpenGLView` initializes the viewing context, adjusts the viewport when the window is resigned or its shape is changed, and orchestrates the process of drawing the user’s view of the 3-D scene. The `OpenGLView` is also the direct recipient of all mouse, keyboard, and tablet events directed at the document window. However, although the view receives these events from the operating system, they are not processed directly, but are instead passed along to the controller portion of the program.

Because the `OpenGLView` represents the user interface, and processes events and signals generated by the system, its codebase is written in Objective-C so

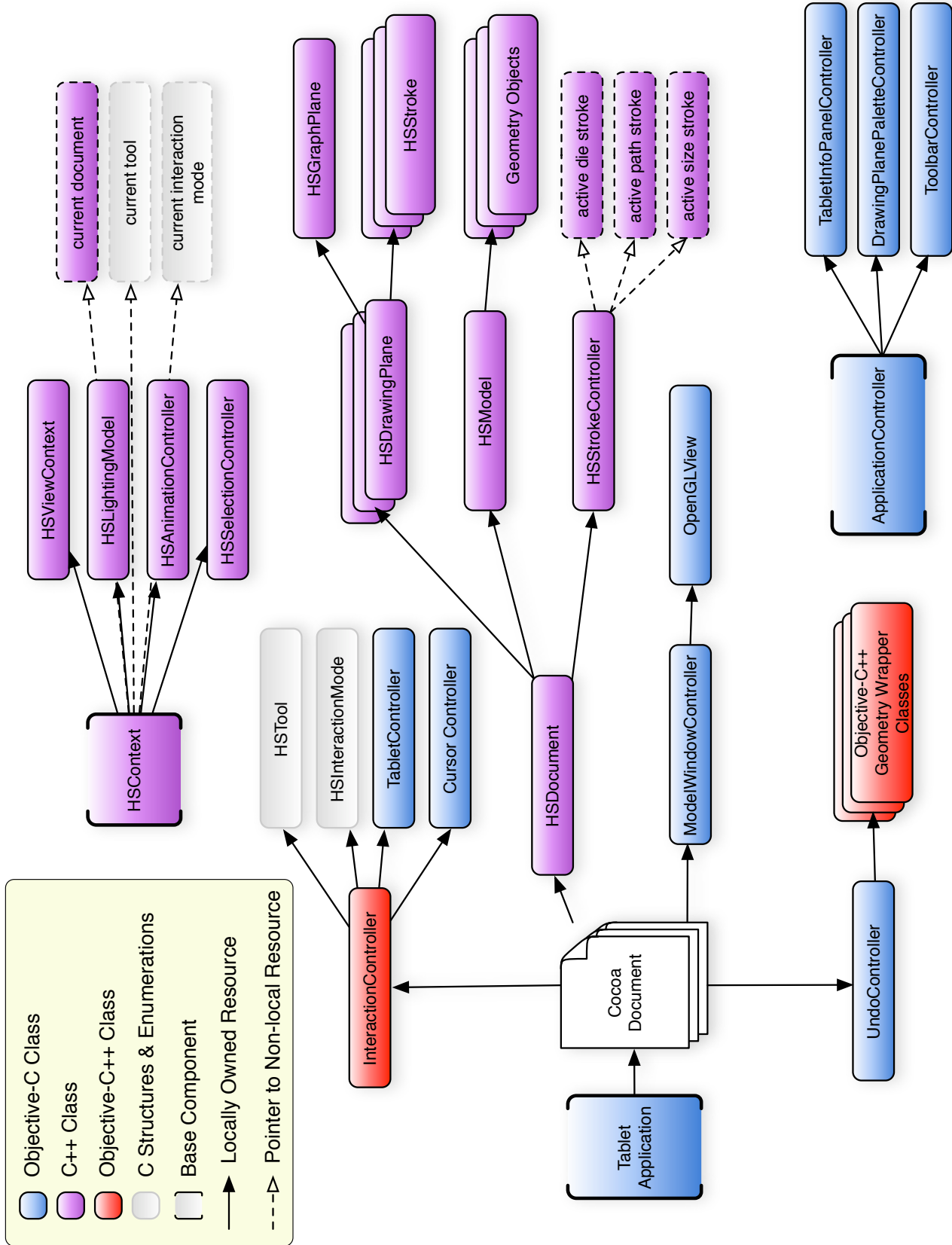


Figure 7.1. Application Architecture Diagram

that it can interact with the system libraries. Under normal circumstances, this might also mean that the OpenGLView is the least portable portion of the code. However, because the viewing commands and components are basically wrappers around the portable OpenGL API, and because system input like mouse, keyboard, and tablet events are quickly passed to other portions of the code, although not directly portable the structure of the OpenGLView's code should make the porting process straightforward.

The InteractionController represents the primary component of the controller portion of the system architecture. When user events are generated, after being caught by the OpenGLView or other GUI elements, they are passed to the InteractionController where they are processed. Based on contextual information stored in the InteractionController and elsewhere in the program such as the current tool or types of active strokes, the InteractionController analyzes each event, and passes it to more specialized classes and structures throughout the program to be interpreted. Thus, the controller forms a bridge between the interface and the back-end components where user input is interpreted and then forwarded to make changes in the internal structure of the program and its data.

The InteractionController is a jumping off point for many other portions of the program including the curve fitting system, the tablet gesture systems, and the 2-D to 3-D conversion system, just to name a few. Because the controller must interact with both the Objective-C based interface, and the C++ based back-end components, it is written in a mix of Objective-C and C++ and makes extensive use of GCC's Objective-C++ capabilities. Because of this tight coupling, the InteractionController itself is probably the least portable component of the system. However, in order to minimize the amount of bridging code, whenever possible

controller functionality that only deals with one side of the application architecture is spun off into separate classes that are more reusable, or manageable and isolated targets for porting.

Representing the persistent data portions of the application, including all of the modeling information generated by the user is the HSDocument class. This class is composed primarily of the list of drawing planes, each of which contains the information defining that drawing plane and any strokes drawn on its surface, and a list of geometry objects that compose the user's 3-D model. These structures are in turn composed of a wide variety of smaller more basic classes defining everything from curve segments and grid lines to mesh components and camera structures.

All of these classes are derived from a small set of abstract bases that provide a common interface to the drawing and I/O routines. Each class is responsible for performing the OpenGL drawing commands needed to render itself, and for reading and writing its own persistent data when files are opened or saved. Thus, when a draw or save command is issued, the instruction originates from one of the interface classes and cascades down through the structure of the program to the actual data at its extremities.

All of the model classes are written in pure C++ and are completely portable. The only exception is at the HSDocument level. Here, in order to integrate the model into the interface components, the HSDocument class is thinly wrapped in an Objective-C CocoaDocument class, allowing the model to participate in Cocoa's document architecture.

## 7.2 Strokes

As discussed in Section 6.4.6, as points are generated by the digitizing tablet and collected into the application they form a polyline, a structure of sequential points connected by straight line segments. Although the polyline structure faithfully represents the samples generated by the digitizing tablet, the polyline is both inefficient, and an inaccurate representation of the path of the user's pen. Because the tablet generates samples at a roughly constant rate, areas of the stroke where the user's hand has lingered are over sampled, containing dozens of virtually coincident points. Moreover, in areas where the user's hand was more animated, the sampling rate of the tablet may have been too low, degenerating smooth sweeping curves into jagged, corner-cut shapes. To more accurately and efficiently represent the user's intended stroke, the raw sample points of the polyline undergo a three-stage process of *filtration*, *classification*, and *conversion*.

### 7.2.1 Filtration

The first step of the process is filtration. As the user moves the stylus across the tablet's surface, the tablet hardware generates somewhere on the order of 100 sample points every second. Obviously because of the high granularity of the digitizing tablet, the raw polyline generated by the application contains a large number of redundant points. Thus, before any further processing of the raw stroke data is performed, a simple distance-based filtration process is applied to each point as it comes in.

For each new sample point, its location in screen coordinates is compared to the most recent previous point not already filtered. A Euclidean distance is calculated, and if the new point is less than two pixel's distance from the previous,



the point is excluded. This filtration process is performed in the 2-dimensional screen coordinate system so that the filtration process is independent of any scaling or zoom in the user's view of the modeling environment. The only exceptions to the filtering process are the initial and final points. So that the stroke exactly interpolates the start and finish of the user's stroke, these points are not filtered.

Points that pass successfully through this initial filtration are then projected into the modeling environment onto a drawing surface, a process that generates a new 3-dimensional set of coordinates for the point, this time in world space. The projection process is achieved by generating a ray starting at the point on the near clipping plane of the modeling environment and directed through the corresponding point on the back clipping plane. This ray is intersected with the plane representing the current drawing surface to find the stroke point's 3-dimensional location. The formulation is based on examples by Dunn & Parberry [Dunn & Parberry, 2002], and is calculated as follows. The ray is represented by the parametric equation,

$$r(t) = q_0 + t\hat{v} \tag{7.1}$$

where  $q_0$  is the point on the near clipping plane, and  $\hat{v}$  the unit vector describing the ray's direction. The drawing surface is represented by the implicit definition of the infinite plane,

$$p \cdot \hat{n} = d \tag{7.2}$$

where  $p$  is a point in the plane,  $\hat{n}$  the plane's normal vector, and  $d$  is the distance from the origin to the nearest point on the plane. To determine the point where the ray intersects the plane, first the dot product of the plane's normal vector  $\hat{n}$  and the ray direction vector  $\hat{v}$  is taken. If the dot product is zero then the ray

is parallel to the plane and no intersection has occurred. If not, we can solve for the parametric value at which the ray strikes the plane by substituting the ray equation into the plane's implicit formula.

$$\begin{aligned}
 (q_0 + t\hat{v}) \cdot \hat{n} &= d \\
 q_0 \cdot \hat{n} + t\hat{v} \cdot \hat{n} &= d \\
 t\hat{v} \cdot \hat{n} &= d - p_0 \cdot \hat{n} \\
 t &= \frac{d - p_0 \cdot \hat{n}}{\hat{v} \cdot \hat{n}} \tag{7.3}
 \end{aligned}$$

The three dimensional location of the stroke point is then simply found by plugging this parametric value into the original ray equation.

### 7.2.2 Classification

At the next stage, each point that has survived the filtration process is classified as either a *corner* or *non-corner* point. This classification process is necessary because the next stage will remove any sharp corners or discontinuities currently present in the polyline representation. For the most part, corners and angles currently present in the polyline representation are unwanted, and will be smoothed over as gentle curves. However, the design specifications indicate that the system should be capable of generating models with both smooth and angular features. Because the 3-D geometry will be generated from 2-D strokes, it is necessary to preserve the discontinuities in the user's stroke that represent intentional sharp edges.

The classification of each point is determined by comparing the angle between two vectors, one calculated from the previous point to the current point, and one

from the current point to the following point. If the angle between these vectors is below a threshold then the point is classified as a corner. Corner points are used as boundaries in the fitting process to indicate the divisions between contiguous blocks of sample points that should be smoothly fit. Fitting operations are not performed over a range containing internal corner points, and so corners defined by these points are preserved in the final stroke.

The classification process, like filtration, is performed as points are still coming in to the system. However because an additional point beyond the classification target is needed in order to calculate the two classifying vectors, each point is actually classified after its following point is filtered. The two exceptions to this are the initial and final points of the user's stroke, which are always classified as corners.

### **7.2.3 Conversion**

After filtration and classification, the resulting polyline is passed to the curve fitting system. The filtered sample points representing the curve are grouped into contiguous sets of non-corner points, and each set is then in turn converted to a parametric representation as a chain of cubic Bézier curves. The following sections describe this parametric representation, and the curve fitting process used to perform the conversion.

#### **7.2.3.1 Representation**

As noted in Section 5.1, a Bézier curve is an efficient and stable parametric method of representing polynomial curves of arbitrary degree, developed independently by Paul de Casteljaou and Pierre Bézier [Farin, 2002]. Bézier curves have a

number of properties that make them an attractive curve representation for this application. A Bézier curve of degree  $n$  is represented by  $n + 1$  control points, which act as constraints on the path of the curve. The curve exactly interpolates the first and last of these control points, and the path of the curve fits completely within the control polygon formed by connecting the control points with straight lines.

Mathematically, the Bézier curve  $C$  is represented by the parametric equation,

$$C(t) = \sum_{i=0}^n P_i B_i^n(t), \quad t \in [0, 1] \quad (7.4)$$

where the degree of the curve is  $n$ , there are  $n + 1$  control points  $P_0, \dots, P_n$ , and  $B_i^n$  is the  $i^{\text{th}}$  Bézier basis function of degree  $n$ , expressed as,

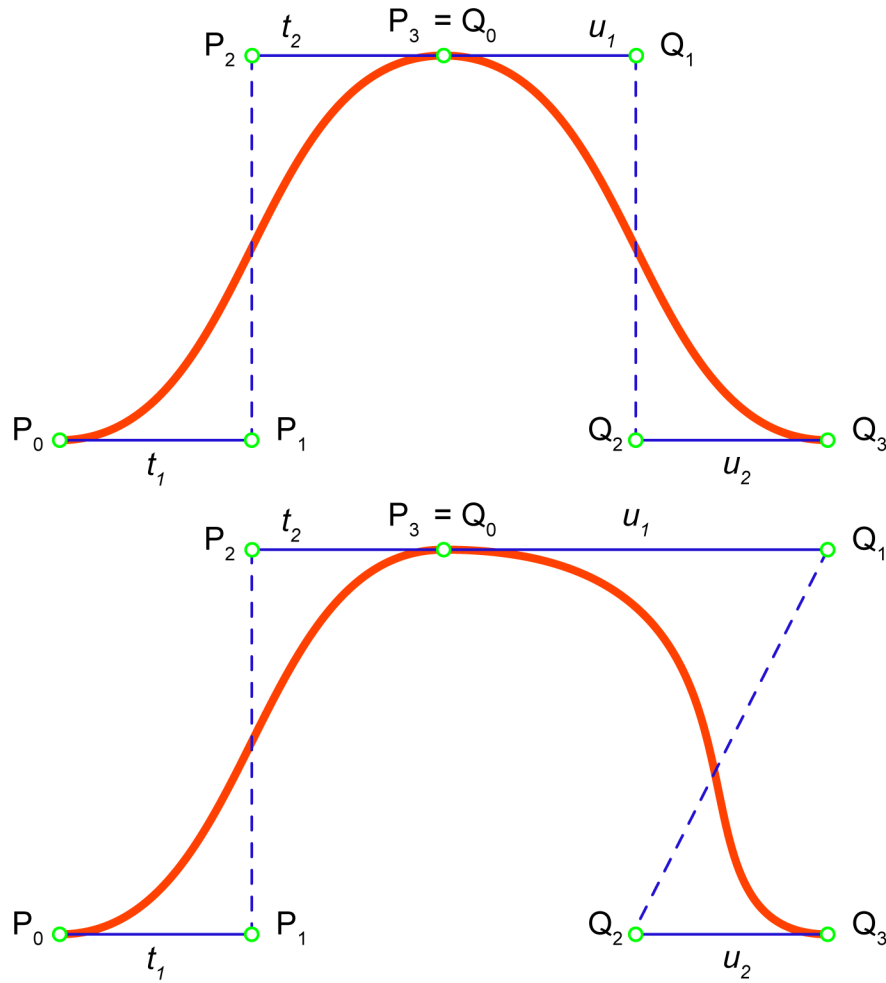
$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}, \quad i = 0 \dots n \quad (7.5)$$

In general, it is not possible to represent a single stroke entered by the user as a single Bézier curve. This is because the user's stroke contains too much information to be faithfully represented by a curve of such low degree. The Bézier curve representation is capable of representing parametric curves of virtually any degree, however as the degree of the curve increases certain properties of the curve make the representation undesirable. Curves with a high degree are inefficient to evaluate using the standard Bézier curve algorithms, and may lead to numerical instability. Furthermore, as the number of constraints on the curve increases it becomes more and more difficult to define a curve that is visually smooth. Rather than using a single curve of high degree, in the present application the stroke is instead represented as a chain cubic Bézier curves.

The chain representation provides several attractive properties. First, each segment of the chain is represented as a single cubic Bézier curve. The cubic representation strikes a good balance between expressiveness and efficiency. In addition, cubic Bézier curves are constrained by four control points, one at each end of the curve segment, and two internal to the curve. The first and last legs of the control polygon drawn between the first and second control points and third and fourth control points respectively also represent the directions and magnitudes of the tangent vectors to the Bézier curve at its beginning and end—see Figure 7.2. These properties make the curve easier to conceptualize, and many vector graphic and modeling applications such as Adobe’s Illustrator [Adobe, 2007a] or Macromedia’s FreeHand [Macromedia, 2005] use these tangent vectors and the control points that define them as handles for interactive manipulation of curve segments. Although this interaction mechanism is available, it does not correspond to the sketching process and so is not used in the application. However, familiarity with this means of visualizing the curves makes it possible to visually inspect the results of curve generation during the development process.

In order to represent the user’s entire stroke, a number of cubic Bézier segments are connected together to form the chain. The chain is constructed so that each Bézier segment shares its initial and final control points with the previous and succeeding curve segment respectively. In order that the curve segments connect together to form a single seamless stroke, it is also necessary to place constraints between neighboring curve segments to ensure that they connect with the necessary degree of *continuity*.

Continuity between curve segments is described mathematically by the related concepts of parametric and geometric continuity [Farin, 2002]. Two Bézier curves



**Figure 7.2.** The red curve in the top figure is composed of two cubic Bézier curve segments. The segments share a single control point, indicated alternately as  $P_3$  and  $Q_0$ . Looking at the final leg of the control polygon of the first curve segment between control points  $P_2$  and  $P_3$ , and the initial leg of the second between control points  $Q_0$  and  $Q_1$ , we can see that they share the same direction, and equal lengths. This means that these two curve segments meet with  $C^1$  parametric continuity. Compare this arrangement to the lower diagram. Here, the second control point of the second curve segment has been repositioned, such that the initial leg of its control polygon points in the same direction, but is now longer than the final leg of the previous segment. Despite this change, the two curve segments still meet at a visually smooth junction, although the two sides of the bell shape are now no longer identically curved. Because of this change, the lower curves are now no longer  $C^1$  continuous, but are  $G^1$  geometrically continuous.

are said to meet with  $C^0$  *parametric continuity* if the initial control point of the second curve is coincident with the final control point of the first.  $C^0$  parametric continuity is sufficient to define continuity between curves on either side of a corner point, however for smooth sections of a stroke further constraint is necessary.

Two Bézier curves are said to meet with  $C^1$  parametric continuity if they are  $C^0$  continuous, and furthermore, the magnitude and direction of the tangent vector of the first curve at its final point is equal to the initial tangent vector of the second curve. Although higher degrees of parametric continuity are possible, from a general standpoint,  $C^1$  continuity is sufficient constraint to describe visually seamless transition from one curve segment to the next.

Mathematically, first degree parametric continuity ensures that as the parametric variable sweeps out the Bézier curve, not only is the path positionally continuous, but so too is its first derivative. Visualized in another way, we can say that for any given constant delta of the parametric parameter to the curve, the speed of a particle moving along that curve will also be continuous, and will have finite acceleration over the entire parametric range.

Although this constraint produces smoothly connected curve segments, it is important to remember the physical activity the stroke is intended to represent. As the artist's hand produces a stroke on the tablet it is not necessarily the case that the velocity of the stylus is continuous. The user may suddenly accelerate, stop in mid stroke and resume, or otherwise manipulate the drawing tool in a discontinuous motion. According the design principle I, the system should strive to replicate the physical activity of drawing. Thus constraining strokes to  $C^1$  parametric continuity is inappropriate.

Although parametric continuity can be described in geometric terms, the con-

straints it applies to adjoining curves are analytical, expressing a relationship between  $n^{\text{th}}$  derivatives of each curve at a given point. Alternatively, *geometric continuity* focuses more on the visual smoothness of a series of curves through geometric constraints alone. Whereas  $C^1$  continuity requires the terminal and initial tangent vectors of two adjoining curve segments to be equal in both direction and magnitude,  $G^1$  geometric continuity requires only that the tangent vectors have the same direction. Visually, we can imagine two curves forming a rounded hill, with the point of connection between the two segments at the crest of the hill—refer again to Figure 7.2. If the two curves meet with  $C^1$  continuity, then the apparent steepness on either side of the hill must be the same, a reflection of the matching magnitudes of the first derivatives of each curve. If however the curves meet with  $G^1$  continuity, then the far side of the hill can have a much gentler slope, a reflection of the larger magnitude of its initial tangent vector. This less stringent constraint allows a series of curve segments to more faithfully represent the physical source of stroke creation.

The application utilizes an elegant least-squares based divide-and-conquer curve-fitting algorithm to convert the raw sample points into a chain of Bézier curves that approximate their path. Contiguous segments of non-corner points are input to the algorithm where they are converted into a series of one or more Bézier curve segments meeting with  $G^1$  geometric continuity. Each of these smooth sections is joined together at corner points with  $C^0$  positional continuity to create a final, fully connected stroke.



### 7.2.3.2 Stroke Fitting

The algorithm used to fit Bézier curve segments to the raw sample points is based on a method originally described by Philip Schneider [Schneider, 1990]. The basic idea of the algorithm is this. Initially, a least-squares method is used to generate a single cubic Bézier curve approximating the path of a set of arbitrarily many sample points. Next, for each sample point, the closest point on the newly generated curve is generated, and the distances between each sample and its corresponding curve point calculated. If all of the distances are below a fitting threshold, then the curve is considered to be a good fit, and accepted. If any one of the sample point distances is larger than the threshold, then the set of sample points is divided in two, broken at the sample point with greatest distance from its corresponding curve point. Each of these sets is then recursively fit using the same process. Recursive subdivisions of the sample point sets continues until the paths of each generated curve are within the distance tolerance of their sample points, or the sample point sets are reduced to two points, or become nearly linear, in which case a linear curve is used to fit them.

The first step of the process is to generate a cubic Bézier curve that approximates the path of a given set of sample points. If we represent the Bézier curve as in formula 7.4, then we would like to minimize the sum of the squares of the difference between each 3-dimensional sample point  $d_i$ , and the corresponding point on the Bézier curve  $C(t_i)$ ,

$$S = \sum_{i=1}^n [d_i - C(t_i)]^2 \quad (7.6)$$

$$= \sum_{i=1}^n [d_i - C(t_i)] \cdot [d_i - C(t_i)] \quad (7.7)$$

Considering the Bézier representation of a cubic curve, the curve is constrained by four control points,  $P_0$ ,  $P_1$ ,  $P_2$ , and  $P_3$ . Because the curve must exactly interpolate its initial and final points, we know that control points  $P_0$  and  $P_3$  are simply equal to the initial and final sample points. Furthermore, in order for the curve to maintain  $G^1$  geometric continuity with its neighboring curves, we know that the initial and final tangent vectors of the curve must have the same direction as the corresponding tangent vectors of their neighbors. If we denote these initial and final tangent vectors as  $\vec{v}_1$  and  $\vec{v}_2$ , then we know that  $\vec{v}_1 = \alpha_1 \hat{u}_1$  and  $\vec{v}_2 = \alpha_2 \hat{u}_2$  where  $\hat{u}_1$  and  $\hat{u}_2$  are unit vectors in the direction of the of the constrained tangent vectors, and  $\alpha_1$  and  $\alpha_2$  are unknown scaling factors.

In order to generate the cubic curve, we need to find values for  $\alpha_1$  and  $\alpha_2$  that minimize the sum of squares. This is achieved by taking the first derivative with respect to  $\alpha_1$  and  $\alpha_2$  of the sum above, and setting the result to zero.

$$\frac{\delta S}{\delta \alpha_1} = 0, \quad \frac{\delta S}{\delta \alpha_2} = 0 \quad (7.8)$$

Beginning with  $\alpha_1$ , we can expand this equation to,

$$\frac{\delta S}{\delta \alpha_1} = \sum_{i=1}^n 2[d_i - C(t_i)] \cdot \frac{\delta C(t_i)}{\delta \alpha_1} \quad (7.9)$$

where,

$$\begin{aligned} \frac{\delta C(t_i)}{\delta \alpha_1} &= \frac{\delta}{\delta \alpha_1} (P_0 B_0^3(t_i) + (\alpha_1 \vec{u}_1 + P_0) B_1^3(t_i) \\ &\quad + (\alpha_2 \vec{u}_2 + P_2) B_2^3(t_i) + P_3 B_3^3(t_i)) \\ &= \vec{u}_1 B_1^3(t_i) \end{aligned} \quad (7.10)$$

Filling this result into the original, we can simplify and rearrange to,

$$\frac{\delta S}{\delta \alpha_1} = \sum_{i=1}^n 2[d_i - C(t_i)] \cdot \vec{u}_1 B_1^3(t_i) = 0 \quad (7.11)$$

$$= \sum_{i=1}^n B_1^3(t_i) C(t_i) \cdot \vec{u}_1 = \sum_{i=1}^n \vec{u}_1 B_i^3(t_i) \cdot d_i \quad (7.12)$$

Defining  $A_{i,1} = \vec{u}_1 B_i^3(t_i)$ , this can be rewritten more clearly as,

$$\sum_{i=1}^n C(t_i) \cdot A_{i,1} = \sum_{i=1}^n d_i \cdot A_{i,1} \quad (7.13)$$

Expanding the  $C(t_i)$  term, we get,

$$\begin{aligned} & \sum_{i=1}^n C(t_i) \cdot A_{i,1} \\ &= \sum_{i=1}^n C(t_i) \cdot (P_0 B_0^3(t_i) + \alpha_i A(i, 1) \\ & \quad + P_0 B_1^3(t_i) + \alpha_2 A_{i,2} + P_3 B_2^3(t_i) + P_3 B_3^3(t_i)) \\ &= \sum_{i=1}^n A_{i,1} \cdot P_0 B_0^3(t_i) + \alpha \sum_{i=1}^n A_{i,1}^2 + \sum_{i=1}^n A_{i,1} \cdot P_0 B_1^3(t_i) \\ & \quad + \alpha_2 \sum_{i=1}^n A_{i,1} \cdot A_{i,2} + \sum_{i=1}^n A_{i,1} \cdot P_3 B_2^3(t_i) + \sum_{i=1}^n A_{i,1} \cdot P_0 B_3^3 t_i \end{aligned} \quad (7.14)$$

This expression replaces the original in equation 7.13, giving,

$$\begin{aligned} & \left( \sum_{i=1}^n A_{i,1}^2 \right) \alpha_1 + \left( \sum_{i=1}^n A_{i,1} \cdot A_{i,2} \right) \alpha_2 \\ &= \sum_{i=1}^n (d_i - (P_0 B_0^3(t_i) + P_0 B_1^3(t_i) + P_3 B_2^3(t_i) + P_3 B_3^3(t_i))) \cdot A_{i,1} \quad (7.15) \end{aligned}$$

Following a similar process, we can derive this expression from the derivative of  $S$  in terms of  $\alpha_2$ ,

$$\begin{aligned} & \left( \sum_{i=1}^n A_{i,1} \cdot A_{i,2} \right) \alpha_1 + \left( \sum_{i=1}^n A_{i,1}^2 \right) \alpha_2 \\ &= \sum_{i=1}^n (d_i - (P_0 B_0^3(t_i) + P_0 B_1^3(t_i) + P_3 B_2^3(t_i) + P_3 B_3^3(t_i))) \cdot A_{i,2} \end{aligned} \quad (7.16)$$

Combining these two equations, we can write them in a somewhat clearer form using the following shorthand,

$$\begin{aligned} c_{1,1}\alpha_1 + c_{1,2}\alpha_2 &= X_1, \\ c_{2,1}\alpha_1 + c_{2,2}\alpha_2 &= X_2 \end{aligned} \quad (7.17)$$

These two equations can also be expressed in matrix form as,

$$\begin{bmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \quad (7.18)$$

which allows us to solve for the values of  $\alpha_1$  and  $\alpha_2$  simply as,

$$\begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{bmatrix}^{-1} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \quad (7.19)$$

Once a single candidate curve has been fit, the curve must be compared to the original sample points to test its quality. To generate points on the fit curve with which to compare the sample points, each sample point is assigned a corresponding parametric value in the range of the curve using a simple chord-length parameterization. For each sample point starting with the second point, the dis-

tance between that point and the initial sample point along the straight segments of the polyline is calculated. The calculation is simply the distance stored with the previous point, plus the Euclidean distance between the previous and current point. This has the effect of measuring the length of the straight line segments that connected successive pairs of points. Once each of the sample points is associated with a distance, the whole set of distances is divided through by the total chord length of the polyline to generate distances scaled to a unit parameter range. These distance values are then entered into the curve equation as parametric parameters to find a corresponding point on the curve.

By comparing the squared distances between the sample points and their corresponding curve points, the quality of the fit provided by the fitting algorithm can be determined. If none of the distances are larger than the threshold distance for the fitting process then the fit is successful, and the current curve becomes part of the finished stroke. If however any distance is above the threshold, further actions must be taken. The current application uses a squared distance tolerance of approximately one one-thousandth the default modeling space width. Qualitative tests have shown that this tolerance strikes a good balance between performance and visual accuracy.

If the initial fit should fail, at this point in the algorithm it's possible to simply divide the set of sample points and begin recursive calls. However, as Schneider points out [Schneider, 1990], the set of curve points used for comparison is based solely on the chord length parameterization, and is very likely an inaccurate representation of the points on the curve closest to the samples. Rather than making the recursive call, which can be relatively expensive, if the fitting error is within a multiple of the threshold value then it may be worth the effort to calculate more

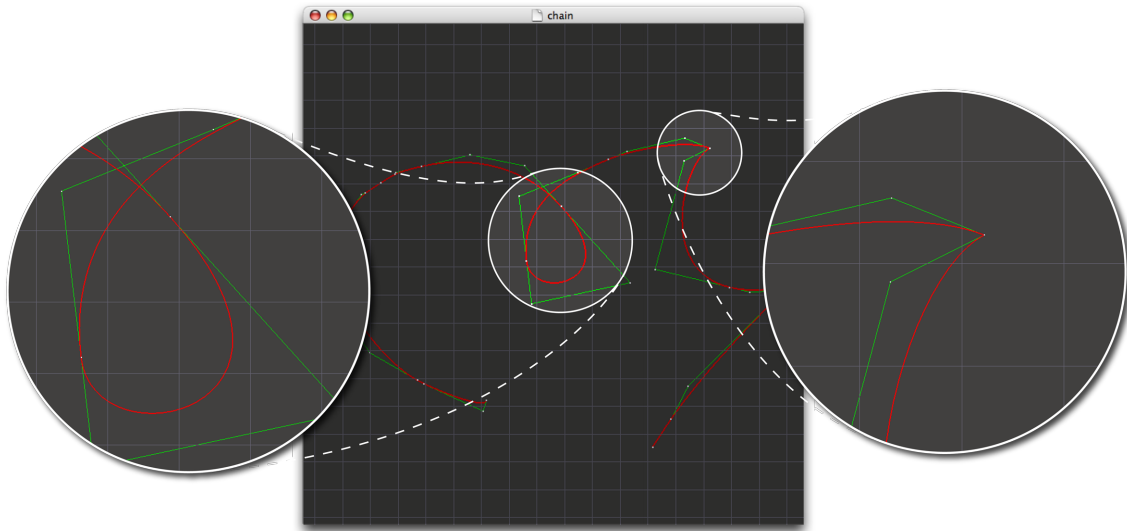
accurate points in the hope that the fit actually was successful and no further recursion is necessary.<sup>1</sup>

Using the chord length parameterization derived values as a starting point, a simple Newton-Raphson iteration is used to calculate new parametric values, and recalculate the fitting error. If the fit continues to fail, the iteration process can be preformed multiple times to increase the accuracy of the parameter values. Because the iteration process is fast to calculate and converges quickly, Schneider suggest that four to five iterations be run before a fit is officially declared a failure and the recursive call is made. By setting the iteration limit artificially high, the author has found that successful fits can often be found at slightly higher iteration levels. Based on these results, the present application allows as many as ten iterations before throwing in the towel.

The implementation of this fitting algorithm is based on portions of the sample C code provided in Schneider's article [Schneider, 1990]. However, to integrate the algorithm into the application it was rewritten into an object-oriented style, which allowed in some instances a more straightforward representation of the underlying theory and a number of memory management optimizations. The final implementation runs very efficiently on the testing hardware, and is capable of fitting very large sample point sets—on the order of 1000 points—at interactive rates. The algorithm's performance could certainly have allowed curves to be fit in real time, however as discussed in Section 6.4.7, for interface reasons it was decided to apply the algorithm as a single batch process at the end of each input stroke. The algorithm is applied to each set of contiguous non-corner points generated by

---

<sup>1</sup>The present application uses 100 times the threshold or approximately one twentieth the width of the viewing volume.



**Figure 7.3.** This figure demonstrates the results of the stroke fitting process. The image in the center is a single stroke fit by the system (shown in red), with its associated control points (white) and control polygon (green). To the left, a section of the stroke has been magnified, showing how a smooth section of the stroke is represented by three separate but  $G^1$  continuous Bézier curve segments. To the right, a different section of the curve is magnified, showing a corner where two  $C^0$  continuous segments meet.

a single stroke event. The result of this process is a single chain of cubic Bézier curves representing each stroke—see Figure 7.3. The smooth segments of each curve are represented as curve segments meeting with  $G^1$  continuity while each corner point joins two curves with  $C^0$  continuity.

#### 7.2.4 Bézier Curve Length

One additional algorithm that has proved important in working with the Bézier curve representation is the ability to quickly calculate the length of a Bézier curve segment or chain of segments. A particularly elegant algorithm for this purpose is described by Jens Gravesen [Gravesen, 1992]. The basic idea is as follows. The length of a Bézier curve can be estimated by calculating the average of the length

of a chord from the initial to final control point with the length of the curve's control polygon.

For a Bézier curve of degree  $n$  with control points  $P_0, \dots, P_{n+1}$ , the length estimate is calculated as:

$$\begin{aligned}\mathcal{L}_{chord} &= |P_0P_{n+1}|, \\ \mathcal{L}_{polygon} &= \sum_{i=0}^n |P_iP_{i+1}|,\end{aligned}\tag{7.20}$$

$$\mathcal{L}_{curve} = \frac{2\mathcal{L}_{chord} + (n-1)\mathcal{L}_{polygon}}{n+1}$$

Given a cubic Bézier curve with control points  $P_0, P_1, P_2$ , and  $P_3$ , this expression reduces to,

$$\begin{aligned}\mathcal{L}_{chord} &= |P_0P_3|, \\ \mathcal{L}_{polygon} &= |P_0P_1| + |P_1P_2| + |P_2P_3|,\end{aligned}\tag{7.21}$$

$$\mathcal{L}_{curve} = \frac{1}{2}\mathcal{L}_{chord} + \frac{1}{2}\mathcal{L}_{polygon}$$

This calculation provides only a rough estimate of the curve's length. A measure of the error of this length estimate is  $\mathcal{L}_{polygon} - \mathcal{L}_{chord}$ . The formulation is derived from the fact that, as the curve segment becomes more and more linear, the control polygon length and chord length approach each other. Unless the curve is linear, the result of this process is not of much use, however, the error measure provides a simple method of deriving the length to within a desired tolerance. To find the length, we can recursively subdivide the curve at its midpoint and perform the length calculation again on each side, summing the results. This algorithm works because as the curve is subdivided each segment becomes more linear until in the limit the polygon and chord lengths are the same. Thus, as we



subdivide, the error of the sum total of lengths approaches zero as  $2^{-4m}$  where  $m$  is the number of subdivisions. An implementation of this algorithm is used by the application to quickly generate the length of any Bézier curve segment. These lengths can then be summed together to generate the length of an entire stroke.

### 7.3 3-D Construction Foundations

Owing to the sketch-based nature of the construction system, the strokes generated by the user, as explained in depth in previous section, form the foundations of the data structures and construction methods used to form 3-dimensional geometry. However, creating 3-dimensional curves and surfaces places different sorts of demands on that data. Whereas the Bézier chain representation and its associated algorithms provide certain benefits to the curve fitting process, they also have certain drawbacks when used for 3-D construction. To deal with this, when utilized as construction components, the user's strokes undergo a conversion process that generates a visually identical curve representation that is better suited to the 3-D construction algorithms.

Similarly, a structure is needed to hold and manipulate each 3-D modeling component. Because of the large amount of geometric data involved, and the frequency at which this data must be transported to the graphics hardware, the efficiency of this data structure is very important to the performance of the application. At the same time, in order to make the same data easily and efficiently accessible and understandable to algorithms that will manipulate the components once constructed, a great deal of additional topological information must also be stored. Rather than addressing these issues with a single representation, a set of parallel structures is used, one a highly memory efficient set of vertex arrays,

and the other a highly interconnected topological structure. Internally these two alternate views of the same data are maintained in unison as a mesh structure that provides a unified interface to both interpretations of the data.

The following two sections discuss these foundations of the 3-D construction system.

### **7.3.1 Curve Conversion**

As part of the construction process the application will often need to calculate points at various locations along the user's strokes. The cubic Bézier curve chain representation used to describe drawing strokes within the system has many useful properties, as described in Section 7.2.3.1. However, a major drawback to the chain representation is that it is difficult to calculate a position on the stroke as a whole, given a single parametric parameter. This is because the chain is composed of an arbitrary number of cubic Bézier segments, each with an arbitrary length. In order to find the point, for example, half way along the stroke, the system must measure the lengths of each curve segment to determine in which segment the midpoint falls, then, calculate the corresponding parametric values for the beginning and ending points of this segment in terms of the entire length of the curve, and finally compute the value of the stroke's midpoint in terms of the local parametric range of the curve segment.

If we were to store the curve segment lengths and parametric values for their end points, none of these calculations is particularly complex, however not only is the calculation of points important for many of the construction algorithms, but so too are normal vectors and multiple derivatives, each of which would required similar mathematical acrobatics to calculate from the chain representation.

A more elegant alternative is to convert the cubic Bézier chain representation into a more general curve representation that supports the operations we are interested in. For the present application, the *b-spline* curve representation was selected. B-spline curves, like Bézier curves, are a parametric representation for polynomial curves of arbitrary degree based on control point constraints and a set of basis functions. However, unlike the Bézier curve, b-splines have an additional set of parameters called a *knot vector*. The knot vector is a non-decreasing list of parameter values spanning the parametric range of the b-spline curve. The knots are used to determine, for any given parametric value, which subset of the control points will affect the curve in that location. The parametric equation for the b-spline curve is expressed as,

$$C(u) = \sum_{i=0}^n P_i N_i^p(u), \quad u \in \{u_0, u_1, \dots, u_m\} \quad (7.22)$$

Where the degree of the curve is  $p$ . There are  $n + 1$  control points  $P_0, \dots, P_n$ , and  $m + 1$  knot values  $u_0, u_1, \dots, u_m$ .  $N_i^p$  is the  $i^{\text{th}}$  b-spline basis function of degree  $p$  where,

$$N_i^0(u) = \begin{cases} 1, & \text{where } u_i \leq u < u_{i+1} \\ 0, & \text{otherwise} \end{cases} \quad (7.23)$$

$$N_i^p(u) = \frac{u - u_i}{u_{i+p} - u_i} N_i^{p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1}^{p-1}(u) \quad (7.24)$$

Notice that unlike the Bézier representation, the degree of the curve does not directly correspond to the number of control points. The relationship between the number of control points and degree is a function of the number of knot values, expressed as  $m = n + p + 1$ . This is because the b-spline curve can represent multiple spans of equal degree in a single representation. For a curve of degree

three, if four control points and eight specific knot values are provided then the expression for the b-spline curve degrades to match the expression for the identical Bézier curve. If however, for the same degree three additional control points and knots are provided, the representation can express the entire set of Bézier curve segments that make up the chain of a user's stroke, all within a single b-spline curve. By carefully calculating the positions and frequencies of knot values, it's even possible to express the degree of parametric continuity between the curve segments.<sup>2</sup>

Because the b-spline curve is a generalization of the Bézier representation, it shares many of the same attractive mathematic properties of the Bézier. This includes affine invariance, the convex hull property, and the variation diminishing property. However, by wrapping the entire Bézier curve chain up into this single formulation, we can easily calculate position, tangent, and higher derivative information at any point along the curve using the standard b-spline curve algorithms. Because of the mathematical properties of the b-spline representation and its basis functions,<sup>2</sup> some operations such as calculating multiple derivative values can be performed more efficiently.

The process of translating the chain of cubic Bézier curves into a single cubic b-spline curve is achieved with a simple mechanical algorithm. The basic algorithm is described by Sederberg and Greenwood [Sederberg & Greenwood, 1995]. The control points of the b-spline curve itself are identical to the control points of the individual Bézier curve segments, however because each successive segments contains a coincident control point with its neighbors some of the original control

---

<sup>2</sup>For more information on the b-spline curve representation please see [Farin, 2002]. The b-spline algorithms used in the present application are based on Piegl & Tiller [Piegl & Tiller, 1997].

points are repeats. For the b-spline representation these extra points are superfluous and are removed. The only real work of the algorithm is to generate knot values that will drive the path of the b-spline along the same route defined by the chain of Bézier segments.

The parametric continuity between the curve segments composing the b-spline is also a function of the knot values, in this case determined by the multiplicity of each knot. At the parametric knot value  $u_i$  the curve is  $C^{p-k}$  parametrically continuous where  $p$  is the degree of the curve, and  $k$  is the multiplicity of the knot value  $u_i$ . In addition, in order to interpolate the initial and final control point of the b-spline curve, the initial and final knot values must have multiplicity equal to the degree plus one.

As noted in Section 7.2.3.1, each Bézier curve segment meets with  $G^1$  geometric continuity. As you will recall, this is a less stringent constraint than  $C^1$  parametric continuity, so forcing  $C^1$  in the b-spline representation would change the path of the curve. Instead, each segment in the b-spline will meet with  $C^0$  positional continuity, meaning that each internal knot value will be repeated three times. Thus, the resulting knot vector will have the form  $[k_0, k_0, k_0, k_0, k_1, k_1, k_1, \dots, k_{n-1}, k_{n-1}, k_{n-1}, k_n, k_n, k_n, k_n]$  where  $n$  is the number of segments in the original Bézier chain. It turns out that the overall parametric range of the knot vector is immaterial, and so for simplicity the values of  $k_0, \dots, k_n$  are evenly spaced. The result of this process is a cubic b-spline curve that exactly follows the path of the original cubic Bézier chain.

### 7.3.2 Model Representation

The 2- and 3-dimensional modeling components generated by the application are composed of a network of triangles. As the models become larger and more complex, the number of triangles increases rapidly, and so it is necessary to organize this information, lest the situation devolve into a veritable ‘triangle soup’. From a geometric perspective, it is desirable to arrange the triangle data into blocks of contiguous memory. This allows the vertex information to be passed to the OpenGL sub system, and in turn the graphics hardware, in large chunks, rather than clogging the hardware bus with thousands of individual instruction. By providing the information to OpenGL in this way, vertex throughput, and in turn display performance is increased substantially.

From a topological perspective, it is desirable to store not only simple triangle and vertex information, but also information about the connectivity of each vertex, edge, and face of the model. By providing algorithms with a quick and easy means of querying this adjacency information, it becomes possible to efficiently calculate other properties of the surface. In the present application this information is used to calculate surface normals to provide more accurate lighting effects.

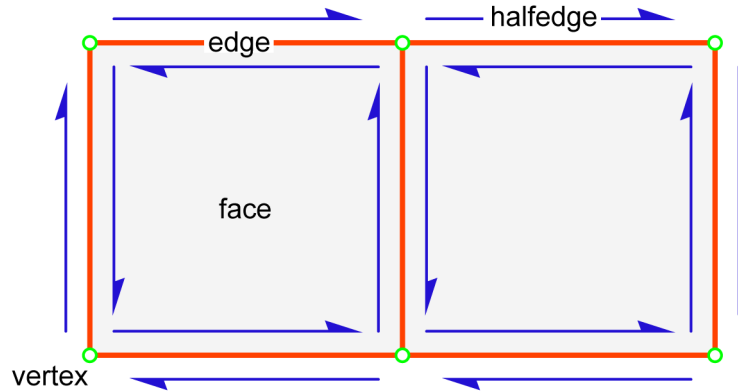
To capture both the geometric and topological information of each model a mesh data structure was developed. The mesh structure is composed of two main components, one each to support the geometric and topological information. On the geometric side, vertex information for each model component is stored in a single contiguous array. Indexes into this array are then stored in two additional arrays, one containing the ordered list of indices for the points defining arbitrary polygons, and a second containing order lists of indices that form strips of triangles. When the time comes to render the modeling component, pointers to these

blocks of memory can be passed directly to the OpenGL subsystem where they are rendered at high speed.

Although the vertex array structure is an ideal interface for the graphics hardware, it proves to be a very poor interface for algorithms attempting to access the data. Instead, a secondary mesh structure stores all of the relevant topological information as a highly interconnected network of pointers. The computer graphics literature provides examples of a number of mesh structure designs, each with its own strengths and weaknesses. A primary concern when choosing a mesh design is the structure's ability to correctly encode all of the topological properties supported by the modeling system. Many structures cannot represent, for example, isolated components or surfaces with holes or non-manifold conditions. In the context of the current application, each mesh structure will be used to represent a single connected modeling component, generated through the tessellation, sweeping, or generalized cylinder methods described in Section 6.4.11. Based on these design constraints, a version of the *halfedge* data structure was selected because it supports the necessary functionality and has a relatively simple and straightforward structure.

The halfedge structure developed for this application is loosely based on the design outlined by Brönnimann [Brönnimann, 2001]. The structure is composed of four classes: the *mesh vertex*, *mesh edge*, *mesh face*, and *mesh halfedge*. As the name suggest, the main component of the structure is the halfedge class. To understand exactly what a 'halfedge' is, it helps to consider this simple example. Imagine a mesh representing two squares, connected along a single edge. This structure contains six vertices, seven edges, and two faces—please refer to Figure 7.4. We can think of each edge of this mesh as actually composed of two

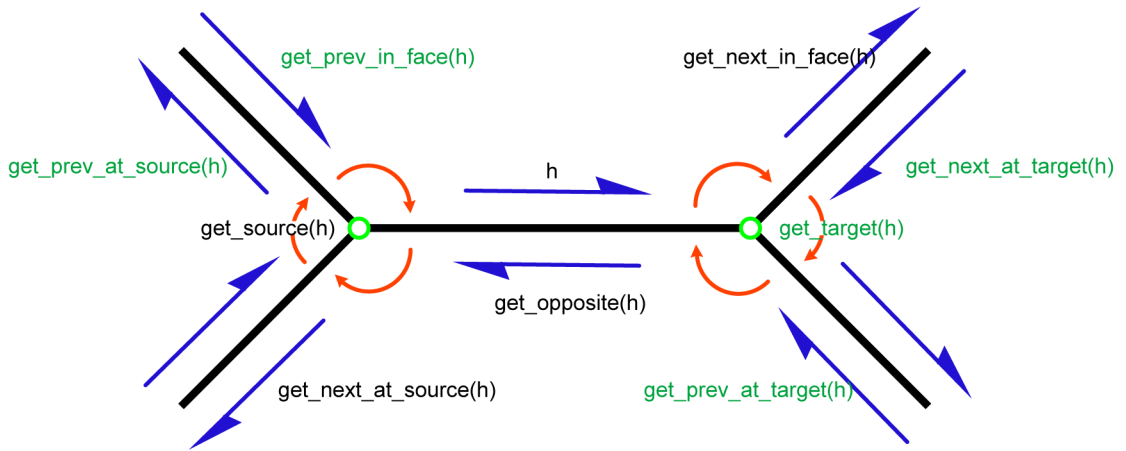
halfedges—two lines running between the endpoints of the original edge, but each directed, and pointing in opposite directions. Halfedges are arranged so that following the direction of the halfedges from one vertex to the next traverses a path around each face in a counterclockwise direction, or around the outer boundary of the mesh in a clockwise direction. This network of directed halfedges provides a simple means of moving around the local neighborhood of the model.



**Figure 7.4.** This diagram represents a mesh structure composed of two squares, connected along one edge. Notice that each edge in the figure (shown in red) is composed of two alternately facing halfedges (shown in blue). Following the halfedges within each square forms a counterclockwise circuit between the four vertices (shown in green) forming that square. At the same time, the halfedges along the outside of the figure form a clockwise circuit of the perimeter of the diagram.

Structurally, the halfedge class is defined by six pointers. The first three point to the *next*, *previous*, and *opposite* halfedges to the current halfedge. The remaining three pointers point to the mesh vertex at the origin of the halfedge, the mesh edge associated with the halfedge, and the face to the left of the halfedge. Along with this information, the halfedge class provides an interface of query functions that allow all of this information to be accessed. These queries are





**Figure 7.5.** This figure provides a graphical depiction of the queries available in the halfedge mesh structure, and their relationship to the mesh’s topological interpretation. Given the halfedge  $h$  (shown in the center of the diagram), each of the surrounding edges halfedges, and vertices can be accessed. Queries shown in black are direct properties of the halfedge objects, while queries shown in green are accessed as proxy functions by applying one of the black operators to access a neighboring component and then using the same functions from its position. This diagram is based on an original diagram by Brönnimann [Brönnimann, 2001].

shown in Figure 7.5.

Because most of the topological information is stored in the halfedge class, the vertex, edge, and face classes are very simple. Each contains a pointer to its associated halfedge, and a reference to its corresponding geometric information. The implementation of the mesh structure is written using generic C++ templates, allowing any geometric information to be stored. For the present application this simply entails the vertex array indexes of the associated geometric components for faces and vertices. Each class also contains proxy methods to many of the query functions provided by the halfedge structure to shield the developer from dealing with the halfedges directly.

This mesh implementation is capable of representing any connected, 2-manifold

topological structure with boundaries. It does not support non-manifold conditions of vertices, edges or faces, nor faces with internal holes. Based on Brönnimann generic description of the halfedge structure and its possible features, the current implementation can roughly be described as *bidirectional*; *supports vertices*; *source-linked*; *target-linked*; *facet-linked*; *no holes*; *connected*; *non-singular vertices*; and *mutable*.

Although it's possible to build a mesh structure by hand, because of the highly interconnected nature of the mesh, creating each mesh component and weaving together their pointers is an extremely tedious and error prone process. Thus, rather than dealing with the mesh's constituents directly, two generic interface systems were also developed to provide a layer of abstraction between the application's algorithms and the low level mesh structure. The first is a set of generic iteration objects that can be used to traverse the mesh structure. These objects are designed to resemble the iterator design pattern, and are derived from iterator base classes in the C++ Standard Template Library, however because of the cyclic nature of the structure they are referred to as *circulators* rather than iterators.

Each circulator is based on a mesh component input type and a mesh component output type: the input type determines what aspect of the mesh structure will guide the path of the circulator, and the output type determines what type of object will be returned. Thus, if the developer wants to traverse the ring of edges surrounding a given point, then the input type would be a mesh vertex and the output mesh edges. Similarly, to find the vertices around a given face, the input type would be a mesh face and the output mesh vertices. By using these circulators rather than traversing the structure by hand, the developer's code is agnostic as to the actual implementation of the mesh or its structure, allowing

the structure to be updated at a later time, or the algorithm to be applied to a different implementation of the mesh structure in a different application without being rewritten.

The second abstraction system integrated into the mesh structure is a set of operators that wrap all of the pointer manipulation tasks for a number of common construction functions into single, atomic operations. These operators are based on the Euler graph construction operators or *Euler operators*, which provide a closed and sufficient set of operations to construct a connected graph. Euler operators have been applied to mesh construction in the past, especially in the construction of boundary representation meshes for volume modeling, because the operators guarantee that the mesh structure will be sound after each operation is applied. The original Euler operators consist of very simple operations that can be applied in succession to construct a more complex mesh shape. The operators have self-descriptive names like `MakeVertexEdge` or `MVE` to create a new vertex joined to the current by a new edge, `MakeEdgeLoop` or `MEL` to create a new edge joining a loop, etc. The current application defines the following mesh construction functions:

- **MakeVertex (MV)**—create a new, isolated vertex.
- **MakeVertexEdge (MVE)**—extend a new wire edge from the current vertex to a new vertex.
- **MakeEdgeLoop (MEL)**—Close two ends of a disconnected wire edge into a loop.
- **MakeFaceKillLoop (MFKL)**—Create a face from a closed loop.

These operators do not represent the full complement of standard Euler operators. Specifically, most of the inverse operators used to deconstruct a mesh are omitted from the current implementation because they are not necessary for the construction process. The operators above provide the basic tools necessary to construct the mesh structures used in the application, however they provide a highly granular interface to the construction process. Furthermore, although these operators can be used to form constructions ensuring that the mesh is correctly configured at each stage, there are some situations in which it is more efficient to perform a large number of changes all at once, only ensuring that the mesh is in a sound state at their completion. For example, when dozens or hundreds of point samples from a stroke are converted into a wire edge polyline, it is much more efficient to create the mesh components all at once rather than making hundreds of calls to the `MakeVertexEdge` operator. Thus, the following pseudo Euler operators were also developed, which subsume a number of common or inefficient operations into a single unified operation.

- **MakeEdgeFace (MEF)**—Make a new edge closing an open wire edge and construct a face in the resulting loop. Functionally equivalent to `MakeEdgeLoop` followed by `MakeFaceKillLoop`.
- **MakeWireEdge (MWE)**—Make a wire edge from an ordered list of points. Functionally equivalent to `MakeVertex` followed by a number of `MakeVertexEdge` operations.
- **MakeWireLoop (MWL)**—Construct a closed loop from an ordered list of points. Functionally equivalent to `MakeVertex` followed by a number of `MakeVertexEdge` operations followed by `MakeEdgeLoop`.

- **MakeFace (MF)**—Construct a face from an ordered list of points. Functionally equivalent to MakeVertex followed by a number of MakeVertexEdge operations followed by MakeEdgeLoop followed by MakeFaceKillLoop.
- **ExtrudeWireEdge (EWE)**—Given a wire edge, construct a new identical wire edge in which every vertex along the edge is connected to corresponding vertices in the original wire edge, forming a strip of triangles.

These operators are used extensively in the construction of 3-D modeling components.

## 7.4 Sweep

The simplest 3-D construction available to the user takes the form of a generalized extrusion or sweep. The cross sectional shape of the sweep is defined by the active die stroke drawn by the user. The path of the sweep is derived from the user's active path stroke. To construct the sweep, the die stroke is used as a template to create a number of contour curves. Each of these curves is then positioned relative to the original die stroke along a path mirroring that of the path stroke. At the same time, each contour curve is also oriented relative to original die stroke's orientation based on an orienting frame extracted from the first and second derivatives of the path curve at the current position along the sweep path. The sweep process is performed iteratively, adjusting and positioning one contour curve at a time until the path stroke is exhausted.

From a mathematical perspective, the surface of the 3-dimensional model is parametrically defined by two parameters: the distance along the path curve, which indicates the position and orientation of a contour curve; and the distance

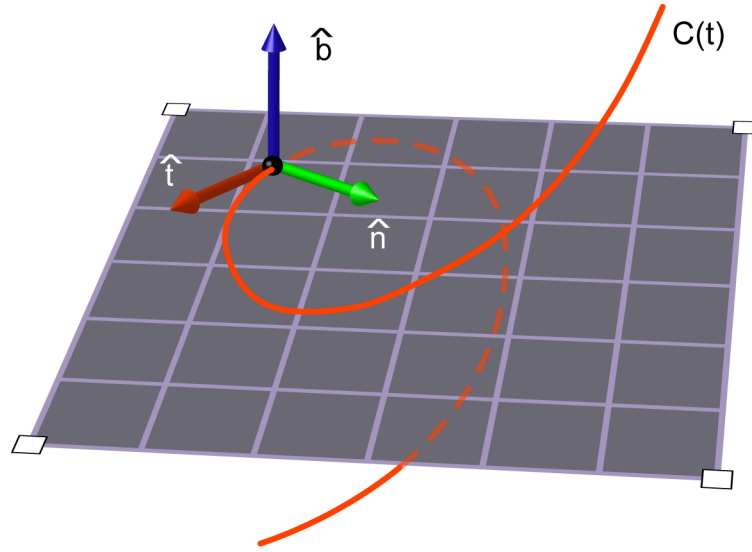
around the current contour at the current location, which indicates a point on the surface. Although the surface can be evaluated using these parametric methods, to improve the drawing performance and allow the surface to be manipulated by other methods, the component is evaluated once, as it's constructed. The evaluated points are connected into a persistent mesh structure that can then be easily and efficiently manipulated and displayed.

#### 7.4.1 Frenet Frame

The most important aspect of the sweeping process is correctly placing the contour curves along the sweep path. Finding the position of each contour is relatively easy. As points are evaluated along the path stroke, each relative distance between the current point and the last can be fashioned into a translation transformation to place the contour curves along the sweep path like beads on a string. A more difficult challenge is orienting the contours so that they follow the dynamics of the path, turning as the path turns and then straightening out again as the path becomes straight.

As a method of correctly orienting the contours, Davison suggests the *Frenet frame* [Davison, 2003]. The Frenet or Frenet-Serret frame is a form of reference frame that describes the kinematics properties of a particle as it moves along a space curve. The formulation is derived from the Frenet-Serret formulas, which were independently developed by Jean Frédéric Frenet and Joseph Alfred Serret [Contributors, 2007a]. The frame is composed of three unit vectors derived directly from the curve at a given parametric value. These three vectors form what is called the *Frenet triad*, which follows the path of the curve and orients itself according to the direction and curvature of the curve's path—see Figure 7.6. The three normal

vectors are also mutually orthogonal, meaning that they form an orthonormal basis for each position along the curve. The basis vectors can be arranged into a transformation matrix that positions and orients the contour to the corresponding point on the space curve.



**Figure 7.6.** A Frenet triad composed of mutually orthogonal unit tangent (red), normal (green), and binormal (blue) vectors is placed along a space curve. The three vectors of the triad define a Frenet frame.

To calculate the basis vectors for the Frenet frame at a given parametric value on the curve, it is first necessary to calculate the first and second derivatives  $\vec{t}$  and  $\vec{a}$ , and the curvature vector at the parameter value  $\vec{k}$ . The first derivative represents the tangent vector to the curve, and the second is related to the curvature at the given location. From these two vectors, the curvature vector can be calculated as,

$$\vec{v} = \frac{\delta}{\delta t} C(t_i), \quad \vec{a} = \frac{\delta^2}{\delta t^2} C(t_i), \quad \vec{k} = \vec{v} \times \vec{a} \times \frac{\vec{v}}{|\vec{v}|} \quad (7.25)$$

The three unit vectors composing the Frenet triad are then calculated as follows. The first is the unit tangent vector  $\hat{t}$ , derived directly from the first derivative above. The second is the unit normal vector to the curve  $\hat{n}$ , calculated as the unit vector in the direction of curvature. The final unit vector, referred to as the binormal  $\hat{b}$ , is the unit vector orthogonal to the other two. The formulations are as follows,

$$\hat{t} = \frac{\vec{v}}{|\vec{v}|}, \quad \hat{n} = \frac{\vec{k}}{|\vec{k}|}, \quad \hat{b} = \hat{t} \times \hat{n} \quad (7.26)$$

The unit tangent vector and normal vector span the osculating plane to the curve at the given parametric location, the normal and binormal span the normal plane to the curve, and the tangent and binormal span the rectifying plane. These basis vectors, along with the position of the curve evaluated at the parameter value are arranged into the an affine transformation matrix  $M$ ,

$$M = \begin{bmatrix} \hat{n}_x & \hat{b}_x & \hat{t}_x & C(u_i)_x \\ \hat{n}_y & \hat{b}_y & \hat{t}_y & C(u_i)_y \\ \hat{n}_z & \hat{b}_z & \hat{t}_z & C(u_i)_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7.27)$$

This matrix is sufficient to position and orient contour curves defined in canonical position to any point along the path curve. However, Bloomenthal points out that plotting contours along the curve using this method directly can lead to undesirable twists or torsions in the resulting sweep [Bloomenthal, 1990]. These twists can appear at locations where the second derivative of the curve is discontinuous—see Figure 7.7. At these discontinuities the curvature vector can suddenly change direction, causing a drastic flip or twist in the reference frame. Recalling that in the present application the curve segments composing the user’s strokes meet with



$G^1$  geometric continuity, which does not guarantee a continuous second derivative, it is likely that these twists will appear in sweeps constructed directly from the path curve. In addition, in straight sections of the curve the second derivative disappears, shrinking the magnitude of the curvature vector to zero. In these sections it's not possible to calculate the frame's normal vector component, nor the binomial that depends on the normal for its definition.



**Figure 7.7.** An example of the effects of Frenet frame twist from Davison's human ear modeling system [Davison, 2003].

To address these issues, rather than calculating a new frame at each parametric location along the path curve, instead we can calculate a single frame at the beginning of the curve, and then propagate this frame along the curve by updating its component basis vectors when new information is available [Bloomenthal, 1990]. This method has the disadvantage that it is no longer possible to calculate the frame from an arbitrary point on the curve without first calculating the initial frame and propagating it to the desired position. However, by updating

the frame, we can avoid the undesirable twists, and generate frames over straight sections of the curve.

The initial frame is calculated as described above. For each successive frame, the new frame can be generated by rotating the original frame so that the tangent vector of the previous frame aligns with the tangent vector at the current location on the curve. The axis of rotation is any vector perpendicular to the two tangent vectors, thus,

$$axis = \hat{t}_{i-1} \times \hat{t}_i \quad (7.28)$$

where  $\hat{t}_{i-1}$  is the previous unit tangent vector, and  $\hat{t}_i$  is the current unit tangent. The angle of rotation is then easily calculated as,

$$\theta = \cos^{-1} \left( \frac{\hat{t}_0 \cdot \hat{t}_1}{|\hat{t}_0||\hat{t}_1|} \right) \quad (7.29)$$

Using this formulation, the normal and binormal orientation are calculated directly only once, at the beginning of the curve. All subsequent normal and binormal vectors are derived from rotations of the originals, ensuring that they change smoothly over the course of the curve. In instances where the curve's path straightens out, the previous and current tangent vectors are in alignment, so no rotation is necessary. Given this formulation, the only corner cases are instances in which the start of the curve is perfectly straight, and so no initial second derivative can be calculated. In this situation, the normal vector of the plane containing the stroke is used as a suitable estimate for the binormal, and the unit normal of the frame is calculated as the cross product of the substitute binormal and the tangent vector.

### 7.4.2 Sweep Construction

The sweep construction algorithm can be broken into a preparation stage, and a simple iteration. The preparation consists of five steps:

1. Calculate the stroke lengths.
2. Convert the input strokes into construction curves.
3. Orient the construction curves.
4. Generate the parametric parameters that will define the surface.
5. Prepare for the iteration.

Once the preparation is complete, a simple 5-step iteration is performed to construct the 3-dimensional structure.

1. Propagate the alignment frame.
2. Generate a profile curve.
3. Position the profile curve.
4. Evaluate the profile over the parametric parameters.
5. Extend the mesh with the resulting surface points.

The sweep is generated from two input sources, the user's active die stroke and active path stroke. Before information from these strokes can be utilized, several steps must be taken to correctly prepare the data. First, the length of both strokes is calculated using the Bézier length algorithm described in Section 7.2.4, and stored for future use. This length value will be used to calculate the number of sample points plotted along each stroke to form a convincingly smooth surface. Next, because the strokes will need to be repeatedly evaluated as part of the construction process, each is converted into a b-spline representation using the

Bézier chain to b-spline conversion algorithm described in Section 7.3.1. These converted versions of the strokes are referred to as the *die curve* and *path curve* respectively, as opposed to the *die stroke* and *path stroke* cubic Bézier chain representations. The final step in preparing the curves is to reorient the curves in preparation for the sweep.

When initially collected, as discussed in Section 6.4.9, the user's strokes are projected onto a 2-dimensional drawing plane. Each drawing plane and its components are positioned in the modeling space by an internal reference frame. Whenever the plane's position or orientation is adjusted by the user, the new position is stored internally into this reference frame, which is then used to position the plane itself as well as grid lines and associated interface components when the drawing plane is drawn by OpenGL. However the position of the points defining each stroke—both the raw sample points, and the control points of curve segments generated in the fitting process—are stored internally in the world coordinate system, meaning that the position of each curve is represented independent of the drawing plane on which it sits. In order to simplify the sweeping process, the *die curve*, generated from the *die stroke*, is moved to a canonical position by multiplying its control points with the inverse of the positioning frame from the drawing plane on which it was drawn. In effect the resulting curve sits as if it were drawn on the default xy-plane in the modeling space.

Similarly, the *path curve* must also be repositioned so that its position does not change in relation to the *die curve* by multiplying its control points by the inverse of the *die stroke's* drawing plane's positioning frame. If, for example, the *path stroke* were defined roughly perpendicular to the plane of the *die stroke*, the effect of this adjustment is to place the *path curve* in the same roughly perpendicular

orientation to the canonically positioned die curve.

After the die and path curves are correctly oriented, the system generates a list of the parametric parameters at which each curve will be evaluated to form the surface. Recalling that parametric locations on the path curve define the positions of contours of the surface, and parametric locations on the die curve will generate surface points, it is vital that the curves are evaluated with a high enough frequency that the resulting surface is visually smooth. To achieve this, the length of each curve is divided by a resolution setting to determine how many parameter values should be generated. These values will be evenly spaced along the extent of each curve.

For a smooth curve, this even spacing of parameter values provides a pleasing result. However, recall that the curves, and the strokes that defined them, might contain discontinuous corners and sharp turns. If the evenly spaced parameter values are placed to either side of one of these discontinuous features, then the sharp corner will be rounded over, destroying the visual effect of the corner. To account for these discontinuities, additional parametric values are generated for each unique knot value in the curve's b-spline representation. This has the effect of evaluating the curve at the join points between each of the Bézier curve segments in the original chain. The set of parametric values for the die curve will be referred to as  $[t_0, t_1, \dots, t_{n-1}, t_n]$ , and the values for the path curve  $[u_0, u_1, \dots, u_{m-1}, u_m]$ .

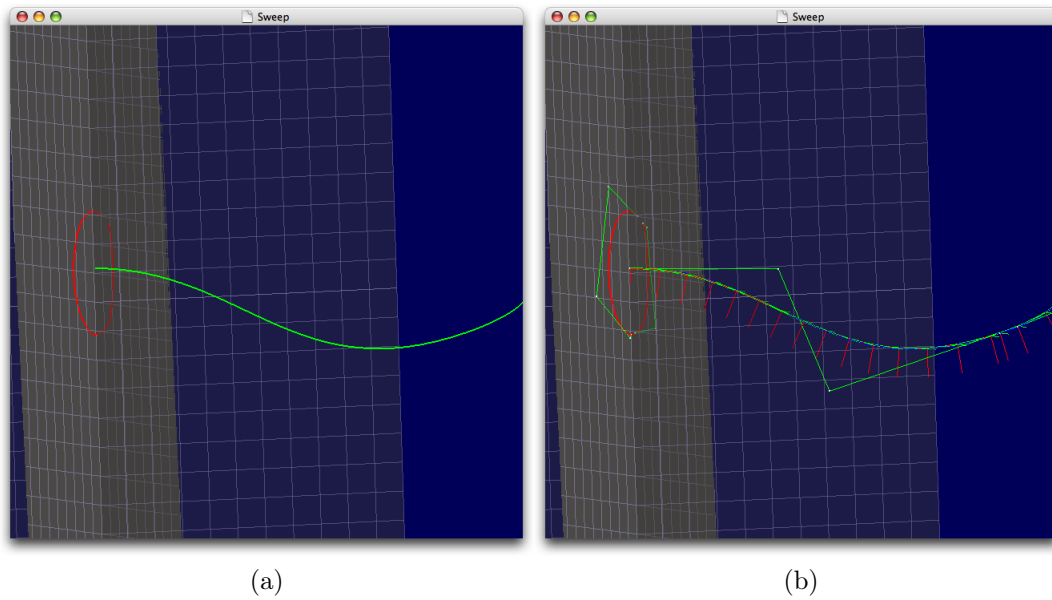
The final preparatory step is to initialize the system's state before iteration begins. First, a new mesh data structure is created that will hold the geometric and topological information defining the modeling component. The mesh is initialize with a single wire edge consisting of points generated from the die curve at each of the parametric locations  $t_i$ . The last step is to generate the initial Frenet

frame at the beginning of the path curve. This reference frame will serve as the current reference frame  $F_c$ , and will be propagated along the path of the curve during the iteration.

Along with this frame, which is represented as a transformation matrix, two additional transformation matrices are initialized and will be updated during the iteration process. The first is the inverse of the previous reference frame  $I_p = (F_c)^{-1}$ , and the second is an accumulating transformation  $A$ . These two transformations, along with the current reference frame will be used to position each profile curve.

Once the preparations are complete, the iteration process begins. The iteration is performed over each parametric value  $u_i$  generated for the path curve. The first step is to save the inverse of the previous reference frame into  $I_p$ , and then propagate the reference frame to the current parametric location on the path curve. Next, a profile curve is generated using the original die curve as a template. Each profile is initially positioned at the beginning of the sweep in a canonical orientation. The next step is to position this curve along the sweep path.

The Frenet reference frames generated from the path curve describe a position that is centered along the path curve like a spine—see Figure 7.8. However, rather than sweeping the 3-D surface out along the path stroke directly, we would like the 3-D structure to emanate from the position of the user’s die stroke. This method produces a more intuitive behavior, and allows the user to draw his or her path stroke at different locations in relation to the die stroke to achieve different effects as noted in Section 6.4.11. To achieve this, what’s required is a transformation describing the relative difference in translation and rotation between each reference frame and the previous. This difference is equivalent to a transformation of the coordinate space by the inverse of the previous frame, and then a transformation



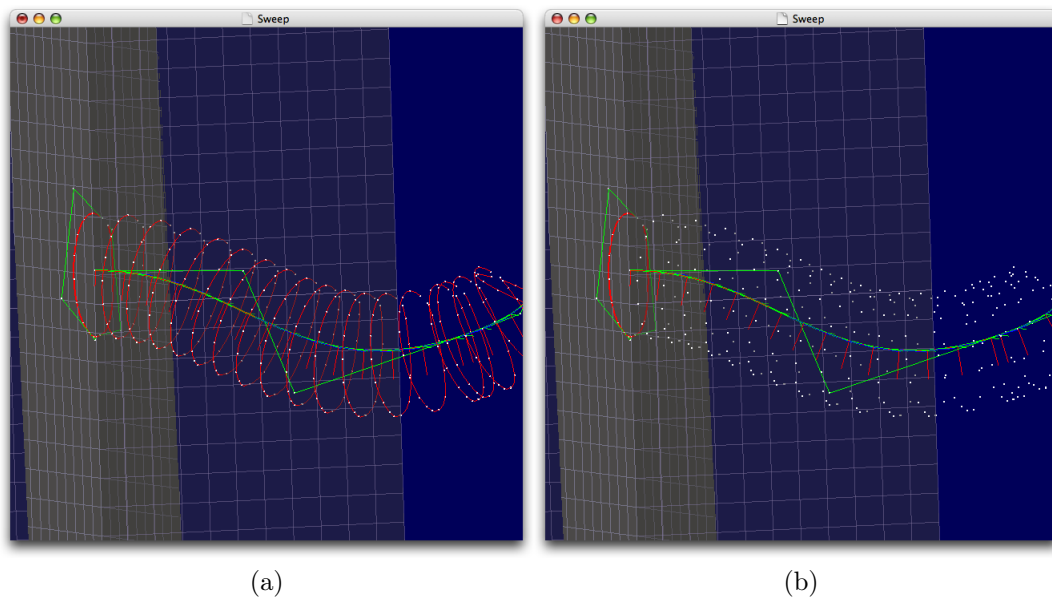
**Figure 7.8.** Based on the user's input strokes (left), b-spline curve representations are generated. Along the path curve (right), Frenet frames are generated, one in each step of the iteration.

by the current frame. This preserves the difference between the two transformations, but removes any similar movements, thus repositing the transformation so that it is centered at, and relative to the origin of canonical space. Finally, because the resulting transformation is only relative to the previous transformation, we need to accumulate each of these incremental transformations into a single transformation that will position the current profile. This entire process is performed using the matrix multiplication,

$$A_i = (F_c \cdot I_p) \cdot A_{i-1} \quad (7.30)$$

where  $(\cdot)$  denotes matrix multiplication. Notice that the multiplications are performed in reverse order from the description above. This is to account for OpenGL's column order matrix representation. The order of the resulting trans-

formation will be to transform from  $A_{i-1}$  (the previous accumulated position) then  $I_p$  (the inverse of the previous frame) and finally  $F_c$  (the current frame). The profile curve is positioned by multiplying its control points by the resulting accumulation matrix  $A_i$ .

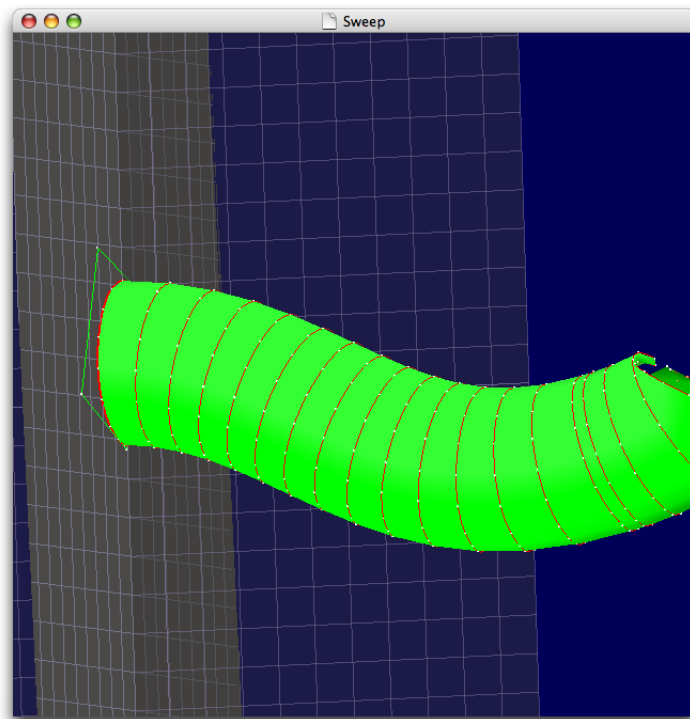


**Figure 7.9.** Contour curves are generated as copies of the user’s initial die stroke, and positioned along the path curve (left)—one in each step of the iteration—using the Frenet frames generated in the previous step. By evaluating these contour curves, surface points are generated (right).

Once the profile has been correctly positioned, it can be evaluated over each of the parametric parameters  $t_j$ —see Figure 7.9. This generates an ordered list of 3-dimensional points along the surface of the modeling component. The list of points is collected into a simple polyline, and used to extrude the leading edge of the mesh structure using the `ExtrudeWireEdge` pseudo Euler operator defined in Section 7.3.2—see Figure 7.10. The iteration process continues until all of the path curve’s parametric values have been processed. The result is a mesh structure



positioned at canonical coordinates and defining the geometry and topology of the modeling component. Along with the mesh structure is stored the orienting frame extracted from the die stroke's drawing plane. This frame is used to position the modeling component in relation to the user's die and path strokes at rendering time.



**Figure 7.10.** The points generated from the contour curves form the basis for strips of triangles which actually form the surface of the modeling component.

## 7.5 Generalized Cylinder

The generalized cylinder based construction method provides the user with a more expressive alternative to the 3-dimensional sweep described in the previous section. Like the sweep, the cross section of the cylinder is defined by the user's

active die stroke. Unlike the sweep, the path along which the structure is constructed is defined not by the active path stroke alone, but by an average of the active path stroke and active size stroke. The relationship between the path and size strokes is also used to uniformly scale the cross section of the cylinder along its averaged extrusion path, and to derive the orientation for each contour.

The construction of the generalized cylinder is very similar to that of the sweep, and works with a nearly identical iterative algorithm. A number of contour curves are generated from the die stroke, and then aligned and oriented along the averaged extrusion stroke. Also like the sweep, the results of this construction process are captured as a triangle mesh, which is then used to represent and display the modeling component.

### 7.5.1 Stroke Averaging

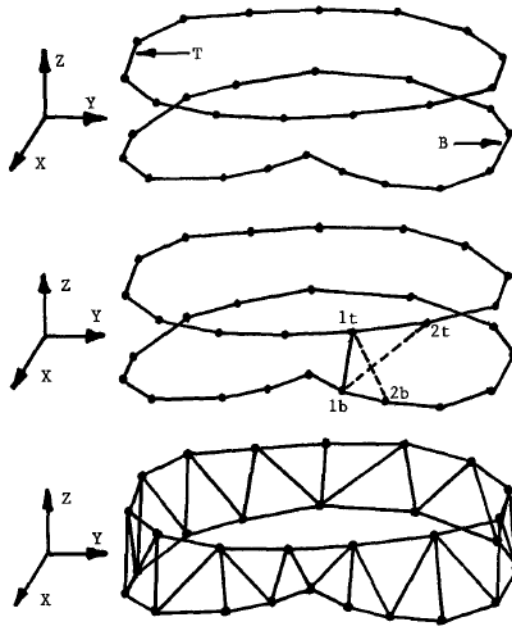
Although the contour curves of the generalized cylinder are not aligned to a single sweep path as in the sweep construction method, a sweep path of sorts called the *extrusion path* still functions as a guide to the construction process, and provides the translational component of the contour placement. The extrusion path is an artificially generated stroke that forms an average path between the active path stroke and active size stroke. The generation of this path may seem straightforward at first, but it turns out that the method of averaging the two strokes has a large effect on quality of the resulting extrusion path.

Because both the path stroke and size stroke were originally defined by discrete sample points, and because the distribution of those sample points along the path of the stroke reflects, to some degree, the dynamics with which each segment of the stroke was created by the user, the initial implementation of the curve averaging

system was design to work directly with this raw point information. Because of the dynamics of the user's stroke, and the manner in which the data is collected, even strokes with relatively similar visual lengths can have drastically different sample point counts. Furthermore, even when the point counts are similar, the relative distribution of points along each curve can be very different. These properties quickly rule out any straightforward point-by-point averaging method.

One solution to this problem is presented by Christiansen and Sederberg [Christiansen & Sederberg, 1978]. In a 1978 paper, they describe an algorithm for generating polygon mosaics between successive contours defined as polylines where in each contour may have different lengths and contain an arbitrary number of nodes (vertices). Refer to Figure 7.11. The idea of the algorithm is to generate a series of edges between nodes of each successive pair of contours to describe a smooth and efficient triangulation of the surface strung between them. Although our goal was not to generate a surface between the path and size stroke, the desired average stroke does lie on the resulting surface, mid way between the two strokes. Thus, it was hoped that this algorithm could be modified to address the current problem.

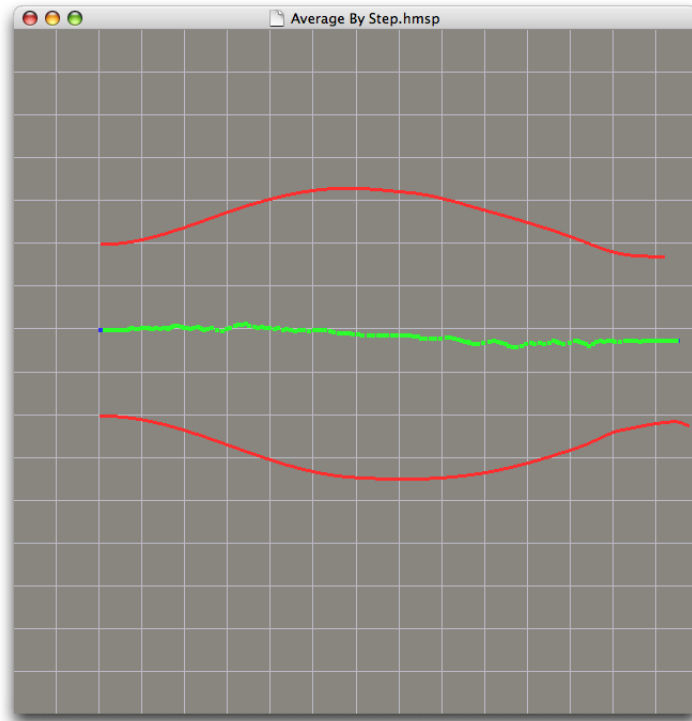
The modified algorithm is referred to as *averaging by step*, and works like this. To begin, a line segment is drawn between the first sample point of the path stroke and the first sample point of the size stroke. These two points become the current points, one on the path stroke and one on the size stroke. Then in each succeeding step, two additional line segments are generated, one from the current path stroke point to the next size stroke point, and one from the current size stroke point to the next path stroke point. The lengths of these two lines are compared, and the longer of the two is rejected. The base of the shorter line then becomes the current



**Figure 7.11.** Demonstration of creating a polygon mosaic between two closed polyline loops using Christiansen and Sederberg’s algorithm from the author’s publication [Christiansen & Sederberg, 1978].

point for its associated stroke, and the process is repeated. This continues until the sample points from both lines have been exhausted. The midpoints of each bridging line segment then form a new ordered list of sample points, which are input to the curve fitting system to generate the average extrusion path stroke.

The method derived from Christiansen and Sederberg’s algorithm was able to generate a stroke that followed a roughly average path between the path and size stroke. However, the resulting stroke contained numerous bumps, wiggles, and other imperfections that make the average path unsuitable for use as an extrusion framework. It appears that the midpoints of each line segment contain too much deviation to make an effective averaging stroke. An example of the result of the algorithm is displayed in Figure 7.12. After numerous adjustments and repeated testing this method was abandoned.



**Figure 7.12.** This diagram shows two strokes averaged using the averaging by step algorithm derived from Christiansen and Sederberg [Christiansen & Sederberg, 1978]. Notice that although the generated points in green form an average path between the two outer curves, the path they describe is bumpy and full of discontinuities.

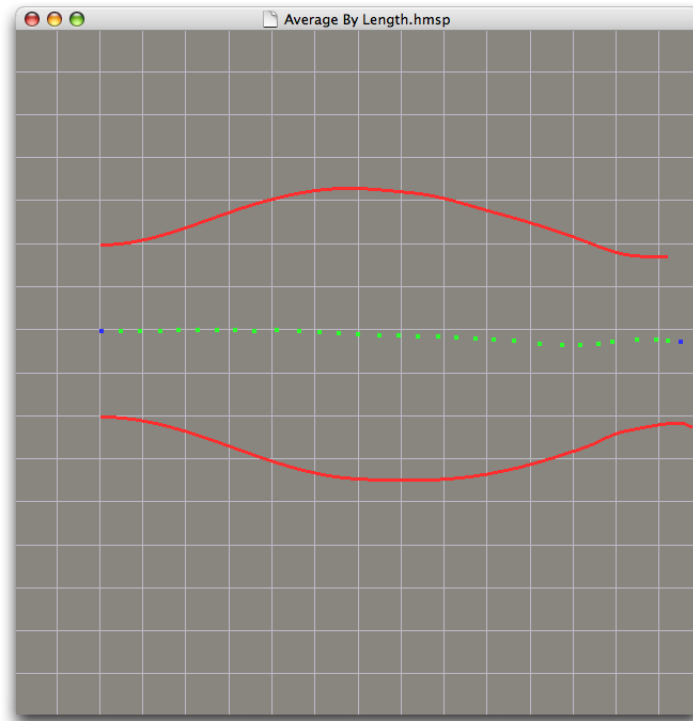
For an alternative method, rather than focusing on the raw input data it was decided that a more effective approach might be to deal with the strokes fit to that data. Although the path and size strokes may have different orientations, lengths, and chain structure, because they are used to define the silhouette outline of a 3-dimensional shape, it is assumed that the strokes follow a roughly parallel path in which the strokes move in roughly the same direction, and that the initial and final control points are positioned in relative proximity. This is consistent with silhouette lines of a 3-dimensional object, which may take arbitrary paths to define the sides of an object, but must align with a set of planar end caps such as the top and bottom of a vase.

Based on this assumption, a new algorithm referred to as *average by length* was devised, and functions as follows. The lengths of the path and size strokes are measured using the Bézier length algorithm defined in Section 7.2.4. The longer of the two lengths is selected and then divided by a quantization factor to calculate a sample count. Each stroke is then converted to an equivalent b-spline representation using the algorithm from Section 7.3.1, and evaluated over the calculated number of evenly spaced samples. The resulting sample points are matched between the two curves, and corresponding points are averaged. These points are then input to the fitting system to generate a new average extrusion path stroke. An example of the result of the algorithm is displayed in Figure 7.13. By maintaining a quantization factor that generates points with generous spacing—approximately 4-5 times the average spacing of user generated sample points—the fitting system is able to generate an acceptably smooth extrusion path.

Once the strokes have been averaged, just as in the sweep construction system, the generated average extrusion stroke has its length measured and recorded, and is then converted into equivalent b-spline representations, which will be referred to as the extrusion path curve.

### 7.5.2 Alignment Frame and Scaling

Just like the previous sweep based construction method, the generalized cylinder uses reference frames to position and orient a number of contour curves along the path of the extrusion. However, unlike the previous method, which only had a single path from which to derive orientation information, the generalized cylinder has two paths defining the outer silhouette of the 3-D shape. This extra path's worth of information provides the system with an alternative to the Frenet-Serret



**Figure 7.13.** This figure demonstrates the results of averaging the same strokes from Figure 7.12 using the average by length algorithm. Notice that in this case the generated points form a much smoother path.

frame.

Recall that a reference frame is composed of three orthonormal basis vectors that can be arranged into a triad centered at a point on the parametric path. The three components of this triad are the unit tangent vector, which points along the path of the curve; a unit normal vector, which point in a perpendicular direction to the curve; and a binormal vector, which is orthogonal to the other two components. As in the sweep algorithm, normalizing the first derivative of a path curve provides the tangent component of the frame, although in this case the generated average extrusion path curve is used rather than using the path curve directly. The major difference between this and the Frenet frame based

method is in the calculation of the other two components. Rather than generating a second vector from the second derivative of the extrusion path curve, Cherlin suggests that when silhouette strokes are available, normalizing the vector between corresponding points on the two curves gives an effective roughly orthogonal vector to the curve [Cherlin *et al.*, 2005]. The resulting vector takes the place of the binormal of the reference frame triad, and the part of the normal is played by the cross product of the replacement binormal and unit tangent vectors.

This alternative reference frame construction method has four distinct advantages over the generation and propagation of Frenet frames. First, as long as the silhouette curves do not intersect at corresponding locations, the vector between the two curve points is always defined. This means that there is no need to generate an initial frame and then propagate that frame along the curve to avoid discontinuities in or disappearance of the second derivative. Second, because propagation can be avoided, the frame can be calculated directly for any location along the extrusion path. It's not necessary to generate an initial frame and then propagate it along the path to find the frame at a desired location. Third, because the orthogonal vector to the extrusion path is generated from the difference of the two curves—a geometric property that is visually obvious and intuitive to the user—rather than from the curvature of the stroke—something that can, at times, be difficult to visualize—the resulting orientation and thus behavior of the 3-D construction is more predictable to the user. Fourth and finally, because the system's behavior is more predictable and the construction derives from an easily manipulated property of the user's input, the user has greater control over the construction process as a whole.

Along with alignment information, at each evaluated location along the extru-



sion path, the profile curves defining the 3-dimensional surface are also uniformly scaled. Like the curve's orthogonal vector, the scale factor is derived from the same vectors between corresponding points on the path and size curves. First, a reference distance is measured between the initial points of the path and size curves. Then, at each parametric value along the extrusion path, the distance between corresponding parametric values on the path and size curves is calculated, and normalized with respect to the reference distance to generate a scale factor. The scale factors are applied to each contour curve to adjust the diameter of the cylinder in sync with the silhouette lines. Each scaling transformation is centered at the initial point of the average extrusion path. When the path and size strokes are drawn as silhouette lines, the average extrusion path generally falls in the middle of the die stroke's cross sectional shape. This has the effect of centering the scale in the middle of the die so that the cross section appears to shrink and grow about its central axis.

### **7.5.3 Generalized Cylinder Construction**

Just like the sweep construction, the generalized cylinder construction algorithm can be broken into a preparation phase, and a simple iteration, each consisting of basically the same five steps. Once again, the preparation steps are:

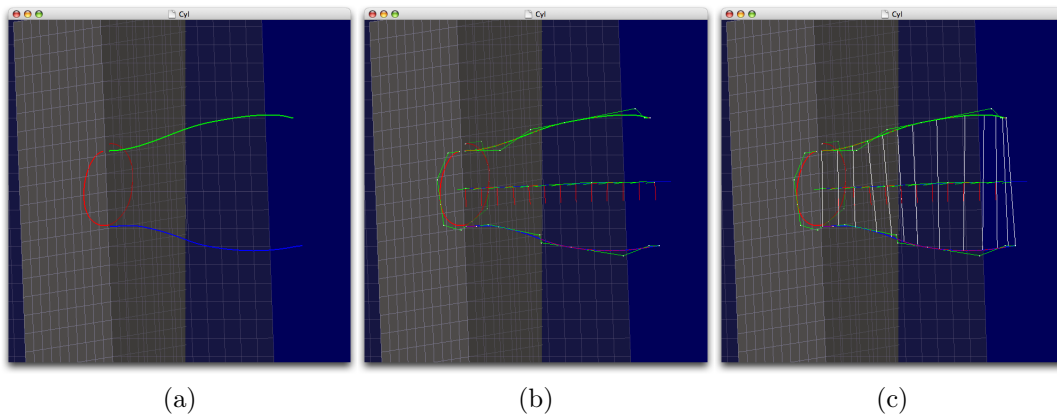
1. Calculate the stroke lengths.
2. Convert the input strokes into construction curves.
3. Orient the construction curves.
4. Generate the parametric parameters that will define the surface.
5. Prepare for the iteration.

And similarly, the iteration process involves the following steps:

1. Generate the alignment frame.
2. Generate a profile curve.
3. Position and scale the profile curve.
4. Evaluate the profile over the parametric parameters.
5. Extend the mesh with the resulting surface points.

Recalling the sweep algorithm on page 379, the construction process proceeds in nearly the same manner as the previous algorithm. The input strokes, including the average extrusion stroke, are all measured and then converted to b-spline representations. The curve versions of each stroke are then reoriented to place the die curve in a canonical position. The next step is to generate parametric values at which to evaluate the curves. Parametric values are generated for both the die curve, and the average extrusion path curve. Finally, the system's state is initialized by creating storage for the new mesh structure, and transformation matrices that will hold the positioning information, just as in the sweep algorithm. In addition, the system also prepares storage for one additional transformation matrix  $S_c$ , which will hold the current scaling transformation.

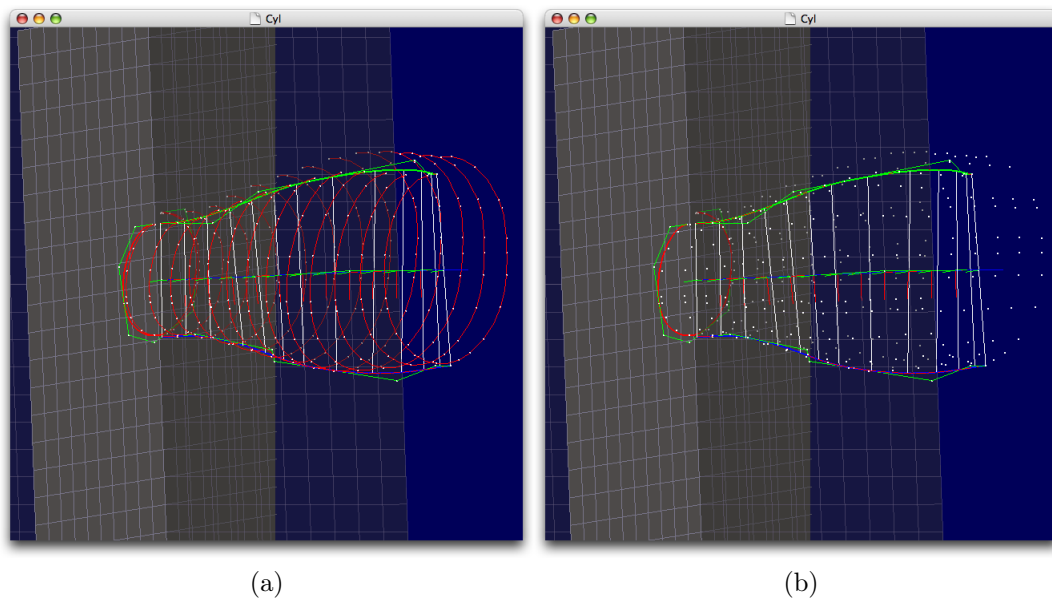
Once the preparations are complete, the iteration process begins. As before the iteration is performed over each parametric value  $u_i$  generated for the average extrusion path curve—see Figure 7.14. For each parametric parameter a scale factor must be calculated. First, the current parametric parameter  $u_i$  is normalized to the parametric range  $[0.0, 1.0]$  by dividing it by the length of the average extrusion path. This normalized value is then used to evaluate the path and size curves within their individual parametric ranges. This adjustment of the parametric value accounts for the fact that all three curves are likely to have different



**Figure 7.14.** Based on the user’s input strokes (a), b-spline curve representations are generated. An extrusion path is also generated from the average path of the path and size strokes. Along the extrusion curve (b), alignment frames are generated, one in each step of the iteration. In addition to each frame, a scale factor is also derived (c)—here visualized by scaling spans—by matching corresponding points between the path and size curves.

parametric ranges, even if they follow similar paths and have nearly identical lengths. Evaluating the two curves produces two 3-dimensional points, which are then subtracted to generate a vector. The magnitude of this vector is compared with the initial distance between the two curves as described above to generate the scale transformation.

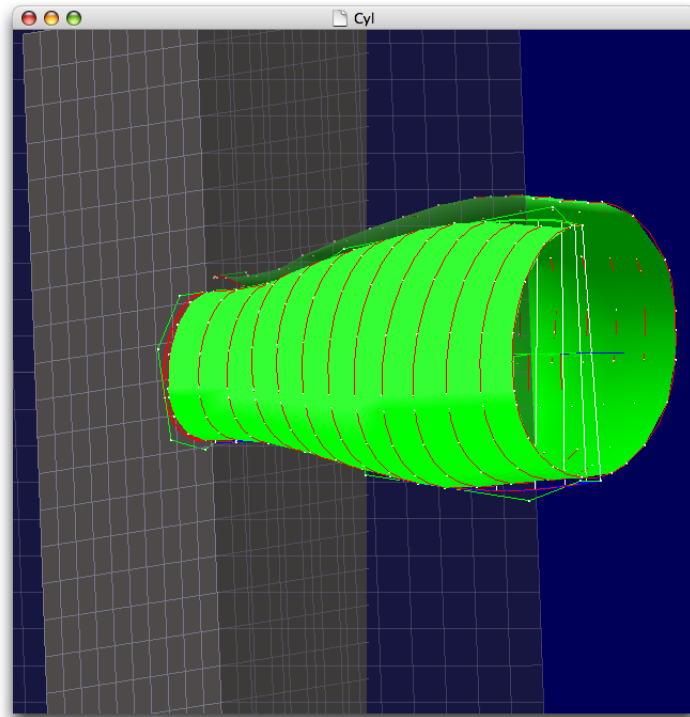
The direction of the same vector, along with the first derivative of the extrusion path curve at the current parametric value are then used to generate the positioning frame. The current, previous, and accumulated transformation are then combined—just as in the sweep algorithm—to generate a transformation that will position the contour curve. However, before each contour curve is positioned, that is, while it is still in canonical position, the contour is first multiplied by the current scaling transformation. Recall that the scaling transformation is centered at the start of the average extrusion path. By performing the scale first, before the



**Figure 7.15.** Contour curves are generated as copies of the user’s initial die stroke, and positioned along the extrusion curve (left)—one in each step of the iteration—using the alignment frames generated in the previous step. Each contour is also scaled using scaling factors derived from the scaling spans. By evaluating these contour curves, surface points are generated (right).

contour is position, the algorithm ensures that the scaling center is maintained relative to the cross sectional shape. Although the positioning transformations are accumulated in the  $A$  matrix, the scale transformation is not, but is instead calculated and applied in isolation for each parametric value.

From this point on, the algorithm is identical to the sweep algorithm—see Figures 7.15 and 7.16. Each profile is evaluated over the set of parametric values for the die curve, and those points are used to build out the mesh structure. As before, the resulting modeling component is then positioned using the orienting frame extracted from the die stroke’s drawing plane.



**Figure 7.16.** The points generated from the contour curves form the basis for strips of triangles which actually form the surface of the modeling component.

## 7.6 Visual Environment

The current application is designed as a prototype and proof of concept. Because of this, visual effects, window dressing, and other aesthetic features of the application and its interface have been kept to a minimum. However in some circumstances, principally aesthetic features can serve a dual role to both beautify the program and its output, and provide the user with more accurate or comprehensible visual stimulus. This additional visual stimulus is especially important in 3-dimensional modeling applications because of the inherent ambiguity in visualizing the 3-dimensional modeling environment on the computer's 2-dimensional display.

One aspect of the three-dimensional rendering process the author has found to be particularly advantageous is the use of an appropriate environmental lighting model coupled with surface lighting effects. Just as an artist adds shading and shadow to a drawing to highlight its three-dimensional shape, these lighting effects visually underscore the 3-dimensionality of modeling components. This section briefly describes the lighting arrangement used by the application. In addition this section describes the implementation of an algorithm used to calculate normal vectors for the modeling components. These normal vectors interact with the environmental lighting model to create more visually realistic shadows and highlights on the surfaces of modeling components.

### **7.6.1 Environmental Lighting Model**

Constructing a lighting system for the 3-D modeling environment is very similar to arranging physical lights in a photographer's studio or on a movie set. As a lighting designer, the 3-D application developer has at his or her disposal a number of light sources, which can be positioned in the 3-D modeling space just like setting up stage lights. Properties of each light such as its color, intensity, attenuation, and interaction with the surface properties of objects in the 3-dimensional scene can then be configured to produce a variety of lighting effects.

For the purposes of the present application, the goal of the lighting model is to provide the user with a clear view of the 3-dimensional structure of each modeling component. Because of the close resemblance to physical lighting design, inspiration for an appropriate lighting environment was taken from the lighting designs used by professional photographers.

When photographers wish to clearly illuminate a subject, a lighting arrange-

ment known as the *three-point lighting system* is a common choice [Lord & Sibley, 2004]. The three-point lighting system is widely used in portraiture photography, for television interviews, in products layouts, and any other situation where photographers, cinematographers, or animators want to clearly illuminate their subjects.

As its name suggests, the three-point lighting system is composed of three light sources. Please refer to Figure 7.17. The first light source, called the *key light*, is the primary light source for the scene. The key light is a fairly bright white light usually placed just to the right and slightly above the eye line of the camera, and directed towards the subject. The purpose of the key light is to clearly illuminate the subject's features. When used with a human subject, for example, the key light clearly illuminates one half of the subjects face, leaving the other half in shadow.

The second light is referred to as the *fill light*. The fill light is usually placed just to the left of the camera and possibly slightly lower than the key light. The purpose of the fill light is to provide illumination for the subject's opposite side. Although this light illuminates the section of the subject left in shadow by the key light, its purpose is not to fully illuminate this portion of the subject, but only provide a better indication of the features the key light left in shadow. Thus, the fill light is dimmer than the key light, and the light it produces more diffuse. This maintains a contrast between the highly lit portion of the subject on the one side, and darker regions on the opposite side, while allowing the viewer to see features of the model in both areas.

The final light is the *back light*, and is usually placed some distance behind and above or below the eye line of the camera. As Lord and Sibley explain, "The

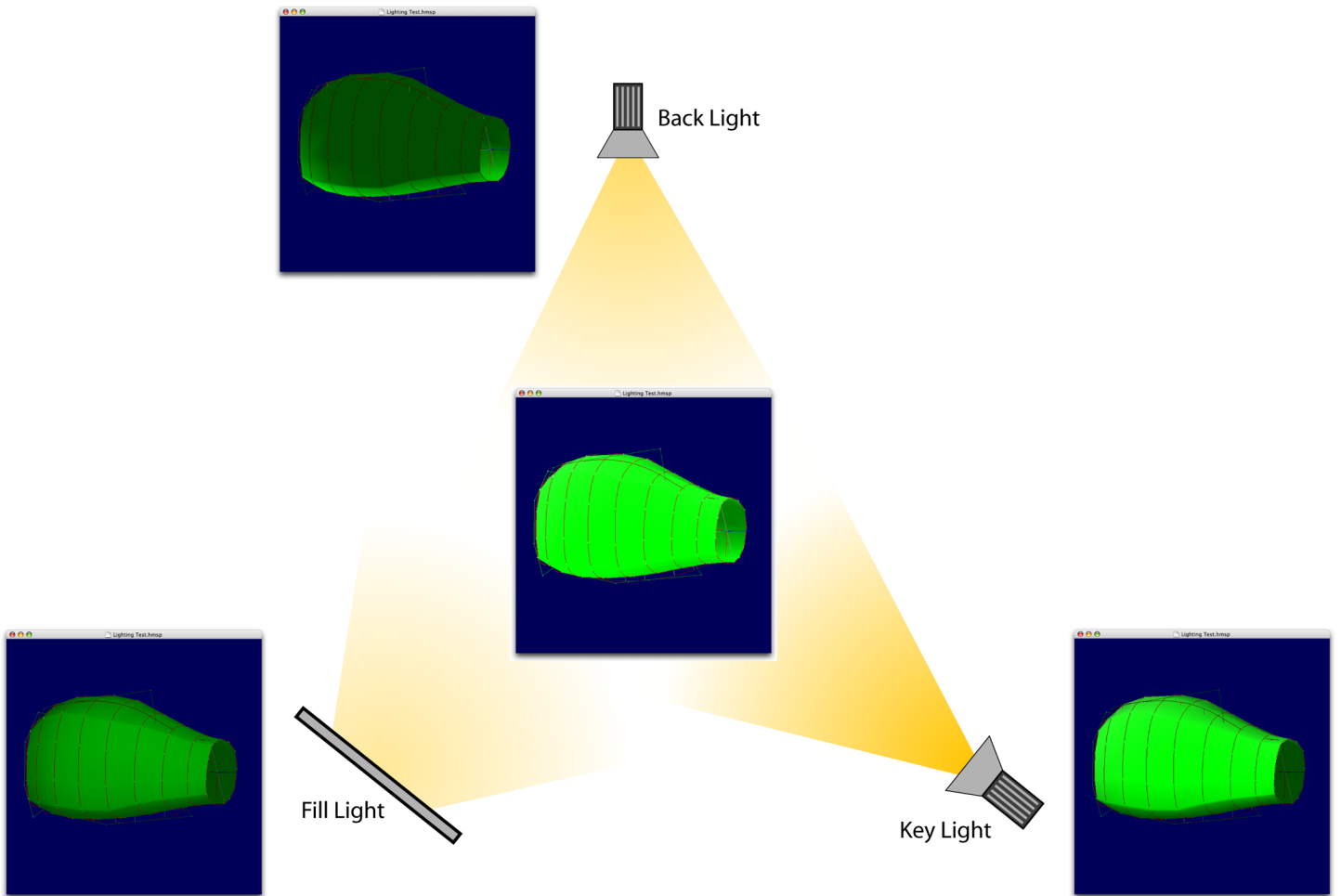
back light is used to ‘rim’ the subject with light, producing a highlight along [its edge] and separating the subject from the background. This is often called giving the subject ‘an edge.’” [Lord & Sibley, 2004].

To illuminate the 3-dimensional modeling environment, a version of the three-point lighting system is used. A key light, fill light, and backlight, are placed to the right and left of the viewpoint, and to the rear of the model respectively. In addition to these fixed light sources, a single low-key ambient light source is also used to provide a general low level of illumination to all parts of the modeling environment. The positions of these light sources are fixed with respect to the camera, meaning that as the user reorients his or her viewpoint of the 3-dimensional modeling environment the light sources move in unison with the camera. All four of these light sources are activated by default to illuminate the modeling environment, however the user can individually toggle the key light, fill light, and back light with a simple keyboard commands. In some instances, adjusting the light sources by hand in this way can be helpful in making sense of the 3-dimensional view.

### **7.6.2 Surface Properties**

The position and attributes of light sources within the modeling environment are only part of the lighting equation. The properties of the light source determine from which direction light enters the environment, and partially explains how the light will interact with the modeling components. The other half of the equation is determined by properties of the surfaces themselves. The color and quality of the light reflected from the surfaces of a modeling components is determined by each component’s material properties. For the modeling components in the present





**Figure 7.17.** This diagram illustrates a three point lighting arrangement from overhead. The subject being lit would be placed in the center where light from the three light sources intersect. Next to each light source is an example rendering from the application in which a subject has been lit using only one of the three lights. The image in the center illustrates how the lights combine to provide a good view of the subject.

application the material properties of the surfaces are fairly simple. All modeling components are rendered with the same moderate surface qualities, striking a balance between a mat and specular finish. The color of each surface is determined by the inkwell settings in the user interface, and default to a pleasant green color. A second and more complex component of the lighting model associated with each surface that surface's is normal vectors.

Normal vectors are directed outward from the model, orthogonal to its surface. Within the mathematical lighting model used by the OpenGL sub system, surfaced normals are compared with the direction of the light source and the viewing angle to determine the quality and intensity of light reflected from that point on the surface. Unlike the material properties, which are defined over the entire surface, the normal vectors vary as the direction of the surface changes. In a completely realistic model, normal vectors would vary continuously over the entire surface. However in this application the surface is not perfectly smooth, but is instead represented by a mesh of small triangular facets. The modeling system simulates the appearance of continuously varying surface normals by calculating normal vectors for each of the triangle vertices and then allowing the graphics hardware to interpolate the normal values over the area of each facet. This sort of normal interpolation is referred to as smooth or Phong shading [Angel, 2003] [Phong, 1973].

Because each modeling component is defined by a parametric process, and surface points are derived by evaluating parametric contour curves, one possible method of calculating approximations of the normal to the surface at arbitrary locations would be to calculate the gradient of the surface at each location [Schroeder *et al.*, 2006]. Unfortunately, because the surfaces are not defined by a

single expression, and because the surfaces may contain discontinuities generated by corners in the user's input strokes, direct calculation of the gradient could be difficult. An estimation of the gradient that is often used in volume rendering applications is to calculate the forward, backward, or central difference of the surface at each location [Schroeder *et al.*, 2006]. Although this method may be viable, because the surfaces in this situation are represented in memory by polygon meshes that contain both geometric and topological information, a much simpler algorithm is available.

The calculation of the vertex normals is a two-step process. First, for each triangular and polygonal facet within the mesh a single normal vector for the entire face is calculated. Because each face of the mesh structure is assumed to be planar, a normal vector for the face can be calculated as the cross product of any two nonparallel vectors that lie within that plane. A simple source for these vectors is the boundary of the face—specifically any two successive boundary edges. Whenever a face is constructed in the mesh using one of the pseudo Euler operators—`MakeFaceKillLoop`, `MakeFace`, or `ExtrudeWireEdge` as described in Section 7.3.2—two successive edges of the face are selected and their cross product calculated to generate a normal, which is then stored along with the geometric and topological information in the mesh. The mesh always assumes that vertices or boundary edges used to construct new faces are provided in a counterclockwise order. Similarly, when selecting images for the cross product the sequence of edges around the face are selected in a counterclockwise order so that, observing the right-hand rule, the surface normals always point in the same direction away from the modeling component.

If the system were to stop at this point and provide the face normals directly to

the graphics subsystem the lighting model would properly calculate colors for the surface facets. However, because each face has only one normal, color calculations over the area of the surface facet would all generate the same color, causing each face to be easily distinguishable from its neighbors. This shading model is known as flat shading [Angel, 2003]. Flat shading is more efficient to calculate than other methods, however it does not provide the desired level of surface realism. To achieve the smooth shading effect it is necessary to calculate normal vectors for each vertex.

The next step is performed only after the entire mesh has been constructed. At this point a single face normal is associated with each face in the mesh. To calculate a normal vector for a vertex in the mesh the individual face normals for each face directly adjacent to that vertex are average. This averaging process is immensely simplified by the topological information stored in the mesh structure, and the circulator iteration mechanisms.

The algorithm works as follows. For each vertex in the mesh a face circulator is generated. The circulator iterates over the faces immediately adjacent to the current mesh vertex in a counterclockwise order. For each face, the face normal is collected from the data structure and accumulated into a running total. Once each face has been visited, the total is divided by the number of adjacent faces to generate a new vector averaging the neighboring face normals. The vertex normals are stored contiguously in the mesh structure, indexed by the vertex array. This allows the vertex normals to be passed to the graphics subsystem in a single operation, just like the geometric information defining the mesh vertices.

At rendering time the graphics subsystem will automatically interpolate the vertex normals across the adjacent faces of the mesh. Visually this smooth shading

will give the appearance that each of the flat facets of the mesh is in fact part of the smoothly curving surface. Not only does this make the three-dimensional structure of the modeling component's surface more visible to the user, it also causes the faceted surface of the model to appear smoother than it actually is. This allows the surface to be drawn with far fewer triangles than might otherwise be necessary to make the surface appear smooth, improving both display and memory performance.

# Chapter 8

## Conclusions

This research effort began with three specific observations. First, examining the current crop of sketch-based modeling interfaces, it was observed that many interfaces attempted to mimic specific traditional sketching, drawing, or sculpture techniques as the basis for construction of 3-dimensional objects. Although this approach provides artists and designers with the appearance of a familiar interface, because these methods were not originally intended for the creation of 3-D computer models, the 3-dimensional information they provide is often impoverished, and their application to a standard computer interface is difficult. Second, where sketch-based modeling interfaces did provide a means of reliably imparting or extracting three-dimensional information from the user, they often failed to address the fact that although the product of the modeling operation is a 3-dimensional subject, the user's experience of that product and an entire modeling process is necessarily 2-dimensional because of the limitations of commodity personal computer input and display hardware. Third, although many sketch-based modeling interfaces provide the user with a familiar drawing-like experience through the use of a digitizing tablet, most choose to utilize only a small subset of the dynamic

physical information provided by this device.

Addressing the first of these issues, the author hypothesized that if a sketch-based modeling method were derived from the mental processes that underlie traditional sketching technique, and based around the way in which artists *think* about creating 3-dimensional forms in 2-dimensional drawings, the resulting interface could provide a more natural and intuitive method of creating 3-dimensional models. To investigate this hypothesis, the author performed an in-depth analysis of traditional and computer-based sketching practices—examining common tools, techniques, and the psychological foundations of the sketching process among professional artists, designers, engineers, and laypersons. The author also examined a variety of current computer-based modeling methods, including conventional modeling interfaces used by commercial and entertainment modeling software, as well as a representative sampling of research projects in the sketch-based modeling arena.

Based on this analysis, a set of design principles and application requirements were developed that formed the basis of the design for a prototype modeling system and research platform. The result of this process is Hammerspace, a three-dimensional modeling application that uses a sweep and generalized cylinder-based construction method to convert the user's 2-dimensional drawing strokes into 3-dimensional modeling components. This modeling method is designed to mimic the way traditional artists break down and reconstruct three-dimensional forms in terms of their basic shapes and structures, and how those shapes are then rendered using basic sketching techniques.

In response to the second observation, although the modeling application presents the user with a full 3-dimensional environment in which his or her 3-

dimensional models are constructed and manipulated, the user's interaction with the environment is through a conventional and familiar 2-dimensional sketching idiom. The user draws 2-dimensional strokes with a digitizing tablet and stylus, which are in turn projected onto 2-dimensional drawing surfaces within the modeling environment. These strokes then form the basis of 3-dimensional constructions in the same way that an artist places foreshortened and perspective strokes on imagined planar surfaces within a traditional 2-dimensional drawing to form the illusion of 3-D structures. This interface not only provides the artist with a familiar approach to the way he or she would think about constructing a 2-dimensional drawing of the same subject, but extends that experience into a 3-dimensional process through which the user's dynamic control of the viewpoint and drawing surfaces simplify and enhance the drawing process.

Addressing the third observation, the author developed a set of novel physical gestures utilizing the dynamic pressure, hover, and tilt information provided by the digitizing tablet hardware. These gestures are based on natural physical movements commonly performed in the process of sketching. Some of the gestures provide natural and intuitive channels for control of the three-dimensional environment through the two-dimensional interface of the drawing tablet, while others simply reflect innate motions of a pen or pencil harnessed as quick and efficient controls for common interface tasks.

The remainder of this chapter discusses a brief evaluation of each of the components developed for this research effort. The next section contains a discussion of preliminary results and evaluation of the sketching interface, system of tablet gestures, the user interface, and the 3-D construction and modeling methods. Following this are three sections that briefly enumerate the specific contributions of



this work, a number of limitations in its current implementation, and possible avenues of future investigation.

## **8.1 Discussion**

Given the user-centric nature of sketch-based modeling in general, and the modeling and interface features of the application developed in this research effort in particular, assessment of the efficacy of the modeling system, sketch based interface, and tablet gesture functionality would ideally be performed through structured or real-world user studies with test subjects in appropriate backgrounds for the target audience of this application. While studies of this kind are most definitely a target of future work, because the current application was designed as a platform for testing modeling methods and interface mechanisms, it lacks some of the basic functionality that would be required in a professionally useful modeling tool. Time limitations for the current study also prevented smaller scale structured user testing. However, the author performed a great deal of informal testing and qualitative assessment during the development and implementation process. Coupled with informal discussions and demonstrations with potential users this information provides a preliminary assessment of the modeling methods and application interface features. The following sections discuss each aspect of the system in turn and provide some initial feedback.

### **8.1.1 Sketching Interface**

For the user, the primary focus of input for the modeling application is the tablet interface. Thus, to a large degree, the quality of the user's experience hinges on how well he or she can work through the tablet to achieve the desired result. As

outlined by design principle I in Section 6.2.1, the primary function of the tablet is as a sketching system, and so this is a logical place to start our assessment.

Creating strokes with the system is very straightforward. Signaled by the familiarity of the stylus as a drawing implement, potential users quickly grasp the concept of using the stylus in this manner and are eager to try their hand. For users inexperienced with the use of a digitizing tablet, the disconnect between drawing on one physical surface and seeing the results on a separate screen can be disconcerting. However as noted in Section 2.4.1, this is a common reaction, and it has been the author's experience with other tablet applications that after a short period of practice—usually on the order of minutes—most users can develop at least a basic rapport with the device.

As discussed in Section 6.4.6, as the user creates a stroke, the stroke fitting system displays the user's pen path as a series of discrete points that are then transformed into the parametric stroke representation in a single operation when the user lifts the stylus from the tablet's surface. There was some concern, especially in early testing, that the use of discrete dots to represent the nascent stroke might be an inadequate visualization of the curve. These concerns drove repeated efforts to incorporate an on-the-fly fitting system. However experience now shows that these fears were unfounded. The discrete visualization appears to be effective on several levels.

First, by drawing the points as white dots against the dark backgrounds of the drawing surfaces and saturated primary colors of the other environmental components, the nascent stroke is easily visible, even over a great deal of visual clutter. Furthermore, as was predicted, because the points are laid down with high frequency along the path, the user's mind has no trouble visualizing a smooth

curve—in gestalt terminology, a ‘closure solution’—that connects the dots and temporarily suggests the user’s stroke.

One area of stroke creation that turned out to be a bit tricky is the use of the OPTION key while drawing to force a closed path. In informal testing, it appears that forming closed die strokes is often the desired operation. Unfortunately, because the OPTION modifier is overloaded—controlling the drawing plane adjustment tool when outside of a stroke, and stroke closure while a stroke is being performed—it takes a degree of coordination and pre-planning to be ready to hold the key during a stroke. Experience with this operation suggests that this can be difficult and even frustrating.

The choice of the OPTION modifier was made because of the addition of modifier proxy keys on newer tablet models. The aim was to place commonly accessed functions on those keys so that the user could avoid trips to the keyboard, or if forced to use one, could limit that access to the familiar grouping of modifiers. In this case however, it appears that closing a stroke requires a more isolated interface method. It may be necessary, for example, to employ a separate key, modal switch like CAPS-LOCK, or perhaps a gestural command. The examination of possible alternatives is a target for future development.

Another issue that raised some concern during the development process was the use of different stroke pens to signify the interpretation of each stroke to the system. A number of alternative methods were tested during the development process including the use of a *selection system* in which the user would explicitly select and designate strokes, and a *stack system* in which the user would ‘load’ a stroke onto a stack after its creation and then empty the stack with a construction operation. Compared to these other options the pen system seems to be the most

intuitive, avoiding the puzzled looks elicited by the other systems when the author attempted to describe their use.

Although the logical interpretation as a ‘tool selection’ may help the user to understand the system, it’s the author’s belief that once the user has a grasp of the interface it’s the use of key colors to mark both the active strokes and the cursor in the environment that make the system most productive for the user. Viewing these colors the user instantly latches onto the system’s interpretation and can easily predict the results of each operation.

During testing, key colors and their quick recognition highlighted another issue. In the zeal of drawing, a common mistake is to first create a die stroke, and then draw a second stroke intending it as a path or size stroke, but forgetting to switch pens first. The mistake is instantly recognized once the stroke is complete and its fitted form is displayed with the wrong color. The application’s undo facility is minimally sufficient to correct the mistake—the stroke can be undone, the tool switched, and then the stroke redrawn. However, as with any dynamic input, it can be difficult to exactly redraw a stroke. If the initial but mistaken stroke is precisely what the user wanted to create, there is currently no way to make the correction without losing the stroke in the process. The realization of this error can be very frustrating. Although additional feedback could be provided to help the user avoid this situation in the first place, this issue highlights the need for a means of explicitly specifying a stroke’s interpretation *after* the stroke is created.

Through the testing process, one additional aspect of the pen system raises some concern. As described in Section 6.4.7, because only one stroke can be designated the current or active stroke for any given pen, when additional strokes are created, previous active strokes are de-emphasized, allowing their use as sketch-

ing elements for the artist's benefit. These visually extraneous constructive and contemplative lines are a characteristic feature of traditional sketching, and so it was very important to provide some access to this behavior in the interface. The design of this system took its cues from the practice of cleanup artists, who trace over the sketchy lines of animators to create clean drawings suitable for later stages of the animation process, but the idea is present even in general sketching. As we noted in our discussions of the sketching process, as a sketch progresses, the use of constructive or contemplative strokes diminishes, making way for more structural elements. This behavior is reflected in the system in the way that the active stroke migrates from earlier input to the latest stroke of the pen.

Despite the presence of these factors in traditional sketching, in working with the application using the strokes in this 'sketchy' capacity feels unnatural. While using the system to generate these inactive strokes is not hindered in any way, the feature is underutilized. The author has found, for example, that more often than not there is an urge to undo initial or partial strokes and create a clean line rather than utilizing this feature. Given the prevalence of this activity in traditional sketching and its link to sketch's mental components, this preliminary reaction warrants a degree of consideration. After a careful examination of the interface two possible contributing factors have been identified.

The first is the fact that, because all strokes pass through status as an active stroke, in the user's mind they attain a feeling of permanence or emphasis, a characteristic that is probably underscored by their visual depiction as bright, primary colors that stand out against other components of the interface. Although the strokes dim to gray once they become inactive, each begins in this very active state, and so users may approach the use of the pen in the same way that one

approaches drawing with a pen as opposed to a pencil. The knowledge that the pen's mark is permanent discourages the creation of frivolous strokes that may mar the final drawing. The idea that the mark is concrete and meaningful to the system, even if it's only at first, encourages the user to contemplate each stroke carefully and prefer erasure and correction to sketch-like accumulation and refinement.

A second consideration may be the behavioral characteristics of the stroke system in comparison to pencil-and-paper sketching. In Section 6.4.5, we noted that a vector graphic representation was selected for strokes in the system. This mathematical representation provided a number of benefits from an algorithmic perspective. We also noted that in contrast, raster formats—that is, pixel based representations—are often used by 2-dimensional art packages because of their ability to mimic natural media, but are a difficult fit for algorithmic input because of their discrete representation. From an interface perspective the vector graphics format was justified as a means of steering the user away from counterproductive 2-dimensional natural media techniques and more towards the construction of 3-D objects. It is still the author's belief that the final cleaned strokes used for 3-D construction should follow this philosophy and be represented in vector graphic terms. However, the exacting and definite visual depiction of these formats as clean connected lines with sharp, crisp features may be contributing to the problem.

To address these issues, several alterations are under consideration for the future development of the stroke interface. First, it seems prudent to separate the creation of sketch-style strokes from the active strokes intended for interpretation by the system. An obvious possibility is to split this functionality off into a

completely separate tool, although this has the unpleasant side effect of requiring additional modal switching by the user. Another option is to create all strokes in their de-emphasized form from the beginning. Once a stroke is drawn the user could then mark it as active if desired with a simple click or other mechanism. This would avoid passing each stroke through an active status and give all strokes an initial equal footing as sketch lines.

Coupled with these options, it may also be advisable to consider the visual depiction of the strokes. We briefly discussed NPR—non-photorealistic rendering—in conjunction with computerized animation systems. This field of research applies special rendering algorithms and techniques to 2-D vector graphic and 3-D geometry images that make them appear more like drawings, paintings, cartoons, or other artistic media rather than the photorealistic renderings usually targeted by 3-D graphics systems. Several sketch-based modeling applications also apply these techniques including [Cherlin *et al.*, 2005] [Igarashi *et al.*, 1999] [Masry *et al.*, 2005] [Cuno *et al.*, 2005] [Karpenko *et al.*, 2002] [Karpenko & Hughes, 2006] [Levet *et al.*, 2006] [Bourguignon *et al.*, 2001] [Tolba *et al.*, 2001] [Ohwada *et al.*, 2003] [Zelevnik *et al.*, 1996]. Pertinent to the present discussion, Hsu and Lee describe a system called skeletal strokes in which a stylized stroke image such as a representation of a pencil line could be stretched and fit along the skeletal framework of a vector graphic curve [Hsu & Lee, 1994]. Using such a system might provide the user with the more impermanent feeling of a pencil stroke.

Although the application of NPR techniques may address visual issues with the current stroke system, it's more likely that the issue has more to do with the behavior of the vector graphic strokes. No matter how well the strokes are disguised by various rendering techniques, the mathematical curve representations

lack the ability to accumulate and meld into one another in the way that natural sketching lines can. Thus, it might be fruitful to explore how raster graphic techniques could be applied.

Because of the algorithms used by the construction system, strokes used for input to the 2-D to 3-D conversion system would still need to have a vector graphic representation. However it might be possible to provide the user with a separate raster sketching tool as well as the vector based stroke tool. Alternatively, strokes could be maintained internally in both representations, or in a format that partially abstracts the need for pixels but is not as stringent as a parametric curve. The intention is still to guide the user into utilizing the application's features for 3-D construction, rather than falling into heavy use of 2-dimensional techniques. However it appears that the current system may have swung the pendulum too far in the direction of representations beneficial to the 3-D components.

### 8.1.2 Tablet Gestures

Aside from the creation of strokes, the tablet also provides the user a means of interacting with other functions in the system through the use of tablet gestures. As described in Section 6.4.4, three tablet gestures were implemented as part of the application interface: a *lift-and-lead* gesture to control camera motion within the modeling environment, a *pounce* gesture to access the crease function of the drawing plane system, and a *flick* gesture to control switching between the available stroke pens. Let's address each of these systems in turn.

First examining the lift-and-lead gesture, based on preliminary testing this interaction method appears to be very promising. Because the author is right handed, while holding the pen in a natural grip the system detects lift-and-lead



input as translational commands to the camera. In this capacity, moving around the modeling environment and positioning the camera for easy access to a drawing area feel very fluid and natural.

The same can be said of zooming operations, which are activated when the pen is held at a vertical angle. Moving the hand into this position is out of the normal grip for writing or drawing, but is still well within the natural motion of a pen or pencil. Furthermore, because the zoom operation is used with less frequency and most often in between drawing operations, moving slightly out of a normal drawing position is not a hindrance to the natural flow of work.

Tilting the stylus past the vertical mark and into an opposite angle activates the orbiting function of the camera. It's this operation that poses some difficulty for the user. Moving the stylus into this angle range is further outside the normal actions associated with a writing or drawing implement. Motion into this range is primarily by two routes: the user can rotate his or her wrist, or reposition the stylus in the hand taking a so-called 'pointing grip'. Rotating the wrist provides a stable grip on the stylus, but places the forearm in an unnatural position that feels less dexterous. On the other hand—so to speak—repositioning the stylus provides much more natural control, but lessens the user's grip on the pen barrel. Furthermore, this repositioning requires a deviation from the natural drawing grip, and so requires more time and effort to switch back and forth. It should be noted that none of these issues is so great as to cause the rotation function to be unusable. However they do represent noticeable drawbacks to the system that should be addressed.

Several improvements are currently under consideration for this gesture. First, the current design differentiates the different operations assigned to the gesture

based on the angular displacement of the stylus along a single axis—along the x-axis of the tablet and about the y-axis. This configuration was designed to facilitate easy conversion between right- and left-handed users. Although differentiating between the various functions did not appear to be an issue during testing, because a natural grip of the stylus orients its barrel slightly askew with the axes of the tablet surface, it may be possible to gain greater control by measuring angular displacements along the axis of this natural grip rather than the tablet's axes. This would require adjustments for right- and left-handed users, and possibly even personalized adjustments for each user.

Second, in response to the issues with far tilt operations, it may be more comfortable and efficient for the user if the range over which the current input function is selected is compressed to fall primarily on the natural side of a vertical grip, rather than extending over an entire 180° range. Further testing will be required to determine if the user can successfully differentiate between angles within this compressed range, however by providing greater fidelity along the current skew of the stylus the user may have greater control. Overall, the lift-and-lead gesture has proven to be a viable interaction method.

The next gesture to consider is the pounce. In testing the application, the pounce gesture presented the greatest difficulty in the user experience. Despite a great deal of tuning of the detection algorithm in order to find reasonable thresholds and sample interval values, activating the pounce gesture continued to occasionally malfunction. Although no intended strokes accidentally activated a pounce operation, occasionally an intended pounce was misinterpreted as a stroke operation. Thankfully the undo facility of the application provided a means of correcting faults when they occurred. This miscommunication was most common

in the first few attempts at the command at the beginning of a testing or development session, suggesting that a brief warm-up or practice was required to get the proper ‘feel’ for the operation. After these initial tries, activation of the command was much more consistent.

Based on these findings, several revisions to the pounce gesture are called for. First, it may be helpful to provide the user with more feedback about the gesture as it’s performed. In the current implementation, feedback is only provided at the end of the operation by the mode change in the interface. If some dynamic indication in the cursor could describe the pressure level or current sampling frequency this may give the user a greater feeling of control. Another option might be the addition of auditory feedback.

Second, it may be beneficial to allow both a quick activation of the pounce gesture for experienced users, as well as a delayed method for novice users who might be unsure or unpracticed in the user of the gesture. The expert version would work as before, but the novice setting might allow the user to dwell on the high pressure state for a set period of time until some feedback is provided. This would allow practice with the correct pressure level, and a means of ensuring activation when necessary. Ramos *et al.* noted in their study of tablet interfaces that functions based on dwell operations were generally very accurate but scored low on user preference scales because of the wait time [Ramos *et al.*, 2004]. By providing the dwell function as a backup to the faster but less accurate expert pounce mode, users could adjust their use of the function based on the circumstances.

Addressing the issue of misinterpreted gestures, observing readouts from the pounce detection system, it appears that the most common cause of misinterpretation are the frequency counts used to differentiate pounce gestures from simple

strokes. This is especially true of the *startup sample range* and *pounce duration*, which record the maximum number of samples between a pen down and the start of a pounce gesture, and the minimum and maximum number of samples over the pressure threshold respectively. When a pounce is misinterpreted it's usually the case that the recorded input falls close but just outside one of these two parameter ranges. Tsang *et al.* in discussing the development of a sustained pressure switch for their modeling application [Tsang *et al.*, 2004], noted that a personal adjustment feature was required to allow each user to select a pressure threshold that was comfortable from one session to another. Although the pressure threshold in the current application appeared to be appropriate over the span of multiple sessions, it does appear that it may be necessary to allow the user to adjust the frequency counter ranges.

Experience with adjusting pressure and clicking thresholds in the tablet driver's preference pane, as well as adjusting settings in the application during the development process, suggests that it can be very difficult for a user to objectively adjust these values. Providing a simple slider or other widget interface to adjust anonymous values is not likely to provide the user with an enjoyable experience. Furthermore, because this adjustment may need to be performed on a regular basis, a more interactive method is required. An attractive alternative is suggested by a similar pressure adjustment system included in Corel's Painter application [Corel, 2006]. The application provides the user with a preferences dialogue containing a number of numeric sliders, as well as a large drawing area. The user produces several strokes within the drawing region, and in response to that input, the system automatically adjusts the sliders in real time, tuning the system's pressure sensitivity to the dynamics of the user's stroke. A similar system could

be implemented as a preference dialogue in the present application. This would not only provide the user with a fast and intuitive means of adjusting the parameters of the pounce detection system, but could also provide a ‘practice session’ encouraging the user to warm-up and get a feeling for the gesture before using it in the application.

In the context of the current application, the use of the pounce gesture was also slightly awkward. As noted previously, the pounce gesture is used to activate the crease operation of the drawing plane tool. This function allows the user to create a crease line along which the current drawing plane will be constrained during subsequent rotation operations. Unfortunately, the pounce gesture was assigned to unmodified input from the digitizing stylus, which is otherwise used to create strokes, while other adjustment operations for the current plane are assigned to strokes under the OPTION modifier key. This means that after the crease is set by a pounce operation, the user must activate the modifier key to make any use of those changes to the crease constraint. This arrangement is unintuitive and would likely be frustrating to new users. A more appropriate setup would be to provide the pounce gesture in the same context as the other plane adjustment inputs.

In general, although the pounce gesture suffered from several issues in its current implementation, the general concept of the pounce still feels viable. The physical performance of the operation is simple, natural, and easy to explain and learn. Furthermore, it avoids the need to maintain pressure levels during an operation, something other studies found to be difficult [Tsang *et al.*, 2004], and focuses pressure detection at the high end of the scale where user control is greatest [Ramos *et al.*, 2004]. With minor adjustments to how pounce is included in the input system, and the addition of greater user control over its parameters,

the pounce gesture's efficacy should improve.

The final gesture under consideration is the flick operation, used in the application to switch between the available stroke pens. Of the three gestures implemented as part of the application, this gesture is by far the most successful. Using the flick motion while working with the application is both natural and enjoyable, and because of the simple physical motion involved, the function is activated correctly with ease.

With very little practice the operation becomes automatic—so much so that at many points during testing there was a desire to have the same functionality available to switch between other sets of options in the system. Users familiar with key combinations like the common ALT+TAB to switch between currently running applications should have some idea of how useful the operation can be. However beyond cycling through options in a single direction, the flick allows bidirectional movement, which is a welcome addition. Future development of this feature will involve expansion of the operation into other areas of the program. By combining the function with modifier keys or limited context awareness the flick gesture should be applicable in many other situations.

Finally, discussing the use of the tablet in general, one additional observation arose during the testing process. Despite warnings by a number of authors regarding mistaken activations of the barrel buttons on the digitizing stylus [Miller, 2005] [Ramos *et al.*, 2004], difficulties of this type did not arise. Although the stylus was manipulated a great deal in order to perform the lift-and-lead gesture for example, no major incidences of errant clicks occurred.

In considering these results, two possible explanations occur. First, the author is very familiar with the user of the digitizing tablet, and has many hours of

experience using the device both with the current application, and in other applications both for drawing and writing. It may be the case that this experience has allowed the author to avoid producing mistaken clicks.

Second, the other authors describing this ergonomic shortcoming may have based their observations on older or alternate versions of the tablet hardware. Miller states that testing of their application was performed on Wacom based tablets and TabletPC hardware, but does not specify which models were used, while Ramos *et al.* describe their testing environment as based on the Wacom Intuos tablet model. Although the basic hardware components of the Wacom line of products are based on the same underlying hardware, there is a degree of variation between the ergonomics of styli in different lines. See for example the differences between the two styli pictured in Figure 6.8 on page 269. This is especially true in the TabletPC market where the stylus often has a compact design, allowing it to be stored internally in the TabletPC. It is likely that, especially with the main Intuos line of products, Wacom developers have focused on improving the ergonomics of the stylus with the express purpose of avoiding accidental activation of the barrel buttons. Because testing of the current application was performed on a Wacom Intuos II tablet device, the design of the stylus may have aided in avoiding this particular problem.

### **8.1.3 User Interface**

Although by design the user's experience of the system is primarily via the digitizing tablet and a small number of modifier keys, the application still offers a wider set of redundant interface methods. This includes the graphical user interface elements such as the tool and plane palettes, as well as support for

modeling with the mouse and keyboard, alone or in tandem with a digitizing stylus.

First, as noted in Section 6.4.2, original designs for the system called for an application devoid of standard graphical user interface components. However, despite these plans, it was decided that a more standard—though simplified—palette-style interface would be added. Subsequent work with the program has justified this design decision. Although the operations and states of the application are much simpler than the complex interfaces of larger applications, the interaction is still complex enough to warrant some indication of the current state of the program. This issue is partially addressed by the use of various tool cursors, but cursors alone do not provide the same degree of feedback.

Second, although the system is designed to work with tablet input, there is some indication that mouse input may deserve greater scrutiny. For very early testing, many of the common interface tasks assigned in the final system to tablet gestures such as view and plane adjustments were performed using a standard mouse because the tablet gesture system had not yet been developed. One of the goals of the gesture system was to provide a more intuitive means of controlling these aspects of the program, under the assumption that switching back and forth between the stylus for drawing and the mouse for other commands would be distracting to the sketching process. However, while construction of the gesture system was under way, working with both the stylus and tablet was a common occurrence. Observations from this period, as well as subsequent use of the application revealed that, although switching input methods can occasionally be a distraction, the situation is not always clear cut.

For some operations, the ability to quickly access alternate functionality with



the digitizing stylus is extremely helpful. This is most often true of small viewing alterations, which are performed in the course of a single set of strokes. The same is true for the use of drawing-like commands, such as the creation of crease lines when manipulating a drawing plane. However, in the case of larger viewing or drawing plane adjustment, it appears that these operations occur as boundary events between focused sketching sessions. The situation is somewhat similar to typing a sentence or paragraph into one section of a text document, and then moving some ways up or down in the text to make another edit. Although the user may be working on a single train of thought, there is a small natural break in activity. Because of this, it can feel just as natural to reach for the mouse to perform the necessary operations as it does using the stylus. This suggests that it may be beneficial to expand support for the mouse for non-primary operations, perhaps allowing basic or more granular control with the tablet, and more complex large scale control with the mouse.

Although interactions within the application are, for the most part, successful, there are several additional features that could improve the user experience. First, in regards to the camera and view system, the current implementation provides no means of spinning the camera about its current depth axis, an operation that is visually manifested as a ‘barrel roll’. On several occasions during the construction process this feature would have been helpful in orienting the view of the modeling environment to gain a comfortable angle for drawing.

Second, as the construction of a model progresses and the number of modeling components and drawing surfaces increases, the view of the environment can become quite cluttered. The application does provide a means of hiding individual drawing surfaces through the drawing plane palette, however a more direct

method that could be accessed from the tablet or keyboard would be helpful. Furthermore, this feature should be extended to include not only planes and their associated strokes, but modeling components as well. More appropriate would be the ability to associate groupings of drawing planes and modeling components together and adjust their visibility *en-masse*.

#### 8.1.4 Modeling Methods

Although the primary focus of input for the user may be strokes, the purpose of those strokes within the system is the creation of 3-D geometry components. Thus, the modeling system is the heart of the application, and its design in large part dictates the success or failure of the ability to create geometry effectively with a sketch-based method.

The design of the current 3-D construction system offers a number of advantages over both traditional sketching, and the other sketch based modeling construction methods we have discussed. One of the most difficult aspects of accurately rendering 3-dimensional shapes in traditional drawing is rendering objects so that they are naturally oriented, structurally consistent, and seem to fit correctly into their environment in relation to other objects. Even for simple volumes like boxes or cubes this can be a challenge and art students typically spend a great deal of time simply practicing compositions with these simple forms. The shapes that compose a box—basic squares and rectangles—are not particularly difficult to render, instead the challenge is in properly skewing and rotating these squares and rectangles until they form a convincing 3-D shape with parallel faces and even sides.

For a box, the one exception to this situation occurs when the artist looks at

one face head-on. From this perspective the sides of the box are perfectly aligned with the line of sight and so all the structural complexity of the box degenerates into the simple rectangular shape of that single front-facing side. This ‘head-on box’ is almost trivial to draw, but at the same time its depiction lacks the visual interest of a box set at an arbitrary angle, and so art students are encouraged to avoid such simple composition in favor of more interesting viewpoints. Designers and engineers are also taught to eschew the degenerate angles, not for any aesthetic reason, but because a depiction of a model in which most of the geometry is obscured are of little use.

Whereas traditional sketch artists must concern themselves with these issues, the 3-D environment of the modeling application makes it possible to avoid this aspect of the sketching process. Rather than being limited to a single point of view, the artist is free to move about the model and examine it from every angle. What’s more, the positions of the drawing surfaces on which the artist works are also completely under his or her control. This means that the artist is free to draw portions of the model from whatever angle is easiest. Thinking about the boxes again, there is no need for the artist to concern themselves with a fixed composition. Instead he or she can select the easiest angle from which to draw any component. This not only makes the drawing task easier, but faster and more enjoyable because the artist can spend more time developing his or her design, and less time struggling with proper perspective and shape. Only later, once the geometry of the model has been defined, does the artist need to devote serious consideration to an appealing composition, something that may not even be a consideration for a design or planning oriented sketch.

For amateur artists whose skills may not yet be developed, this functionality

opens up the drawing process and allows them to create more complex compositions more easily. At the same time, the application provides an ideal learning environment where an art student can experiment with different views of the same subject to help them understand how its different components and geometric shapes interact. For professional artists and designers, the application provides access to every side and angle of the model, giving a more complete picture of its 3-D shape. This helps the user to understand the design more fully, and fuels the creative process. Finally, although it possible to always work from these convenient angles, the artist can also position drawing surfaces at oblique angles to the viewing plane, simulating the familiar feeling of drawing from a traditional 2-dimensional view. This can help traditional artists utilize their existing skills.

Like many of the other sketch-based modeling applications discussed in Chapter 5, this program is designed to allow the user to build 3-dimensional models using sketch as the primary tool. In relation to other sketch based modeling applications the current program offers a number of similar features and improvements.

Compared with blobby information methods, the current application attempts to preserve the same feeling of free drawing by developing 3-D geometry directly from the user's strokes. In the current application this philosophy is taken one step further, allowing the user to not only enter strokes for geometric construction, but also to use strokes in a sketching fashion to develop ideas and provide a framework for model development. This functionality reflects the fact that not all sketching input is intended as a component of the visual representation of the artist's subject. In contrast to the blobby inflation systems, the present application gives the user much greater control over the shape and form of the generated 3-D models. Rather than forcing the user into default rounded cross sections, models are defined by

user supplied die strokes, which can take any shape the user can draw. This includes simple circles and rounded forms, but also sharp corners, flat ribbons, thin sheets, or arbitrary designs. This makes it easy to create a variety of forms very quickly.

Although the present system requires slightly more input from the user than some of the blobby inflation systems, the creation of geometry is still achieved in relatively few strokes. An advantage of this and similar extrusion based systems pointed out by Cherlin *et al.* [Cherlin *et al.*, 2005]. Although the user is free to create as many sketching and constructive strokes as he or she wishes, when the time comes to generate geometry, the three active strokes are all that is required to construct a volume. This is equivalent to, and in some cases less than, the most cursory set of traditional sketch lines used to suggest the same form. Again, this allows the user to work swiftly and experiment easily. Furthermore, although die strokes, path strokes, and size strokes provide the system with the maximum amount of information, as noted above, the user need only supply as much information as is necessary to generate the desired 3-D form. Sizing strokes can be omitted to quickly create uniform tubes and ribbon like shapes, whereas flat shapes and end caps require only a die stroke to define them.

Although the sweep and generalized cylinder construction methods in the present system are very similar to those utilized by Cherlin *et al.* the modeling interface provides several important improvements over that system. First, the present application preserves sharp corner elements within the user's strokes, and allows strokes that are not closed to form the basis of cross sectional shapes. This provides the user with a larger variety of possible shapes. Second, the current system provides for the construction of flat polygon shapes, and uniformly scaled

extrusions, which can be used to quickly generate many common 3-D forms. Furthermore, with the current system the user has a greater degree of control over the behavior of sweeps and generalized cylindrical constructions by adjusting the position of the path stroke relative to the die stroke. Finally, by allowing the user to adjust and arrange his or her own drawing surfaces within the modeling environment, the present application allows the user to create modeling components in place. This is in contrast to Cherlin *et al.*'s system in which components are created at arbitrary positions and then subsequently dragged into place, an operation that the authors identify as a major drawback to their system.

One issue that has arisen with the 3-D construction system not mentioned in Cherlin *et al.*'s publication is in dealing with inaccuracies in the user's path stroke or averaged extrusion stroke. Because the individual contours of the 3-D structure are aligned based on properties of these path strokes, if the stroke contains minor shakes or bumps, these features can be reflected and magnified in the resulting 3-D construction. Often this simply serves to give the resulting shape an inexact and sketchy feeling, but occasionally the results can be annoying or disruptive. Early testing with the present system before the implementation of a workaround for event coalescing in the operating system indicate that screening out additional sample points can reduce the appearance of these 'wiggles'. This suggests that the introduction of a more intelligent line smoothing system or smoothing tool may be beneficial.

### **8.1.5 Summary**

By focusing on the mental processes underlying sketching and developing the modeling interface targeted at these activities, the sketch-based modeling methods

developed for the application show promise as an efficient and intuitive means of creating simple, preliminary three-dimensional models. By providing the user with the familiar 2-dimensional interface of planar drawing surfaces, and allowing the user to arrange and manipulate the surfaces within the modeling environment, the modeling system maintains a strong base in the familiar mental processes utilized by traditional 2-dimensional artists, while providing a new instrument for the outlet of that creative process in a 3-dimensional medium. Finally, the set of tablet gestures demonstrate novel ways in which the underutilized dynamic physical information provided by tablet hardware can be used to control various aspects of the application. Although the current implementations of some of the tablet gestures will require additional development, their initial performance is encouraging, and suggest that future development is well worth the effort. In the case of other gestures, their utility is obvious and further research can now be directed at other application circumstances under which they could be applied.

## 8.2 Contributions

Within the field of sketch-based modeling, the present research effort provides a demonstration of how a successful modeling tool can be developed *not* as a direct translation of a specific physical artistic technique, but instead targeting the mental processes that underlie those physical techniques. This makes it possible to create an interface that can carry a 2-dimensional artist's traditional skills into the computational and three-dimensional domain without also bringing along the limitational baggage of the original artistic technique.

Through the construction, development, and testing of this design philosophy, this work also provides a reference of traditional and contemporary sketching

practice—both analog and digital—as well as a model of the mental process of sketching. Furthermore, in relation to the construction of a sketch-based modeling application, this work provides seven design principles that form a foundation for constructing an application conducive to this cognitive-sketching design philosophy.

Finally, this work provides an examination of the performance characteristics and possible utility of dynamic physical pressure, hover, and tilt information provided by a digitizing tablet hardware device. From this information, design descriptions for five tablet gestures were developed, which could provide an additional or alternative channel of application control. Among these gestures, three specific designs were implemented, and evaluated as components of a unified modeling interface. Preliminary results of their efficacy as means of application control was provided, as well as thorough discussion of their future development or possible application to other application environments.

### **8.3 Limitations**

Although the present application was designed as a modeling tool, its feature set and application interface are commensurate with a prototype application and testing framework. Because of this, many features that would be a necessity in any commercial tool are only modestly represented or have been omitted entirely. Users have limited control over the placement of strokes or modeling components within the environment once they are created. In addition, although modeling components can be placed in proximity to one another to create the illusion of larger connected components, there is currently no means of computationally grouping or associating individual modeling components, or topologically



combining their forms. Furthermore, at present, models cannot be exported into other modeling applications, nor can existing models be imported into the present system.

In terms of the three-dimensional construction methods, although the user can create a wide variety of shapes and forms as discussed in Section 6.4.11.3—in some cases a wider variety of forms that are available in other sketch-based modeling applications with similar construction methods—there are a number of forms that are difficult or impossible to create with the present system. Models within the system are generally limited to common geometric shapes and other two-manifold forms. The system cannot, in general, be used to create highly complex or non-manifold surfaces. In addition, because of the sweep and generalized cylinder-based construction methods, certain common geometric shapes can be difficult to create with a single construction operation. For example, it's easier to create a spherical object from the combination of two hemispheric shapes, rather than as a single construction. Furthermore, although the size and orientation of the cross-sectional shape of a modeling component can be manipulated by the user's input strokes, the system can only accept a single die shape for each construction. This means it is not possible to create forms in which the die shape changes or blends between multiple shapes over the course of the extrusion.

Finally, because all constructions are based on the strokes provided by the user, and all strokes are produced through the projection of the users drawing input projected onto 2-dimensional drawing surfaces, although the constructed components have 3-dimensional form, they are limited to 2-dimensional die shapes and extrusion paths. This means for example, that a tube or noodle shaped object created by the user may follow an arbitrarily twisting, turning, and curving path,

but that path must be confined to the surface of a single plane. It is currently not possible to create a form that follows the path of an arbitrary space curve.

## 8.4 Future Work

An obvious next step for the current application is to begin user testing. In terms of the current prototype a series of supervised individual tests with volunteer users would be a good starting point. These tests could focus on individual features such as the 3-D construction mechanisms, application interface, and tablet gestures, with the goal of eliciting user opinions of the efficacy, strengths, weaknesses, and possible improvements to these features. Observations of test subjects as they use the application would also provide valuable information about the usability of the modeling interface. As the application is developed further from a prototype and testing platform into a more well-rounded system, it could then be distributed to a number of test subjects for more rigorous testing and feedback from real-world use.

Based on preliminary assessment of the current application, there are several areas that warrant further development. A particular area of interest, as noted in Section 8.1.1, is in providing a better incorporation of contemplative and extraneous sketching strokes. Further research in this area could follow several paths. First, as noted previously, there is some interest in investigating the use of raster graphic, or pseudo-raster graphic stroke representations, which may be better able to represent the behavioral characteristics of these sketching lines.

In a related vein, it may be beneficial to consider the incorporation of a stroke correction system—raster derived or otherwise—into the current stroke tools. As discussed in Section 6.4.7, general stroke correction methods appear to be a hin-

drance to the efficiency of the user interface. However, utilizing more limited or focused correction methods, such as the casting strokes described in Section 6.4.7, it may be possible to develop a correction system that provides the user with a better sketching experience.

Given the initial success of the 2-D to 3-D conversion system developed as part of the application, another avenue of potential future development focuses on the expansion of the current interface to allow an even wider variety of 3-dimensional shapes to be created. Cherlin *et al.* for example demonstrate how multiple die shapes can be provided to create modeling components with transitioning or blending cross-sectional shapes [Cherlin *et al.*, 2005]. Furthermore, Sederberg and Greenwood provide descriptions of dynamic programming algorithms that can be used to produce coherent and visually aesthetic blends between outline shapes defined as parametric curves [Sederberg & Greenwood, 1995]. Similarly, through the application of stroke-based deformation methods such as those demonstrated by Kho and Garland [Kho & Garland, 2005], or more appropriately three-dimensional curve construction methods such as those discussed by Grossman *et al.* [Grossman *et al.*, 2001] [Grossman *et al.*, 2002] [Grossman, 2004], Cohen *et al.* [Cohen *et al.*, 1999], and Tsang *et al.* [Tsang *et al.*, 2004], could provide the application with the means of constructing 3-dimensional components based on arbitrary space curves.

Finally, the apparent success of other sketch-based modeling applications based on implicit or volumetric model representations makes exploring these alternative modeling forms an attractive prospect. In particular, the convolution surface implicit modeling representation described by [Bloomenthal & Wyvill, 1990] [Wyvill & Guy, 1998] [Schmidt *et al.*, 2005; Tai *et al.*, 2004] appears to be distinctly

well-suited to the current construction method. This representation describes a three-dimensional model through the convolution of an implicit kernel along a skeletal path. Conceptually this operation is very similar to sweeping a die shape along an extrusion path.

# References

- Accot, Johnny, & Zhai, Shumin. 2002. More than dotting the i's - Foundations for crossing-based interfaces. *Pages 73–80 of: ACM Conference on Human Factors in Computing Systems.*
- Adobe. 2007a. *Adobe Illustrator CS3.*
- Adobe. 2007b. *Adobe Photoshop CS3.*
- Agarawala, Anand, & Balakrishnan, Ravin. 2006. Keepin' it Real: Pushing the Desktop Metaphor with Physics, Piles and the Pen. *Pages 1283–1292 of: Proceedings of CHI 2006 - the ACM Conference on Human Factors in Computing Systems.* Montréal, Québec, Canada: ACM Press.
- Alex, John. 2005. *Hybrid Sketching: A New Middle Ground Between 2- and 3-D.* Ph.D. thesis, Massachusetts Institute of Technology.
- Angel, Edward. 2003. *Interactive Computer Graphics: A Top-Down Approach with OpenGL.* 3 edn. Addison Wesley Pearson Education.
- Apple. 2001. *Mac OS X 10.1 November 2001 Developer Tools CD Release Notes: Objective-C++.* Developer Tools CD Release Notes <http://developer.apple.com/releasenotes/Cocoa/Objective-C++.html>. Apple Computer, Inc.

- Araújo, Bruno Rodrigues de, & Jorge, Joaquim Armando Pires. 2003. BlobMaker: Free form Modelling with Variational Implicit Surfaces. *Pages 17–26 of: 12th Encontro Português de Computação Grafica - 12th EPCG.*
- Autodesk. 2007a. *AutoCAD 2008.*
- Autodesk. 2007b. *Maya 8.5.*
- Baudel, Thomas. 1994. A mark-based interaction paradigm for free-hand drawing. *Pages 185–192 of: Szekely, Pedro (ed), Proceedings of the 7th annual ACM symposium on User interface software and technology.*
- Blender Foundation, The. 2007. *Blender 2.44.*
- Blinn, James F. 1978. Simulation of Wrinkled Surfaces. *Computer Graphics*, **12**(3), 286 – 292.
- Bloomenthal, Jules. 1990. Calculation of Reference Frames along a Space Curve. *Page 833 of: Glassner, Andrew S. (ed), Graphics Gems*, 1 edn. Graphics Gems. San Diego, California, U.S.A.: Academic Press, Inc.
- Bloomenthal, Jules, & Wyvill, Brian. 1990. Interactive Techniques for Implicit Modeling. *Pages 109 – 116 of: Proceedings of the 1990 symposium on Interactive 3D graphics.* Snowbird, Utah, United States: ACM Press.
- Bloomenthal, Mark, Zeleznik, Robert, Fish, Russell, Holden, Loring, Forsberg, Andrew, Riesenfeld, Richard, Cutts, Matt, Drake, Samuel, Fuchs, Henry, & Cohen, Elaine. 1998. Sketch-N-Make: Automated Machining Of CAD Sketches. *In: Proceedings of the 1998 ASME 8th Computers In Engineering Conference.*

- Bourguignon, David, Cani, Marie-Paule, & Drettakis, George. 2001. Drawing for Illustration and Annotation in 3D. *Computer Graphics Forum*, **20**, 114–122.
- Breen, Christopher. 2006 (August 25, 2006). *Apple's Greatest Hits: 30 Significant Products - No. 20: MacWrite and MacPaint*. <http://www.macworld.com/products/anniversary/MacWrite-and-MacPaint.php>.
- Briggs, Robert O., Dennis, Alan R., Beck, Brenda S., & Nunamaker, Jay. 1993. Whither the Pen-Based Interface? *Journal of Management Information Systems*, **9**(3), 71 – 90.
- Brönnimann, Hervé. 2001. Designing and Implementing a General Purpose Halfedge Data Structure. *Page 199 of: Brodal, Gerd Stoelting, Frigioni, Daniele, & Marchetti-Spaccamela, Alberto (eds), Proceedings of the 5th International Workshop on Algorithm Engineering*. Lecture Notes In Computer Science, vol. 2141. Aarhus, Denmark: Springer-Verlag.
- Buchanan, John W., & Sousay, Mario C. 2000. The edge buffer: A data structure for easy silhouette rendering. *Pages 39–42 of: Proc. of NPAR '00*.
- Bærentzen, J. Andreas, & Christensen, Niels Jørgen. 2002. Volume Sculpting Using the Level-Set Method. *Page 175 of: Proceedings of the Shape Modeling International 2002 (SMI'02)*. IEEE Computer Society.
- Carr, J. C., Beatson, R. K., Cherrie, J. B., Mitchell, T. J., Fright, W. R., McCallum, B. C., & Evans, T. R. 2001. Reconstruction and Representation of 3D Objects with Radial Basis Functions. *Pages 67 – 76 of: International Confer-*

- ence on Computer Graphics and Interactive Techniques archive Proceedings of the 28th annual conference on Computer graphics and interactive techniques.*
- Cherlin, Joseph Jacob, Samavati, Faramarz, Sousa, Mario Costa, & Jorge, Joaquim A. 2005. Sketch-based Modeling with Few Strokes. *Pages 137–145 of: Proceedings of the 21st Spring Conference on Computer Graphics.* Budmerice, Slovakia: ACM Press.
- Christiansen, H. N., & Sederberg, T. W. 1978. Conversion of complex contour line definitions into polygonal element mosaics. *Pages 187 – 192 of: International Conference on Computer Graphics and Interactive Techniques Proceedings of the 5th annual conference on Computer graphics and interactive techniques.*
- Clowes, Maxwell B. 1971. On Seeing Things. *Artificial Intelligence*, **2**, 79 – 116.
- Cohen, Jonathan, Hughes, John F., & Zeleznik, Robert. 2000. Harold: A World Made of Drawings. *In: Proceedings of the First International Symposium on Non Photorealistic Animation and Rendering (NPAR) for Art and Entertainment.*
- Cohen, Jonathan M., Markosian, Lee, Zeleznik, Robert C., & Hughes, John F. 1999. An Interface for Sketching 3D Curves. *Pages 17–21 of: Proceedings of the 1999 ACM Symposium on Interactive 3D Graphics.* Atlanta, Georgia: ACM Press.
- Conner, D. Brookshire, Snibbe, Scott S., Hemdon, Kenneth P., Robbins, Daniel C., Zeleznik, Robert C., & Dam, Andries van. 1992. Three-Dimensional Widgets. *Pages 183–188 of: Proceedings of the 1992 symposium on Interactive 3D graphics.* Cambridge, Massachusetts: ACM Press.



Contributors, Internet Movie Database. 2006a (August 15, 2006). *Beauty and the Beast (1991)*. <http://www.imdb.com/title/tt0101414/>.

Contributors, Internet Movie Database. 2006b (August 15, 2006). *Tarzan (1999)*. <http://www.imdb.com/title/tt0120855/>.

Contributors, Internet Movie Database. 2006c (September 30, 2006). *Who Framed Roger Rabbit (1988)*. <http://www.imdb.com/title/tt0096438/>.

Contributors, Wikipedia. 2006d (5 August 2006 22:52 UTC). *Computer Animation Production System*. [http://en.wikipedia.org/w/index.php?title=Computer\\_Animation\\_Production\\_System&oldid=69451218](http://en.wikipedia.org/w/index.php?title=Computer_Animation_Production_System&oldid=69451218).

Contributors, Wikipedia. 2006e (28 September 2006 17:01 UTC). *Corel Painter*. [http://en.wikipedia.org/w/index.php?title=Corel\\_Painter&oldid=74111634](http://en.wikipedia.org/w/index.php?title=Corel_Painter&oldid=74111634).

Contributors, Wikipedia. 2006f (30 September 2006 23:10 UTC). *Lego Mindstorms*. [http://en.wikipedia.org/w/index.php?title=Lego\\_Mindstorms&oldid=78692747](http://en.wikipedia.org/w/index.php?title=Lego_Mindstorms&oldid=78692747).

Contributors, Wikipedia. 2006g (15 August 2006). *Walt Disney Feature Animation*. [http://en.wikipedia.org/w/index.php?title=Walt\\_Disney\\_Feature\\_Animation&oldid=69679121](http://en.wikipedia.org/w/index.php?title=Walt_Disney_Feature_Animation&oldid=69679121).

Contributors, Wikipedia. 2007a (31 May 2007 17:16 UTC). *Frenet-Serret formulas*. [http://en.wikipedia.org/w/index.php?title=Frenet-Serret\\_formulas&oldid=134671006](http://en.wikipedia.org/w/index.php?title=Frenet-Serret_formulas&oldid=134671006).

Contributors, Wikipedia. 2007b (24 August 2007 17:32 UTC). *Hammerspace*. <http://en.wikipedia.org/w/index.php?title=Hammerspace&oldid=151614028>.

- Corel. 2006. *Corel Painter X.5*.
- Cuno, Alvaro, Esperança, Claudio, Cavalcanti, Paulo Roma, & Farias, Ricardo. 2005. 3D Free-form Modeling with Variational Surfaces. *In: The 13th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2005*.
- Davis, James, Agrawala, Maneesh, Chuang, Erika, Popović, Zoran, & Salesin, David. 2003. A Sketching Interface for Articulated Animation. *Symposium on Computer Animation*, 320–328.
- Davison, Joanna. 2003. *Modelling the Human Ear*. Ph.D. thesis, University of Sheffield.
- DeRose, Tony, Kobbelt, Leif, Levin, Adi, Sweldens, Wim, Zorin, Denis, & Schröder, Peter. 2000. SIGGRAPH Course: Subdivision for Modeling and Animation. *In: SIGGRAPH 2000*.
- Derry, John. 1996 (Thursday, September 28, 2006). *Of Pianos and Oranges*. <http://zonezero.com/magazine/articles/derry/derry.html>.
- Diehl, Holger, Müller, Franz, & Lindemann, Udo. 2004. From raw 3D-Sketches to exact CAD product models - Concept for an assistant-system. *Pages 137–141 of: Hughes, John F., & Jorge, Joaquim A. (eds), EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling (2004)*.
- Do, Ellen Yi-Luen. 2005. Design Sketches and Sketch Design Tools. *KBS - Knowledge Based Systems*, **18**, 383–405.
- Do, Ellen Yi-Luen, & Gross, Mark D. 1996. Drawing as a Means to Design Reasoning. *In: Visual Representation, Reasoning and Interaction in Design Work-*

- shop notes, Artificial Intelligence in Design '96 (AID '96)*. Stanford University: Stanford University.
- Dunn, Fletcher, & Parberry, Ian. 2002. *3D Math Primer for Graphics and Game Development*. 1 edn. Wordware Game Math Library. Plano, Texas, U.S.A: Wordware Publishing, Inc.
- Durand, Frédo. 2002. An Invitation to Discuss Computer Depiction. *Pages 111–124 of: Proceedings of The 2nd International Symposium on Non-Photorealistic Animation and Rendering*. Annecy, France: ACM-Press.
- Eggl, Lynn, Brüderlin, Beat D., & Elber, Gershon. 1995. Sketching as a solid modeling tool. *In: ACM Symposium on Solid and Physical Modeling: Proceedings of the third ACM symposium on Solid modeling and applications*. Salt Lake City, Utah: ACM-Press.
- Farin, Gerald. 2002. *Curves and Surfaces for CAGD - A Practical Guide*. 5th edn. San Diego, CA: Academic Press.
- Fekete, Jean-Daniel, Bizouarn, Érick, Cournarie, Éric, & Taillefer, Thierry Galas Frédéric. 1995. TicTacToon: a paperless system for professional 2D animation. *Pages 79–90 of: International Conference on Computer Graphics and Interactive Techniques archive Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*.
- Ferley, Eric, Cani, Marie-Paule, & Gascuel, Jean-Dominique. 1999. Practical Volumetric Sculpting. *Implicit Surfaces*.
- Feynman, Richard P. 1963. The Relation of Physics to Other Sciences. *Pages 47 – 67 of: Six Easy Pieces*, 3rd edn. Cambridge, Massachusetts: Helix Books.

- Fiore, Fabian Di, & Reeth, Frank Van. 2002. A Multi-Level Sketching Tool for "Pencil-and-Paper" Animation. *In: American Association for artificial Intelligence (AAAI 2002) Spring Symposium*. Stanford University in Palo Alto, California: AAAI.
- Fitzmaurice, George, Khan, Azam, Pieké, Robert, Buxton, Bill, & Kurtenbach, Gordon. 2003. Tracking Menus. *ACM UIST 2003 Symposium on User Interface Software and Technology*, 71–79.
- Fleisch, Timo, Rechel, Florian, Santos, Pedro, & Stork, André. 2004. Constraint Stroke-Based Oversketching for 3D Curves. *Pages 161–165 of: Hughes, John F., & Jorge, Joaquim A. (eds), EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling*.
- Fonseca, Manuel J., Pimentel, César, & Jorge, Joaquim A. 2002. CALI: An Online Scribble Recognizer for Calligraphic Interfaces. *Pages 51–58 of: AAAI Spring Symposium on Sketch Understanding, AAAI Technical Report SS-02-08*.
- Forsberg, Andrew S., Joseph J. LaViola, Jr., Markosian, Lee, & Zeleznik, Robert C. 1997. Seamless Interaction in Virtual Reality. *IEEE Computer Graphics and Applications*, **17**(6), 6 – 9.
- Friskén, Sarah F., Perry, Ronald N., Rockwood, Alyn P., & Jones, Thouis R. 2000. Adaptively Sampled Distance Fields: A General Representation of Shape for Computer Graphics. *Pages 249–254 of: Akeley, Kurt (ed), Siggraph 2000, Computer Graphics Proceedings*. ACM Press / ACM SIGGRAPH / Addison Wesley Longman.

- Galyean, Tinsley A., & Hughes, John F. 1991. Sculpting: An Interactive Volumetric Modeling Technique. *Computer Graphics*, **25**(4), 267–274.
- Garner, Steve. 2001. Is sketching still relevant in virtual design studios? *In: Proc., Design Computing on the Net (DCNet 2000), a refereed virtual conference.*
- Girshick, Ahna, Interrante, Victoria, Haker, Steven, & Lemoine, Todd. 2000. Line Direction Matters: An Argument For The Use Of Principal Directions In 3D Line Drawings. *Pages 43–52. of: First International Symposium on Non Photorealistic Animation and Rendering (NPAR 2000).*
- Gooch, Amy, Gooch, Bruce, Shirley, Peter, & Cohen, Elaine. 1998. A Non-Photorealistic Lighting Model For Automatic Technical Illustration. *SIG-GRAPH 1998.*
- Gravesen, Jens. 1992. *Adaptive Subdivision and the Length of Bézier Curves.* Technical Report mat-report no. 1992-10. Mathematical Institute, The Technical University of Denmark.
- Grimstead, Ian. J., & Martin, Ralph. R. 1995. Creating solid models from single 2D sketches. *In: SMA '95: Proceedings of the Third Symposium on Solid Modeling and Applications.* ACM-Press.
- Gross, Mark D, & Do, Ellen Yi-Luen. 1996. Ambiguous Intentions: a Paper-like Interface for Creative Design. *Pages 183–192 of: UIST '96: Proceedings of the 9th annual ACM symposium on User interface software and technology.* Seattle, Washington: ACM Press.
- Grossman, Tovi. 2004. *Alternate User Interfaces for the Manipulation of Curves in 3D Environments.* Ph.D. thesis, University of Toronto.

- Grossman, Tovi, Balakrishnan, Ravin, Kurtenbach, Gordon, Fitzmaurice, George, Khan, Azam, & Buxton, Bill. 2001. Interaction techniques for 3D modeling on large displays. *In: SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*. ACM Press.
- Grossman, Tovi, Balakrishnan, Ravin, Kurtenbach, Gordon, Fitzmaurice, George W., Khan, Azam, & Buxton, William. 2002. Creating principal 3D curves with digital tape drawing. *Pages 121–128 of: CHI 2002 Conference Proceedings ACM Conference on Human Factors in Computing Systems*.
- Guzmán, Adolfo. 1968. Decomposition of a Visual Scene into Three-Dimensional Bodies. *Pages 291 – 304 of: AFIPS Proceedings of the 1968 Fall joint Computer Conference*. Thompson Book Co.
- Götze, Marcel, Neumann, Petra, & Isenberg, Tobias. 2005. User-Supported Interactive Illustration of Text. *Simulation und Visualisierung*, 195–206.
- Henzen, Alex, Ailenei, Neculai, Fiore, Fabian Di, Reeth, Frank Van, & Patterson, John. 2005. Sketching with a Low-latency Electronic Ink Drawing Tablet. *Pages 51 – 60 of: Computer graphics and interactive techniques in Australasia and South East Asia archive Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*.
- Herbert, Daniel M. 1993. *Architectural Study Drawings*. Wiley.
- Hertzfeld, Andy. 2004a (August 25, 2006). *MacPaint Evolution*. [http://www.folklore.org/StoryView.py?project=Macintosh&story=MacPaint\\_Evolution.txt&showcomments=1#comments](http://www.folklore.org/StoryView.py?project=Macintosh&story=MacPaint_Evolution.txt&showcomments=1#comments).

- Hertzfeld, Andy. 2004b (August 25, 2006). *MacPaint Gallery*.  
[http://www.folklore.org/StoryView.py?project=Macintosh&story=MacPaint\\_Gallery.txt](http://www.folklore.org/StoryView.py?project=Macintosh&story=MacPaint_Gallery.txt).
- Hickman, Craig. *Kid Pix: The Early Years*.  
<http://pixelpoppin.com/kidpix/KPHistory/>.
- Hinckley, Ken, Baudisch, Patrick, Ramos, Gonzalo, & Guimbretiere, Francois. 2005. Design and Analysis of Delimiters for Selection-Action Pen Gesture Phrases in Scriboli. *Pages 451-460 of: CHI 2005: Proceedings of the SIGCHI conference on Human factors in computing systems*.
- Hogarth, Burne. 1996. *Dynamic Figure Drawing*. Watson-Guption Publications.
- Hoiem, Derek, Efros, Alexei A., & Hebert, Martial. 2005. Automatic Photo Pop-up. *ACM SIGGRAPH 2005*.
- Hook, Brian R. 2003 (August 28, 2006). *Replacing Desktops Everywhere: The Laptop Trajectory*. <http://www.linuxinsider.com/story/32415.html>.
- Hsu, Siu Chi, & Lee, Irene H. H. 1994. Drawing and animation using skeletal strokes. *In: SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. ACM-Press.
- Huffman, David A. 1971. Impossible Objects as Nonsense Sentences. *Machine Intelligence*, **8**, 475 – 492.
- Huot, Stéphane, Dumas, Cédric, & Hégron, Gérard. 2003. Toward Creative 3D Modeling: an Architects' Sketches Study. *In: INTERACT 2003 - Bringing the Bits Together Ninth IFIP TC13 International Conference on Human-Computer Interaction*.

- Igarashi, Takeo. 2006. *HomePage of Takeo Igarashi*. <http://www-ui.is.s.u-tokyo.ac.jp/~takeo/index.html>.
- Igarashi, Takeo, & Hughes, John F. 2001. A Suggestive Interface for 3D Drawing. *In: 14th Annual Symposium on User Interface Software and Technology, ACM UIST'01*. Orlando, Florida: ACM Press.
- Igarashi, Takeo, & Hughes, John F. 2003. Smooth Meshes for Sketch-based Freeform Modeling. *Pages 139–142 of: Proceedings of the ACM Symposium on Interactive 3D Graphics*.
- Igarashi, Takeo, Kawachiya, Sachiko, Matsuoka, Satoshi, & Tanaka, Hidehiko. 1997a. In Search for an Ideal Computer-Assisted Drawing System. *Pages 104–111 of: The Sixth IFIP Conference on Human-Computer Interaction*.
- Igarashi, Takeo, Matsuoka, Satoshi, Kawachiya, Sachiko, & Tanaka, Hidehiko. 1997b. Interactive Beautification: A Technique for Rapid Geometric Design. *Pages 105–114 of: ACM Annual Symposium on User Interface Software and Technology*.
- Igarashi, Takeo, Matsuoka, Satoshi, Kawachiya, Sachiko, & Tanaka, Hidehiko. 1998. Pegasus: A Drawing System for Rapid Geometric Design. *Pages 24–25 of: ACM Conference on Human Factors in Computing Systems*.
- Igarashi, Takeo, Matsuoka, Satoshi, & Tanaka, Hidehiko. 1999. Teddy: A Sketching Interface for 3D Freeform Design. *In: ACM SIGGRAPH'99*. Los Angeles, California: ACM.
- Igarashi, Takeo, Moscovich, Tomer, & Hughes, John F. 2005. Spatial Keyfram-



- ing for Performance-driven Animation. *In: ACM SIGGRAPH / Eurographics Symposium on Computer Animation.*
- Ives, Colta, & Stein, Susan Alyson. 2005 (December, 2005). *Vincent van Gogh: The Drawings.* [http://www.metmuseum.org/special/Van\\_Gogh/](http://www.metmuseum.org/special/Van_Gogh/).
- Johnson, Crockett. 1981. *Harold and the Purple Crayon.* 50th anniversary edition edn. HarperTrophy.
- Johnston, Ollie, & Thomas, Frank. 1995. *The Illusion of Life: Disney Animation.* Disney Editions.
- Johnston, Scott F. 2002. Lumo: Illumination for Cel Animation. *Pages 45 – ff of: Non-Photorealistic Animation and Rendering Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering.*
- Jung, Thomas, Gross, Mark D., & Do, Ellen Yi-Luen. 2002. Space Pen: Annotation and sketching on 3D models on the Internet. *Pages 95–102 of: International Conference on Intelligent User Interfaces Proceedings of the 7th international conference on Intelligent user interfaces.*
- Kalnins, Robert D., Markosian, Lee, Meier, Barbara J., Kowalski, Michael A., Lee, Joseph C., Davidson, Philip L., Webb, Matthew, Hughes, John F., & Finkelstein, Adam. 2002. WYSIWYG NPT: Drawing Strokes Directly on 3D Models. *In: SIGGRAPH 2002: ACM Transactions on Graphics.* San Antonio, Texas: ACM.
- Kalnins, Robert D., Davidson, Philip L., Markosian, Lee, & Finkelstein, Adam. 2003. Coherent Stylized Silhouettes. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, **22**(3), 856–861.

- Karpenko, Olga, & Hughes, John F. 2006. SmoothSketch: 3D free-form shapes from complex sketches. *In: SIGGRAPH'06*. Boston, Massachusetts: ACM-Press.
- Karpenko, Olga, Hughes, John F., & Raskar, Ramesh. 2002. Free-form sketching with variational implicit surfaces. *Eurographics 2002*.
- Kasik, David J., Buxton, William, & Ferguson, David R. 2005. Ten CAD Challenges. *IEEE Computer Graphics and Applications*, **25**(2), 81 – 92.
- Kay, Alan. 1987. *Alan Kay: Doing with Images Makes Symbols: Communicating with Computers Pt 1*.
- Kerautret, Bertrand, Granier, Xavier, & Braquelaire, Achille. 2005. Intuitive Shape Modeling by Shading Design. *Pages 163–174 of: Butz, Andreas, Fisher, Brian, & Krüger, Antonio (eds), International Symposium on Smart Graphics Lecture Notes in Computer Science*, vol. 3638. Springer-Verlag GmbH.
- Kho, Youngihn, & Garland, Michael. 2005. Sketching Mesh Deformations. *Pages 147 – 154 of: Symposium on Interactive 3D Graphics*. Proceedings of the 2005 symposium on Interactive 3D graphics and games. Washington, District of Columbia: ACM.
- Kimball, Spencer, & Mattis, Peter. 1995. *The GNU Image Manipulation Program (GIMP)*.
- Kurtenbach, Gordon, & Buxton, William. 1993. The limits of expert performance using hierarchic marking menus. *Pages 482–487 of: Proceedings of InterCHI '93*.

- Kurtenbach, Gordon, Moran, T. P., & Buxton, William. 1994. Contextual Animation of Gestural Commands. *Computer Graphics Forum*, **13**(5), 305–314.
- Lake, Adam, Marshall, Carl, Harris, Mark, & Blackstein, Marc. 2000. Stylized Rendering Techniques For Scalable Real-Time 3D Animation. *Pages 13–20 of: NPAR '00: Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*. Annecy, France: ACM Press.
- Lawrence, Jason, & Funkhouser, Thomas. 2003. A Painting Interface for Interactive Surface Deformations. *In: Pacific Graphics*.
- Lehar, Steven. 2004. Gestalt Isomorphism and the Primacy of Subjective Conscious Experience: A Gestalt Bubble Model. *Behavioral and Brain Sciences*, **26**(4), 375 – 444.
- Levet, Florian, Granier, Xavier, & Schlick, Christophe. 2006. 3D Sketching with profile curves. *In: International Symposium on Smart Graphics*.
- Lipson, Hod, & Shpitalni, Moshe. 2002. Correlation-Based Reconstruction of a 3D Object from a Single Freehand Sketch. *In: Davis, Randall, Landay, James, & Stahovich, Tom (eds), American Association for Artificial Intelligence Spring Symposium: Sketch Understanding*. Stanford University in Palo Alto, California: AAAI.
- Long, Jr., Allan Christian, Landay, James A., & Rowe, Lawrence A. 1999. Implications For a Gesture Design Tool. *Pages 40–47 of: Human Factors in Computing Systems (SIGCHI Proceedings)*. ACM-Press.
- Lord, Peter, & Sibley, Biran. 2004. *Creating 3-D Animation: The Aardman Book*

- of Filmmaking*. Revised edition edn. New York, New York, U.S.A.: Harry N. Abrams, Inc.
- Lorensen, W. E., & Cline, H. E. 1987. Marching Cubes: a high resolution 3D surface reconstruction algorithm. *Computer Graphics*, **21**(4), 163–169.
- MacEachren, Alan M. 1995. *How Maps Work: Representation, Visualization, and Design*. New York, New York, U.S.A.: Guilford Press.
- Macromedia. 2005. *Macromedia FreeHand MX*.
- Markosian, J.D. Northrup, & Lee. 2000. Artistic Silhouettes: A Hybrid Approach. *In: Proceedings of the First International Symposium on Non Photorealistic Animation and Rendering (NPAR) for Art and Entertainment*.
- Markosian, Lee, Kowalski, Michael A., Trychin, Samuel J., Bourdev, Lubomir D., Goldstein, Daniel, & Hughes, John F. 1997. Real-Time Nonphotorealistic Rendering. *Pages 415 – 420 of: International Conference on Computer Graphics and Interactive Techniques Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co.
- Markosian, Lee, Cohen, Jonathan M., Crulli, Thomas, & Hughes, John. 1999. Skin: a constructive approach to modeling free-form shapes. *Pages 393 – 400 of: International Conference on Computer Graphics and Interactive Techniques Proceedings of the 26th annual conference on Computer graphics and interactive techniques*.
- Masry, Mark, Kang, Dong Joong, & Lipson, Hod. 2005. A freehand sketching

- interface for progressive construction of 3D objects. *Computers and Graphics*, **29**, 563–575.
- Meier, Barbara J. 1996. Painterly Rendering for Animation. *In: Siggraph 1996*.
- Michalik, Paul, Kim, Dae Hyun, & Bruderlin, Beat D. 2002. Sketch- and Constraint-based Design of B-spline Surfaces. *Pages 297 – 304 of: ACM Symposium on Solid and Physical Modeling Proceedings of the seventh ACM symposium on Solid modeling and applications*. Saarbrücken, Germany: ACM Press.
- Miller, Lynn. 2005. Case Study of Customer Input For a Successful Product. *Pages 225–234 of: Proceedings of the Agile Development Conference*.
- Newell, Dawn. 2003 (January 06, 2003). *Scientific and Technical Achievements Honored with Academy Awards*. Press Release. Academy of Motion Picture Arts and Sciences.
- Odson, Catherine. 2006 (Wednesday, April 26, 2006). *Students build concrete canoe*.
- Ohwada, Shigeru, Nielsen, Frank, Nakazawa, Kazuo, & Igarashi, Takeo. 2003. A Sketching Interface for Modeling the Internal Structures of 3D Shapes. *Pages 49–57 of: International Symposium on Smart Graphics, Lecture Notes in Computer Science (LNCS)*, vol. 2733. Springer-Verlag.
- Parks, Bob. 2005. Blockheads - Lego: The ultimate prototyping material. Seriously. *Make*, **2**, 36 – 37.
- Patterson, John, & Willis, Philip. 1995. Computer Assisted Animation: 2D or not 2D? *Computer Journal*, **37**(10), 829–839.

- Pereira, João P., Jorge, Joaquim A., Branco, Vasco, & Ferreira, F. Nunes. 2000. Towards Calligraphic Interfaces: Sketching 3d Scenes With Gestures And Context Icons. *WSCG2000*.
- Pereira, João P., Jorge, Joaquim A., Branco, Vasco, & Ferreira, F. Nunes. 2001. Reduced Instruction Set Calligraphic Interfaces: Sketching Complex 3d Objects With (Fewer) Gestures. *Pages 194–196 of: 4th European Academy of Design Conference Proceedings*.
- Pereira, João P., Jorge, Joaquim A., Branco, Vasco A., & Ferreira, F. Nunes. 2003. Calligraphic Interfaces: Mixed Metaphors for Design. *Pages 154 – 170 of: 10th International Workshop, DSV-IS*, vol. 2844 / 2003.
- Pereira, João P., Branco, Vasco A., Jorge, Joaquim A., Silva, Nelson F., Cardoso, Tiago D., & Ferreira, F. Nunes. 2004. Cascading Recognizers for Ambiguous Calligraphic Interaction. *In: Hughes, John F., & Jorge, Joaquim A. (eds), EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling*.
- Perry, Ronald N., & Frisken, Sarah F. 2001. Kizamu: A System For Sculpting Digital Characters. *Pages 47 – 56 of: International Conference on Computer Graphics and Interactive Techniques Proceedings of the 28th annual conference on Computer graphics and interactive techniques*.
- Petrovic, Lena, Fujito, Brian, Williams, Lance, & Finkelstein, Adam. 2000. Shadows for Cel Animation. *Pages 511–516 of: Akeley, Kurt (ed), Siggraph 2000, Computer Graphics Proceedings*. ACM Press / ACM SIGGRAPH / Addison Wesley Longman.

- Phong, Bui Tuong. 1973. Illumination for Computer Generated Images. *Communications of the ACM*, **18**(6), 311 – 317.
- Piegl, Les A., & Tiller, Wayne. 1997. *The NURBS Book - Monographs in Visual Communication*. 2 edn. Berlin, Germany: Springer.
- Pixologic. 2007. *ZBrush*.
- Rademacher, Paul. 1999. View-Dependent Geometry. *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, 439–446.
- Ramos, Gonzalo, Boulos, Matthew, & Balakrishnan, Ravin. 2004. Pressure Widgets. *Pages 487–494 of: Conference on Human Factors in Computing Systems Proceedings of the SIGCHI conference on Human factors in computing systems*.
- Raskar, Ramesh, & Cohen, Michael. 1999. Image precision silhouette edges. *In: Symposium on Interactive 3D Graphics Proceedings of the 1999 symposium on Interactive 3D graphics*. Atlanta, Georgia: ACM Press.
- Resnick, Robert, Halliday, David, Krane, Kenneth S., & Stanley, Paul. 2002. *Physics*. 5 edn. Vol. 1. New York: John Wiley & Sons, Inc.
- Roberts, Brian K., & Reardon, Jim. 1991 (April 11). *The Simpsons: Brush with Greatness*.
- Robertson, Barbara. 1994. Disney Lets CAPS Out of the Bag. *Computer Graphics World*, 58 – 64.
- Robertson, Barbara. 1999. Deep Background: New Technology Helps a 2D Tarzan Swing Through a 3D Jungle. *Computer Graphics World*, July 1, 50–51.

- Rushmeier, Holly, Gomes, Jose, Balmelli, Laurent, Bernardini, Fausto, & Taubin, Gabriel. 2003. Image-Based Object Editing. *Page 20 of: Fourth International Conference on 3-D Digital Imaging and Modeling (3DIM '03)*.
- Russell, Bertrand. 1927. *An Outline of Philosophy*. London: George Allen and Unwin.
- Russell, Stuart J., Norvig, Peter, Canny, John F., Edwards, Douglas D., Malik, Jitendra M., & Thrun, Sebastian. 2003. *Artificial Intelligence A Modern Approach*. 2nd edn. New Jersey: Prentice Hall.
- Sabbatini, Renato M.E. 2003 (Feb 23, 2003). *The Neuron Doctrine*.
- Saito, Takafumi, & Takahashi, Tokiichiro. 1990. Comprehensible Rendering of 3-D Shapes. *Computer Graphics*, **24**(4), 197–206.
- Saund, Eric, & Lank, Edward. 2003. Stylus Input and Editing Without Prior Selection of Mode. *Pages 213 – 216 of: Symposium on User Interface Software and Technology archive Proceedings of the 16th annual ACM symposium on User interface software and technology*.
- Schkolne, Steven, Pruett, Michael, & Schröder, Peter. 2001. Surface Drawing: Creating Organic 3D Shapes with the Hand and Tangible Tools. *In: Proceedings of CHI 2001*.
- Schmidt, Ryan, Wyvill, Brian, Sousa, Mario Costa, & Jorge, Joaquim A. 2005. ShapeShop: Sketch-Based Solid Modeling with the BlobTree. *In: 2nd Eurographics Workshop on Sketch-based Interfaces and Modeling*.
- Schneider, Philip J. 1990. An Algorithm for Automatically Fitting Digitized



- Curves. *Page 833 of:* Glassner, Andrew S. (ed), *Graphics Gems*, 1 edn. Graphics Gems. San Diego, California, U.S.A.: Academic Press, Inc.
- Schroeder, Will, Martin, Ken, Lorensen, Bill, Avila, Lisa Sobireajski, Avila, Rick, & Law, C. Charles. 2006. *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*. 4 edn. Colombia: Kitware, Inc.
- Searle, John R. 1997. *The Mystery of Consciousness*. 1st edn. New York: New York Review Books.
- Sederberg, Thomas W., & Greenwood, Eugene. 1995. Shape Blending of 2-D Piecewise Curve. *In:* Lyche, Tom, & Schumaker, Larry L. (eds), *Mathematical Methods in CAGD III*. Academic Press.
- Shesh, Amit, & Chen, Baoquan. 2004. SMARTPAPER—An Interactive and Easy-to-use Sketching System. *Pages 301–310 of:* Cani, M.-P., & Slater, M. (eds), *Proceedings of Eurographics 2004*, vol. 23.
- Simpson, J.A., & Weiner, E.S.C. (eds). 1989. *sketch, n., sketch, v., sketching, vbl.* n. 2 edn. Oxford English Dictionary Online. Oxford: Clarendon Press, Oxford University Press.
- Singh, Karan, & Fiume, Eugene. 1998. Wires: A Geometric Deformation Technique. *Computer Graphics*, **32**, 405–414.
- Skymatter. 2007. *Mudbox 1.06*.
- Sloan, Peter-Pike J., Martin, William, Gooch, Amy, & Gooch, Bruce. 2001. The Lit Sphere: A Model for Capturing NPR Shading from Art. *GRIN'01: Graphics Interface 2001*, 143–150.

- Snyder, John M. 1992. *Generative Modeling for Computer Graphics and CAD: Symbolic Shape Design Using Interval Analysis*. San Diego: Academic Press.
- Software, @Last. 2007. *SketchUp*.
- Spice, Byron, & Watzman, Anne. 2006 (September 24, 2006). *Carnegie Mellon Researchers Teach Computers To Perceive Three Dimensions in 2-D Images*.
- Story, Derrick. 2000 (April 8, 2006). *From Dark-room to Desktop – How Photoshop Came to Light*.  
[http://www.storyphoto.com/multimedia/multimedia\\_photoshop.html](http://www.storyphoto.com/multimedia/multimedia_photoshop.html).
- Strothotte, Thomas, Preim, Bernhard, Raab, Andreas, Schumann, Jutta, & Forsey, David R. 1994. How to Render Frames and Influence People. *Computer Graphics Forum*, **13**(3), 455–466.
- Sutherland, Ivan. 1963. Sketchpad: A man-machine graphical communication system. *Pages 329 – 346 of: AFIPS Spring Joint Computer Conference*.
- Tai, Chiew-Lan, Zhang, Hongxin, & Fong, Jacky Chun-Kin. 2004. Prototype Modeling from Sketched Silhouettes based on Convolution Surfaces. *Computer Graphics Forum*, **23**(1), 71–83.
- Tateosian, Laura G., & Healey, Christopher G. 2004. *NPR: Art Enhancing Computer Graphics*. Tech. rept. Department of Computer Science, North Carolina State University.
- Taub, Eric A. 2001 (March 1, Thursday). *Small Worlds to Create Bold, New Ones*.

- Tolba, Osama, Dorsey, Julie, & McMillan, Leonard. 2001. A projective drawing system. *In: Proceedings of the ACM Symposium on Interactive 3D Graphics, I3D 2001*. ACM-Press.
- Toon Boom, Animation Inc. 2006 (August 16, 2006). *2D Animation Software / Toon Boom*. <http://www.toonboom.com/>.
- Tsang, Steve, Balakrishnan, Ravin, Singh, Karan, & Ranjan, Abhishek. 2004. A suggestive interface for image guided 3D sketching. *Pages 591–598 of: Conference on Human Factors in Computing Systems Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM Press.
- Turk, Greg, & O'Brien, James F. 1999 (May). *Variational Implicit Surfaces*. Tech Report GIT-GVU-99-15. Georgia Institute of Technology.
- Turner, Alasdair, Chapman, David, & Penn, Alan. 1999. Sketching a virtual environment: modeling using line-drawing interpretation. *Pages 155–161 of: Proceedings of the ACM Symposium on Virtual Reality Software and Technology*.
- USGS, United States Geological Survey. 2003. *Topographic Map Of The West Candor Chasma Region Of Mars - MTM 500k -5/282E OMKT*.
- Varley, P. A. C., Martin, R. R., & Suzuki, H. 2004a. Can Machines Interpret Line Drawings? *In: Hughes, John F., & Jorge, Joaquim A. (eds), EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling*.
- Varley, P. A. C., Takahashi, Y., Mitani, J., & Suzuki, H. 2004b. A Two-Stage Approach for Interpreting Line Drawings of Curved Objects. *In: Hughes, John F., & J, Joaquim A. (eds), EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling*.

- Wang, Sidney W., & Kaufman, Arie E. 1995. Volume Sculpting. *Pages 151 – ff of: Proceedings of the 1995 Symposium on Interactive 3D Graphics.*
- White, Tony. 1988. *The Animator's Workbook*. 2nd edn. New York: Watson-Guptill Publications.
- Williams, Lance. 1990. 3D Paint. *Pages 225–233 of: Symposium on Interactive 3D Graphics archive Proceedings of the 1990 symposium on Interactive 3D graphics, vol. 24.*
- Williams, Lance R. 1997. Topological Reconstruction of a Smooth Manifold-Solid from Its Occluding Contour. *International Journal of Computer Vision*, **23**(1), 93 – 108.
- Witkin, Andrew P., & Heckbert, Paul S. 1994. Using Particles to Sample and Control Implicit Surfaces. *Pages 269 – 277 of: International Conference on Computer Graphics and Interactive Techniques.*
- Wood, Daniel N., Finkelstein, Adam, Hughes, John F., Thayer, Craig E., & Salesin, David H. 1997. Multiperspective Panoramas for Cel Animation. *Computer Graphics Proceedings of SIGGRAPH '97*, **31**, 243–250.
- Wyvill, Brian, & Guy, Andrew. 1998. The BlobTree - Warping, Blending and Boolean Operations in an Implicit Surface Modeling System. *Computer Science Technical Reports*, **1998-618-09**.
- Yaeger, Larry, Webb, Brandyn, & Lyon, Richard. 1998. Combining Neural Networks and Context-Driven Search for On-Line, Printed Handwriting Recognition in the Newton. *Lecture Notes In Computer Science*, **1524**, 275 – 298.

- Zelevnik, Robert. 1998. Interaction in 3D Graphics. *ACM SIGGRAPH*, **32**(4).
- Zelevnik, Robert C., Herndon, Kenneth P., & Hughes, John F. 1996. SKETCH: An Interface for Sketching 3D Scenes. *Pages 163–170 of: Rushmeier, Holly (ed), SIGGRAPH '96*. Addison Wesley.
- Zenka, Roman, & Slavik, Pavel. 2004. Estimating the “Mental Image” for Comprehensible Rendering of 3D Objects. *Pages 555–560 of: Eighth International Conference on Information Visualisation (IV'04)*. IEEE Computer Society.

# Appendix A

## Tablet Feature Examination

What follows are a set of tables summarizing examinations of various tablet features. Each table focuses on a single tablet input feature, and discusses its strengths, limitations, and possible uses as a channel of input.

**Table A.1.** A summary of examinations of hovering input to the digitizing tablet and their potential use as application input.

<b>Hovering</b>		
<b>Physical Movement</b>	<b>Application Input</b>	<b>Design Potential</b>
<ul style="list-style-type: none"> <li>• Initial user impression is to hover the entire arm, however this gets tiring and gives gross control.</li> <li>• It is more natural to pivot from the heel of the hand.</li> </ul>	<ul style="list-style-type: none"> <li>• Making strokes while hovering is difficult and inaccurate due to lack of tactile feedback.</li> <li>• Input must be position/movement based or triggered.</li> </ul>	<ul style="list-style-type: none"> <li>• Best suited for gross controls rather than artistic input.</li> </ul>

**Table A.2.** A summary tablet performance in pen-down operations and their potential use as application input.

<b>Pen-Down</b>		
<b>Physical Movement</b>	<b>Application Input</b>	<b>Design Potential</b>
<ul style="list-style-type: none"> <li>• Pen tip gives natural feeling of drawing.</li> <li>• Multiple clicks can be simulated with taps.</li> </ul>	<ul style="list-style-type: none"> <li>• Highly granular positional information.</li> <li>• Pen-style mapped input makes drawing natural, but general mousing tasks tiring.</li> <li>• Multiple clicks as taps must be resilient to pointer movement.</li> </ul>	<ul style="list-style-type: none"> <li>• Best suited for detailed input or drawing functions.</li> <li>• Tapping could be used as a modal switch.</li> </ul>
<ul style="list-style-type: none"> <li>• Eraser tip has slippery feel on tablet surface</li> <li>• Exact eraser tip location is harder to judge.</li> <li>• Pen-style grip for eraser end does not fit pen ergonomics, but is usable.</li> <li>• A flipped grip can be used to quickly transition between pen and eraser.</li> </ul>	<ul style="list-style-type: none"> <li>• Same granular position information.</li> <li>• Barrel buttons are inaccessible from standard grip.</li> <li>• Flipped grip may allow barrel button use, but may also lead to accidental clicks.</li> <li>• Flipped grip is unstable and unsteady after short periods.</li> </ul>	<ul style="list-style-type: none"> <li>• Used in pen grip, well suited for shorter term drawing activities.</li> <li>• Used in flip grip, only gross input tasks are appropriate.</li> </ul>

**Table A.3.** A summary tablet pressure input features and their potential use as application input.

<b>Pressure</b>		
<b>Physical Movement</b>	<b>Application Input</b>	<b>Design Potential</b>
<ul style="list-style-type: none"> <li>• Pressure mechanism creates a slight give on pen down that can be distracting.</li> <li>• Give for eraser is more pronounced.</li> <li>• Fine differences in pressure levels are difficult to conscientiously control.</li> <li>• Isolated high-pressure events are easy to perform.</li> </ul>	<ul style="list-style-type: none"> <li>• Pressure information is only available on left click operations.</li> <li>• Pen tip pressure levels are variable from user to user, session to session.</li> <li>• Pressure data is always dynamic and must pass through zero at the beginning and end of each stroke.</li> <li>• Pressure focused operations must be differentiated from strokes.</li> </ul>	<ul style="list-style-type: none"> <li>• Pressure coupled with stroke information is ideal for media simulation.</li> <li>• Apart from strokes, pressure is useful as a gross dynamic adjustment to a normalized parameter.</li> <li>• Isolated high-pressure events can be used for modal switching.</li> </ul>
<ul style="list-style-type: none"> <li>• Pressure is unavailable while hovering.</li> </ul>	n/a	n/a



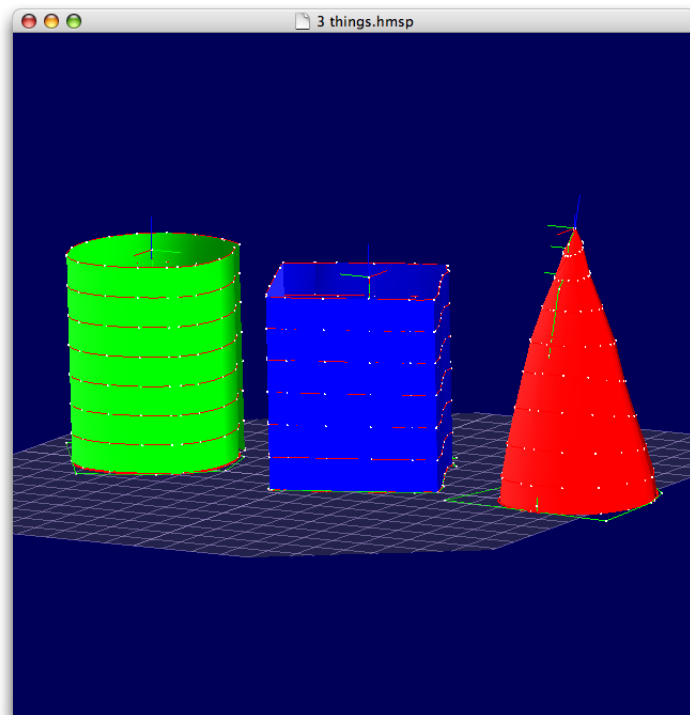
**Table A.4.** A summary of stylus orientation information produced by the digitizing tablet and its potential use as application input.

<b>Tilt</b>		
<b>Physical Movement</b>	<b>Application Input</b>	<b>Design Potential</b>
<ul style="list-style-type: none"> <li>• Long term angular adjustment while hovering requires resting the arm or wrist on the tablet.</li> <li>• Rotating wrist/arm to adjust pen angle is uncomfortable. Digital manipulation is easier.</li> <li>• Pen can be comfortably held in both pen-style grip and alternate pointing style grip.</li> <li>• In alternate grips the barrel buttons are less accessible.</li> </ul>	<ul style="list-style-type: none"> <li>• Range of tilt is limited by proximity.</li> <li>• Tilt angle can be used to differentiate grip.</li> <li>• Must be used with a keyboard action in lieu of barrel button activation.</li> <li>• Dynamic tilt information is inaccurate while hovering due to event coalescing, though generally correct.</li> </ul>	<ul style="list-style-type: none"> <li>• Dynamic angle adjustment is useful for dynamic rotations.</li> <li>• Static pen angles simulate multiple tools by simply changing grip.</li> </ul>
<ul style="list-style-type: none"> <li>• Standard pen grips limits intentional angle adjustment while drawing on tablet surface to a small range.</li> <li>• While drawing, the pen tilt changes naturally.</li> <li>• Fist grip gives the feeling of joystick controls. Range of x-axis motion is limited.</li> <li>• Extremely low y-angle grip fans out the hand as if manipulating drawing surface.</li> </ul>	<ul style="list-style-type: none"> <li>• In pen grip, y-axis tilt can differentiate handedness.</li> <li>• Dynamic tilt information when pen is down is highly accurate.</li> <li>• Barrel buttons are accessible for pen-style grip over a large range of tilt.</li> <li>• Inadvertent barrel button clicks are likely in a joystick grip.</li> <li>• Inadvertent tip movement is likely with a joystick grip.</li> </ul>	<ul style="list-style-type: none"> <li>• Dynamic angle adjustment during strokes can provide tool dynamics.</li> <li>• Joystick grip may be useful as a position or rotation control.</li> <li>• Tilt thresholding could be used to trigger tool or mode changes.</li> </ul>

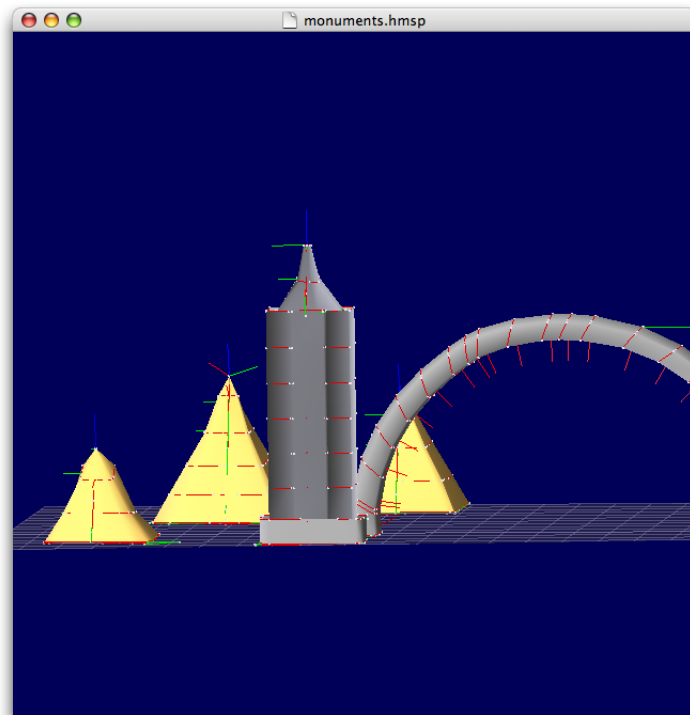
# Appendix B

## Modeling Examples

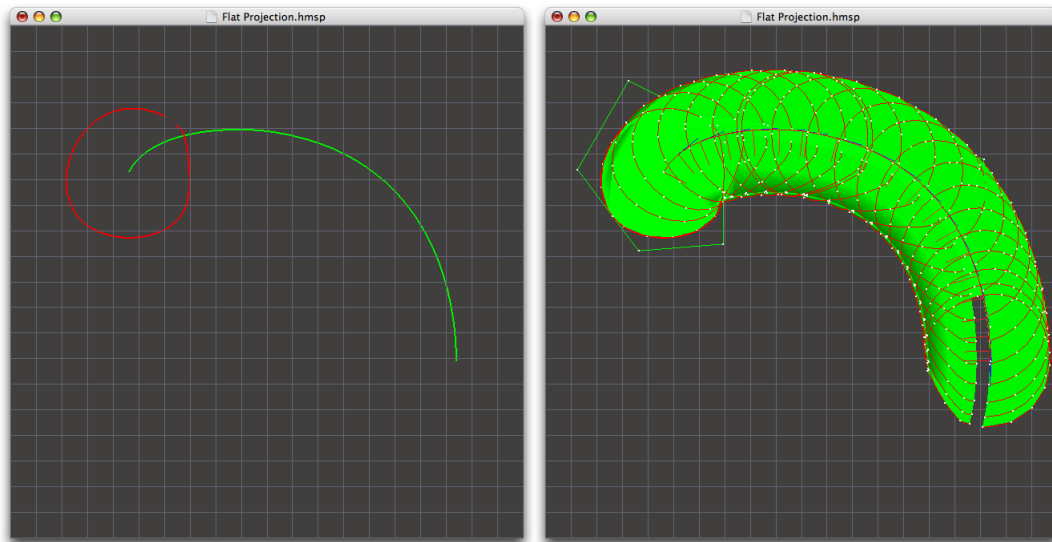
This appendix provides a number of example models created with the modeling system. Each model image is accompanied by a short description.



**Figure B.1.** This model contains three simple forms, a cylinder, box, and cone, demonstrating how basic shapes can be constructed simply. Both the box and cylinder were created using simple sweep constructions with only two strokes. The cone was created similar to the cylinder using the generalized cylinder construction method and three strokes.

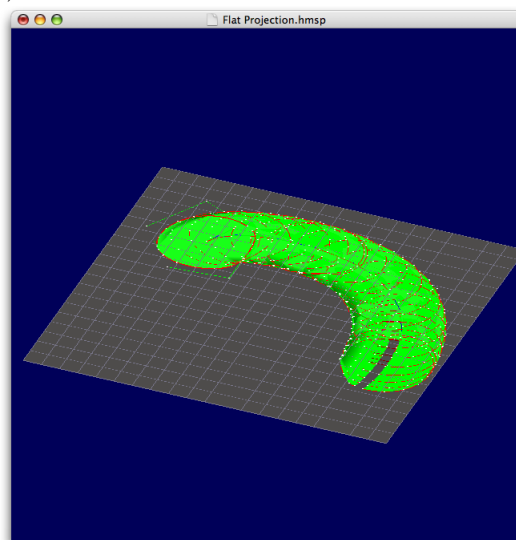


**Figure B.2.** This image demonstrates how architectural forms can be created with the system. Depicted are simple sketched models of a number of monuments. Along the back are three Egyptian pyramids, each created with simple generalized cylinder constructions. To the right is a sketch of the St. Louis Arch, created with a sweep. In the middle is a rough sketch of the Empire State Building. The building was created in sections beginning with a wide foundation, moving to an 'H' shaped mid section, and capping off with a tapering spire.



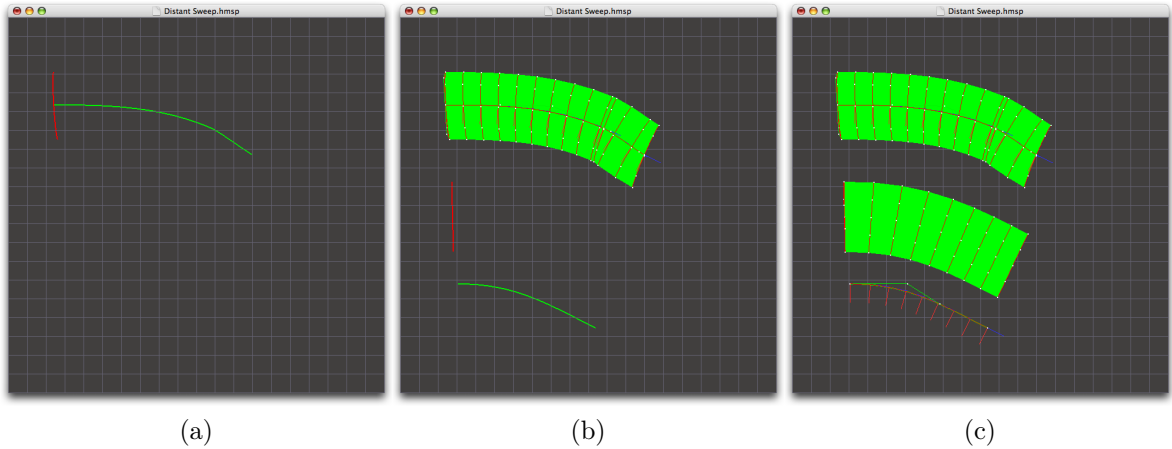
(a)

(b)



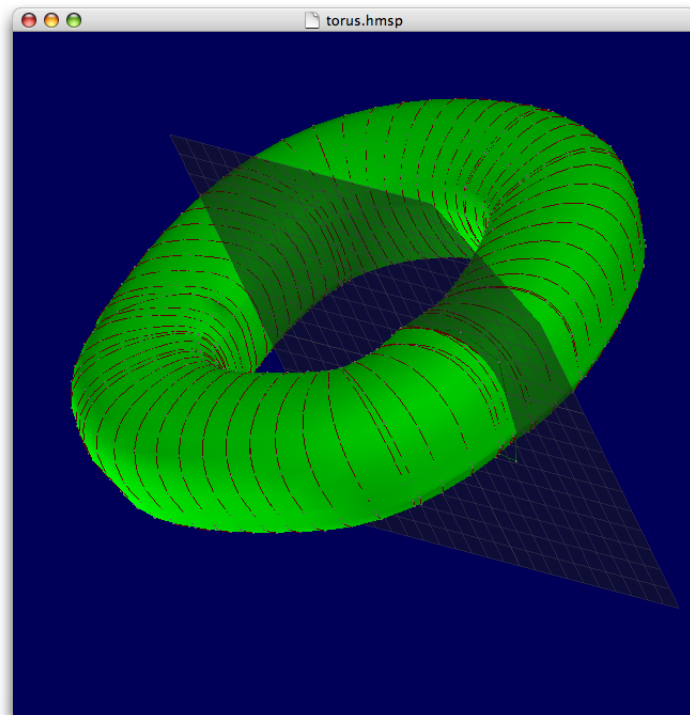
(c)

**Figure B.3.** Under normal circumstances, the plane on which the die stroke is drawn is differently oriented than the planes that hold the path and size stroke. However, there is no requirement that the active strokes supplied to the construction system be situated on separate planes or in different orientations. For example, the user can choose to define his or her path stroke on the same plane as the accompanying die stroke. This has the effect of drawing the die shape out along the flat surface of the plane, as if a 3-dimensional shape had been projected flat against the drawing plane. This technique can be used to create flat shapes such as fins, scales, leaves, feathers, or flat organic forms.



**Figure B.4.** When the system sweeps the die stroke shape out to generate 3-D structure, the resulting 3-D form starts from the position of the original die stroke and follows a path matching the displacement and orientation of the path stroke. This means that, even if the path stroke is drawn some distance away from the die stroke, the resulting 3-D volume will always start from the die stroke’s position. The translational path followed by the die contours simply takes cues from the shape of the path stroke, however orientation information for each contour is derived directly from the relationship between the path stroke and the die stroke. When the path stroke—and size stroke if defined—are drawn in close proximity to the die stroke, as is the usual case, then the resulting 3-D construction will have the expected form. By drawing the path stroke some distance away from the die stroke, or in different positions in nearby relation to the die stroke, the orientation effect of the path can be exaggerated or diminished, creating some interesting effects.

Here you can see how a short linear die stroke can be swept, both with a path stroke drawn in proximity to the die stroke as in the first image, and with a path stroke drawn some distance away. When drawn separate from the die stroke, we can more easily see the Frenet triads placed along the path stroke. Notice that if each triad’s red unit vectors were extended, they would align precisely with the red contour curves along the swept surface.



**Figure B.5.** As discussed in Section 6.4.8 on page 6.4.8, the closing feature was design to give users the ability to create geometrically closed die strokes so that resulting 3-D geometry will have the correct appearance. Although the closing feature was developed for the die pen, the same mechanism can be used with any stroke drawn by the user. This means that path strokes and size strokes can be closed as well. One possible application of this feature is to create a closed 3-dimensional shapes such as a torus, ring, or doughnut.