An "Over-The-Shoulder" Implementation

Marcia Ramos
Victor S. Frost

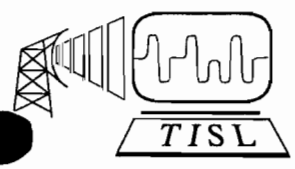
TISL Technical Report TISL-9770-14

Prepared for:

Defense Advanced Research Projects Agency/CSTO

Research on Gigabit Gateways
AARPA Order No. 8634
Issued by EDS/AVS under Contract #F19628-92-C-0080

May 1994



Telecommunications and Information Sciences Laboratory
The University of Kansas Center for Research, Inc.
2291 Irving Hill Drive Lawrence, Kansas 66045

An "Over-The-Shoulder" Implementation

May 5, 1994

Marcia Ramos
Victor S. Frost

TISL Technical Report TISL-9770-14

Sponsored by:
Defense Advanced Research Projects Agency/CSTO
Research on Gigabit Gateways
AARPA Order No. 8634
Issued by ESD/AVS under Contract #F19628-92-C-0080

Table of Contents

Abstract

The objective of this paper is to provide an overview of the implementation of an "Over-The-Shoulder" application. It describes the software and graphics issues involved in developing this application as well as the system requirements and how to run the program. The "Over-The-Shoulder" was designed to provide remote users in the MAGIC testbed with a view of the terrain while it is rendered by TerraVision. The TerraVision application is a terrain visualization of a landscape with superimposition of vehicles and features of a battlefield.

1. Introduction

The "Over-The-Shoulder" (OTS) software application is a client/server implementation of an interactive visualization of terrain images displayed by TerraVision [4]. The OTS permits a client at a remote location to view the terrain displayed by TerraVision in the server site. Several clients can run OTS at the same time and access the same images displayed by TerraVision. The software has been implemented for the DEC Alpha workstation, but it is portable to other environments if the software requirements are met. These requirements are described in Section 3.

The OTS displays both a low-resolution image of the full terrain viewed by TerraVision and a high-resolution image of a selected area of the terrain. These images are displayed in two separate windows, called "LowRes Window" and "HighRes Window." The windows' sizes are set to 512x512 and are resizable according to the user's demand. The windows can be resized with the mouse just like any other window. The area to be viewed in high resolution is selected when the user clicks the left-most mouse button in the low-resolution window display. The cursor inside the LowRes window is viewed as a box that corresponds to the area selected by the user. By clicking the mouse button inside the LowRes window, the user selects the upper-left coordinates of the HighRes area to be displayed.

OTS contains a user interface that allows the users to select the server host that is running TerraVision, OTS's client site, the window displayed by TerraVision that contains the image to be viewed by OTS, and other relevant information such as the LowRes Sample Rate and the LowRes Frame Rate. The LowRes Sample Rate is a user-defined parameter that indicates the sampling rate of the low-resolution image. The default sampling rate is 2, meaning that the low-resolution image is taken from every other pixel of the full terrain displayed in the TerraVision window. The LowRes Frame Rate indicates the time in seconds between each low-resolution image update.

The low-resolution image is therefore updated at a rate specified by the user, while the high-resolution image is constantly updated according to the user's selection of the LowRes area to be viewed in high resolution. If an area hasn't been selected yet by the user, the HighRes image is taken out of the previous coordinates. The default upper-left coordinate is (0, 0).

The OTS software was implemented in C using Xlib [1, 2] for the graphics and Motif [3] for the user interface. It is divided into three main programs: the user interface, the server code, and the client code. The algorithm and software implementation of the OTS software application are explained in detail in Section 2. Section 3 contains OTS's system requirements, such as the libraries used in the current implementation. Finally, Section 4 explains how to run OTS, and Section 5 closes with some conclusions and possible directions for later OTS implementations.

2. Software Algorithm & Implementation

The OTS software application is divided into three main parts, namely:

- User Interface;
- Server; and
- Client.

There are also other support modules, such as an adaptation of the *xwininfo* application in the XWindows system. All the programs were designed in a modular and user-friendly fashion, and they are fully documented to help the user in understanding the code. The programs were written in C using the Xlib functions for the graphics functions and the Motif functions for the user interface features.

The client/server communication is done via TCP/IP. It was designed to achieve the maximum frame rate possible with full utilization of the network and graphics resources. These three components were integrated into a main program that executes the user interface functions and then forks to run the client code in the local machine and the server remotely in the server host entered by the user. The server code thus runs in the same machine as the TerraVision application. The design and implementation of each of these programs is explained in detail in the following subsections.

2.1. The User Interface

The user interface was designed using Motif functions. It contains a simple menu with the self-explanatory "Quit" and "Help" options as well as an option called "File." In this menu option, there is a submenu titled "OTS View." This option is selected to start the client/server communications culminating in the display of the terrain images on the screen.

Inside the main window of the user interface, there are five user interaction boxes that the user fills with the appropriate inputs to OTS. These boxes are:

- Server Host;
- Client Host;
- TerraVision Window Name;
- LowRes Sample Rate; and
- LowRes Frame Rate.

When the user runs the program, these boxes as well as the main menu will appear on the screen with the cursor set in the first box, i.e., the Server Host box, so that the user can start entering the inputs. The Server Host is the address of the machine running the server and TerraVision, for example, onyx-atm.bcbl.magic.net. The

client host is the address of the machine where the OTS user interface and client are running, for example, mauchly.ukans.magic.net.

The third box is for the name of the window displayed by TerraVision that contains the image to be viewed at the client site. Currently, two windows are of interest to the user. They are called "Out The Window View" and "Overhead View." By typing one of these names in the user's interaction box, OTS will get the low-resolution and high-resolution images from this window.

The other two boxes concern the low-resolution image's parameters, namely the sample rate and the frame rate. As mentioned before, the sample rate indicates the sampling of the low-resolution image. A sample rate of 2 indicates that the low-resolution image is taken from every other pixel of the terrain image displayed in the TerraVision window. The other parameter, the frame rate, indicates the time in seconds between each update of the low-resolution image.

After entering all the correct inputs, the user then can select the "OTS View" option in the file menu to start visualizing the images. All the functions described here were implemented with simple Motif functions. The user interface code starts just like any other Motif application by initializing the X toolkit using the function *XtVaAppInitialize*. It then creates the main window that contains the menu and user interaction boxes, creates the menu, and creates the boxes. The boxes were created using the "rowcol" widgets defined by Motif, which facilitated the placement and alignment of the boxes on the screen as well as the grabbing of the user's inputs.

Each box is associated with a callback function that gets the appropriate input entered by the user. Initially, the cursor is placed in the first box, i.e., the Server Host box. When the user enters the name of the server host and presses "Return" to enter the next input, the X toolkit passes control to the callback function of the Server Host and gets the name of the server host entered by the user. This process repeats with all the other user interaction boxes.

When all inputs are entered, and the user is ready to start visualizing the images, the user then can select the "OTS View" option in the "File" pull-down menu. Just like the user widgets, each option in the menu, "Quit," "File," and "Help," is associated with a callback function that will perform the operations desired. The "Quit" callback function contains a safety widget that pops up an "Are you sure?" message to the user in case this widget was called at an inappropriate time. The help widget currently contain only one message but can be expanded to display a full help for the whole OTS code.

The file callback function is called when the "OTS View" option is selected. This callback function, called *file_cb*, is the most important function of the program, for it is when this function is called that the image visualization starts. The function's implementation is quite simple: it contains a fork() to the client process and the server process, with the server as the parent and the client as a child. The server is

run remotely at the server host location entered by the user by doing an *execl* function call. The client is run by just calling the function *ClientSide* that implements all the client operations. Notice that the client executable code needs to reside on the client host while the server executable code needs to reside on the server host. When calling the server remotely using the *execl* function, it is necessary that the appropriate path where for server code is given.

The next section explains in detail the client/server communications.

2.2. Server/Client Communications

The server/client communications is done using TCP/IP. When the server starts running remotely at the server host, it allows connections to all the clients and waits until the client program establishes a connection to the server. The connection is done through the ports 21000 as the TCPServerPort and 22000 for the TCPClientPort. When running more clients, the client ports need to be different from one another. The TCP operations for the server were implemented in the function called *CreateTCPSocket* inside the server program.

The client starts by allowing connections from all servers, implemented in the function *CreateTCPClientSocket* inside the client code, and by using the *connect* function to establish the communications link with the server. This was implemented in the function called *CreateTCPServerSocket* inside the client code.

Once this communication link is established between server and client, the OTS visualization process starts. Several implementations were considered before adopting the current approach of using the X functions *XGetImage* and *XPutImage* to grab and display the images at the client site. This approach was adopted after trying UDP to transmit the image data, which caused a considerable loss of data, and after using TCP/IP to do the data transmission. The latter approach, although being more reasonable than giving full control to the X functions, was discarded due to lack of interaction with the TerraVision application. In order to do the data transmission using TCP/IP, it is necessary that the data be grabbed directly from the frame buffer of the Onyx machine, thus requiring some coding inside the TerraVision application to pass the location of the data in memory to the OTS application. The X approach was then adopted to avoid this interaction and to test the capabilities of the X functions to handle this problem. After testing the OTS this approach proved to be workable at the moment. Not only did it not require any extra coding inside the TerraVision application, but it also resulted in fairly reasonable speeds when tested over the ATM link of the MAGIC testbed.

With this approach, the server starts by doing an *XGetImage* of the low resolution image being displayed by TerraVision and an *XPutImage* to display the image inside the LowRes window created by the client code. In order to do an *XGetImage*, the server needs to get TerraVision's window information. This is done by using an

adaptation of the xwininfo X application. With only the window name, this function returns the window ID as well the size information required by *XGetImage* to function properly. For *XPutImage* to work, the server needs the window IDs of the LowRes window and the HighRes window created by the client. This is done inside the client code right after the windows are created by sending the window IDs through the TCP socket created earlier.

The server then samples the low-resolution image to display it in the LowRes window according to the LowRes Sample Rate entered by the user. The server then updates the high-resolution image, constantly taking care of the user's requests for a specific area of the screen. As mentioned in Section 1, the user selects this area by just pressing the left-most mouse button, thus giving the upper-left coordinates of the sampling area. The user has a feeling for the area being selected as the cursor inside the LowRes window was transformed into a 64x64 box (the maximum cursor size allowed by a DEC3000 Alpha workstation). This approach was preferred over drawing rubber-band boxes each time so that the speed in processing the images is maximized.

The server then updates the high-resolution image according to the coordinates selected by the user. If no requests have been made, the server simply assumes the previous coordinates selected in order to update the image (the default initially is (0,0) for the upper-left coordinates). The images are updated by just doing an XPutImage to the corresponding window. After the LowRes frame time entered by the user has passed, the server code updates the low-resolution image. It then continues to update the high-resolution images, constantly taking care of the client's requests.

The next subsections explain the server code and the client code implementations in more detail.

2.3. The Server Code

The server code contains 5 main functions as follows:

- CreateTCPSocket;
- GetServerImage;
- SampleLowResImage;
- GetHighResImage; and
- ServerSide.

The *CreateTCPSocket* function, as explained in the previous subsection, handles the TCP operations required to establish communications with the client program. It uses simple socket functions to create the TCP socket, and it waits for a connection from a client by doing a *listen* on the socket. The *CreateTCPSocket* function was designed to allow connections from all clients as long as the ports have been defined

properly (one port for each different client).

The *GetServerImage* simply does an *XGetImage* on the appropriate window name entered by the user. This function grabs the whole image being displayed in the TerraVision window. After the image data is grabbed, the image is sampled according to the LowRes Sample Rate entered by the user. This was implemented in the SampleLowRes function. The sampling is done by using the X functions *XGetPixel* and *XPutPixel* to create the low resolution image data structure from the whole image grabbed from the TerraVision window. The default window size is 512x512, so the image will be sampled to fit this window. If the image is smaller than 512x512, the user can easily resize the window by using the mouse. The image is then displayed by doing an *XPutImage* to the LowRes window being displayed at the client site.

The next function, *GetHighResImage*, gets the high-resolution image from the low-resolution image displayed at the LowRes window according to the latest coordinates selected by the user. The mapping from the LowRes to the HighRes window is done so that the high-resolution image fits a 512x512 window, making sure that the upper-left coordinates will not extrapolate the 512x512 size. The function gets the image with an *XGetImage* and displays it in the HighRes window using an *XPutImage*. The high-resolution frame rate is also calculated in this function. The frame rate is calculated by getting the frames displayed over the time period specified by the user as a command line argument.

The *ServerSide* function is the main function of the server code; it controls all the communications with the client and calls the functions described above. The function starts by establishing the connection with the client and by getting the TerraVision window ID using the function xwininfo. It then opens a display connection to the client so that the *XPutImage* can be executed properly. The server then gets the LowRes window ID and the HighRes window ID by receiving this information from the TCP socket. It was necessary to change the byte order of the IDs since the byte order of the Onyx machine (TerraVision) is not the same as the byte order of the Alpha workstation being used as the client site.

The server code then loops indefinitely, listening to the user's requests for high-resolution images and calling the functions described above to display the images. Updates of the LowRes image are controlled in this loop by checking the timing between each frame.

The next subsection explains the client code in more detail.

2.4. The Client Code

The client code contains 7 main functions as follows:

- CreateWindow;
- CreateTCPServerSocket;
- CreateTCPClientSocket;
- SendRequest;
- HandleEvents;
- HandleHighResEvents; and
- ClientSide.

The *CreateWindow* function does the graphics set-up for the display of the images. It creates two 512x512 windows, one for the LowRes display and the other for the HighRes display. Each window is associated with a GC (graphics context), foreground, and background colors. The colormap created for these windows is a simple gray-scale colormap that can be improved to adopt full color in future implementations.

The *CreateTCPServerSocket* and *CreateTCPClientSocket* functions are used for the TCP communications with the server. The second one allows connection from any server, and the first function is called later to establish the connection with the server code via the ports 21000 and 22000, as described earlier. These functions were implemented with simple functions defined in the socket libraries.

The *SendRequest* function handles the transmission of image requests to the server. Each request packet contains the image type (1 for LowRes and 2 for HighRes), the upper-left coordinates of the image requested, and the height and width of the box displayed in the LowRes window, in this case, 64x64. The height and width were added to the data structure in case this box is implemented later as resizable rubber-band rectangles. The RequestPacket was defined in the include file called *Packets.h*. The function takes the image type, the upper-left coordinates, and the height and width of the box as arguments, and constructs the RequestPacket that is sent through the TCP socket using the send command.

The *HandleEvents* function requests the initial low-resolution image to be displayed in the LowRes window and then calls the *HandleHighResEvents* function that controls all user requests for the high-resolution image. The *HandleHighResEvents* function consists of a normal X event loop containing Expose, ButtonPress, ButtonRelease, MotionNotify, and default events. It loops, listening to requests in the queue and processing the requests in the order they occur. Every time a mouse button is pressed, the ButtonPress event is called, and the coordinates selected by the user are grabbed. They are then sent to the server using the function SendRequest.

The ClientSide function, just like the ServerSide function described earlier, is the main function of the client code; it also controls all the communications with the

server and calls the functions described above. It starts by establishing the TCP connection with the server and initializing the graphics set-up via the *CreateWindow* function. It then defines the cursor as a 64x64 box to represent the area in the LowRes window selected by the user to be displayed in high resolution. The function then loops forever, processing the events generated by the user.

The next section describes the programs used in the OTS implementation, where they are residing at the moment, and all the libraries used in the implementation.

3. System Requirements

The current version is set up to run the server code remotely on an Onyx at the BCBL. In order to run OTS with another server, a copy of the server code needs to reside in the new server host and the path to this code changed in the *execl* function of the main program (*OTS.c*). The following list contains all the programs used in the OTS implementation:

OTS.c	Main program with the user interface and calls to the server and client functions (Appendix A)
OTS_client.c	Client code (Appendix B)
OTS_server.c	Server code (Appendix C)
Packets.h	RequestPacket data structure
xwininfo.h	xwininfo application code
dsimple.c	Functions used in xwininfo
makeserver	Make file for the server code (Appendix D)
makeots	Make file for the client code (Appendix D)

The current versions of the operating system, Xlib, and Motif are listed below:

OS	OSF/1 v. 2.0
Xlib	X release 5
Motif	OSF/Motif 1.1

A brief description on how to set up a demo between KU and BCBL is given in the next section.

4. How To Run OTS

To run OTS at KU, go to the directory containing the application and type "OTS." OTS can take two arguments. The second one, "-d," is a debug option, and the first argument is the time interval for the calculation of the HighRes Frame Rate.

The user will see a blue window with the main menu and the user interaction displayed at his/her client site. The user should then enter the parameters requested by the OTS program and select the "OTS View" option in the "File" menu to start visualizing the images. Again, to select a HighRes area of the low-resolution image, the user just needs to press the left-most mouse button. In entering the parameters, the user should be careful to give the host names correctly.

The next section gives some recommendations for future OTS implementations.

5. Conclusions & Future Directions

The OTS was successfully implemented and tested over the ATM link between KU and BCBL. The high-resolution frame rates were about 2 frames/sec, which corresponds to an application data rate of 4.2 Mbps. Note that the maximum rate at which frames can be written on the client workstation is about 15. Future implementations can look into ways of improving this. The main limitations are due to the X functions in handling the display and grabbing of the images and the rate at which frames can be written at the client workstation.

One approach to be tested in the future is to use the frame buffer of the SGI machine to get the image data rather than using the XGetImage function. Another added feature to OTS is an improved colormap displaying color images. Future implementations can also look into ways of displaying the HighRes Frame Rates into boxes in the main window display rather than as just standard output to the screen.

Overall, the OTS code achieved its main goals, but it can certainly be improved in future implementations attending the recommendations mentioned in this document.

6. References

[1] "Xlib Reference Guide," O'Reilly & Associates, 1991.

[2] "Xlib User's Guide," O'Reilly & Associates, 1991.

[3] "Motif Programming Guide," O'Reilly & Associates, 1991.

[4] Leclerc, Y.G., and S.Q. Lau, "TerraVision: A Terrain Visualization System," Technical Note 540, SRI International, Menlo Park, California, March 1994.

```
/*-------------------------------------------------*/
/*        Over the Shoulder Application            */
/*                                                 */
/*           The University of Kansas              */
/*      Telecommunications and Information         */
/*                Sciences Lab                     */
/*                                                 */
/*  Module:       OTS.c                            */
/*  Author:       Marcia G. Ramos                  */
/*  Date:         April 02, 1994                   */
/*  Version:      2.0                              */
/*-------------------------------------------------*/

  /* C libraries */

#include <stdio.h>
#include <string.h>

  /* X libraries */

#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>

  /* Motif libraries */

#include <Xm/LabelG.h>
#include <Xm/RowColumn.h>
#include <Xm/Text.h>
#include <Xm/TextF.h>
#include <Xm/MainW.h>
#include <Xm/MessageB.h>
#include <Xm/Label.h>
#include <Xm/FileSB.h>
#include <Xm/SelectioB.h>

  /* User interaction boxes */

char *text_labels[] = {"TerraVision Server:", "Client Host:",
                       "TerraVision Window:",
                       "LowRes Sample Rate (2, 4, or 8):",
                       "LowRes Frame Rate:"};

char *text_widget[] = {"HighRes Frame Rate:", "HighRes Bit Rate:"};

  /* Help message */

#define help_msg        "Oops! You're in trouble!"

/*-------------------------------------------------------------------------*/

        /* Global variables */

        Widget          toplevel, main_w, w;

        char            *ServerHost;            /* User inputs */
        char            *ClientHost;
        char            *WindowName;
        char            *LowResSample;
        char            *LowResFrame;

        int             fork(void);

        int             dbug = 0;
```

```
/*-------------------------------------------------------------------------*/

void    quit_cb ()
{
        /*---------------------------------------------------------------*/
        /*  This is the callback function for the quit menu.             */
        /*---------------------------------------------------------------*/

        static  Widget  dialog;

        if (!dialog)
        {
          exit (0);
        }
}

/*-------------------------------------------------------------------------*/

void    file_cb ()
{
        /*---------------------------------------------------------------*/
        /*  This is the callback function for the file menu.             */
        /*---------------------------------------------------------------*/

        static  Widget  dialog;
        Widget          popdialog;
        Arg             args[3];
        XmString        msg;


        if (dbug == 1)
          printf ("Start!\n");

        msg = XmStringCreateSimple ("HighRes Frame:");

        if (dbug == 1)
          printf ("Start!\n");

        /* Select "View OTS" and call Client program */

        if (!dialog)
        {
          if (dbug == 1)
            printf ("View!\n");

          /* Run server on appropriate host */

          if (dbug == 1)
          {
            printf ("ServerHost = %s\n", ServerHost);
            printf ("ClientHost = %s\n", ClientHost);
            printf ("LowResSample = %s\n", LowResSample);
            printf ("LowResFrame = %s\n", LowResFrame);
          }

          if (fork() == 0)
          {
            if (dbug == 1)
              printf ("parent!\n");
            ClientSide (ServerHost);
          }
          else
          {
            if (dbug == 1)
            {
              printf ("child!\n");
```

```
            printf ("host = %s\n", ServerHost);
            printf ("window name = %s\n", WindowName);
          }
          execl ("/usr/bin/rsh", ServerHost,
                 "/usr/people/mramos/OTS_server",
                 ServerHost, ClientHost, WindowName, LowResSample,
                 LowResFrame, 0);
        }
      }
}

/*-------------------------------------------------------------------------*/

void    help_cb ()
{
      /*---------------------------------------------------------------*/
      /*  This is the callback function for the help menu.             */
      /*---------------------------------------------------------------*/

      static  Widget  dialog;
      Arg             args[1];
      XmString        help_message;

      /* Display help message */

      if (dbug == 1)
        printf ("Help!\n");
      if (!dialog)
      {
        if (dbug == 1)
          printf ("Help---\n");
        help_message = XmStringCreateLtoR (help_msg, XmSTRING_DEFAULT_CHARSET);
        XtSetArg (args[0], XmNmessageString, help_message);
        dialog = XmCreateInformationDialog (toplevel, "help-dialog", args, 1);
      }

      XtManageChild (dialog);
      XtPopup (XtParent(dialog), XtGrabNone);
}

/*-------------------------------------------------------------------------*/

void    ServerHost_cb (widget, client_data, cbs)
Widget                          widget;
XtPointer                       client_data;
XmRowColumnCallbackStruct       *cbs;
{
      /*---------------------------------------------------------------*/
      /*  This is the callback function for each user interaction box */
      /*---------------------------------------------------------------*/

      Widget  pb = cbs->widget;

      if (dbug == 1)
        printf ("calles!\n");
      ServerHost = XmTextGetString (widget);
      if (dbug == 1)
      {
        printf ("ServerHost = %s\n", ServerHost);
        printf ("%s: %d\n", XtName (pb), cbs->data);
      }
}

/*-------------------------------------------------------------------------*/

void    ClientHost_cb (widget, client_data, cbs)
```

```
Widget                          widget;
XtPointer                       client_data;
XmRowColumnCallbackStruct       *cbs;
{
        /*-----------------------------------------------------------------*/
        /*  This is the callback function for each user interaction box */
        /*-----------------------------------------------------------------*/

        Widget  pb = cbs->widget;

        if (dbug == 1)
          printf ("calles!\n");
        ClientHost = XmTextGetString (widget);
        if (dbug == 1)
        {
          printf ("ClientHost = %s\n", ClientHost);
          printf ("%s: %d\n", XtName (pb), cbs->data);
        }
}

/*-------------------------------------------------------------------------*/

void    WindowName_cb (widget, client_data, cbs)
Widget                          widget;
XtPointer                       client_data;
XmRowColumnCallbackStruct       *cbs;
{
        /*-----------------------------------------------------------------*/
        /*  This is the callback function for each user interaction box */
        /*-----------------------------------------------------------------*/

        Widget  pb = cbs->widget;

        if (dbug == 1)
          printf ("calles!\n");
        WindowName = XmTextGetString (widget);
        if (WindowName == "Out The Window View")
          WindowName = "Out_The_Window_View";
        if (WindowName == "Overhead View")
          WindowName = "Overhead_View";
        if (dbug == 1)
        {
          printf ("WindowName = %s\n", WindowName);
          printf ("%s: %d\n", XtName (pb), cbs->data);
        }
}

/*-------------------------------------------------------------------------*/

void    LowResSample_cb (widget, client_data, cbs)
Widget                          widget;
XtPointer                       client_data;
XmRowColumnCallbackStruct       *cbs;
{
        /*-----------------------------------------------------------------*/
        /*  This is the callback function for each user interaction box */
        /*-----------------------------------------------------------------*/

        Widget  pb = cbs->widget;

        if (dbug == 1)
          printf ("calles!\n");
        LowResSample = XmTextGetString (widget);
        if (dbug == 1)
          printf ("text = %s\n", LowResSample);
        if (dbug == 1)
```

```
          printf ("%s: %d\n", XtName (pb), cbs->data);
}

/*------------------------------------------------------------------------*/

void    LowResFrame_cb (widget, client_data, cbs)
Widget                          widget;
XtPointer                       client_data;
XmRowColumnCallbackStruct       *cbs;
{
        /*---------------------------------------------------------------*/
        /*  This is the callback function for each user interaction box */
        /*---------------------------------------------------------------*/

        Widget  pb = cbs->widget;

        if (dbug == 1)
          printf ("calles!\n");
        LowResFrame = XmTextGetString (widget);
        if (dbug == 1)
          printf ("text = %s\n", LowResFrame);
        if (dbug == 1)
          printf ("%s: %d\n", XtName (pb), cbs->data);
}

/*------------------------------------------------------------------------*/

main    (argc, argv)
int     argc;
char    *argv[];
{
        /*---------------------------------------------------------------*/
        /*  Main routine - controls the interface and callbacks          */
        /*---------------------------------------------------------------*/

        Widget          menubar, widget, pane, rowcol;
        XtAppContext    app;
        char            buf[8];
        int             i;
        XmString        file, quit, help, ots, about;
        Arg             args[10], targs[9];
        int             n;
        void            file_cb(), help_cb();
        char            *text;
        char            HostName[64];
        char            ifbug;

        /* Get debug info */

        if (argv[2])
        {
          printf ("BUG SET!\n");
          dbug = 1;
        }

        /* Initialize toolkit */

        if (dbug == 1)
          printf ("toplevel!\n");

        toplevel = XtVaAppInitialize (&app, "Demos", NULL, 0,
                                      &argc, argv, NULL,
                                      XmNwidth, 1268, XmNheight, 1024, NULL);

        /* Create main window */
```

```
if (dbug == 1)
  printf ("main_window!\n");

main_w = XtVaCreateWidget ("main_w", xmMainWindowWidgetClass,
                           toplevel,
                           NULL);

/* Create a menu bar with three items */

if (dbug == 1)
  printf ("Create string!\n");

quit = XmStringCreateSimple ("Quit");
file = XmStringCreateSimple ("File");
help = XmStringCreateSimple ("Help");

if (dbug == 1)
  printf ("menubar!\n");

menubar = XmVaCreateSimpleMenuBar (main_w, "menubar",
                                   XmVaCASCADEBUTTON, quit, 'Q',
                                   XmVaCASCADEBUTTON, file, 'F',
                                   XmVaCASCADEBUTTON, help, 'H',
                                   NULL);

if (dbug == 1)
  printf ("free file!\n");

XmStringFree (file);

/* Tell the menubar which button is the file button */

if (widget = XtNameToWidget (menubar, "button_2"))
  XtVaSetValues (menubar, XmNmenuHelpWidget, widget, NULL);

/* Quit menu */

XmVaCreateSimplePulldownMenu (menubar, "quit_menu", 0, quit_cb,
                              XmVaPUSHBUTTON, quit, 'Q', NULL, NULL,
                              NULL);
XmStringFree (quit);

/* File menu */

ots = XmStringCreateSimple ("OTS View");
XmVaCreateSimplePulldownMenu (menubar, "file_menu", 1, file_cb,
                              XmVaPUSHBUTTON, ots, "O", NULL, NULL,
                              NULL);

XmStringFree (ots);

/* Help menu */

XmVaCreateSimplePulldownMenu (menubar, "help_menu", 2, help_cb,
                              XmVaPUSHBUTTON, help, 'H', NULL, NULL,
                              NULL);
XmStringFree (help);

XtManageChild (menubar);

/* Create widgets for text entry */

rowcol = XtVaCreateWidget ("rowcol", xmRowColumnWidgetClass,
                           main_w, XmNpacking, XmPACK_COLUMN,
                           XmNnumColumns, XtNumber(text_labels),
                           XmNorientation, XmVERTICAL,
```

```
                                        XmNadjustLast, False,
                                        XmNisAligned, True,
                                        XmNresizeWidth, True,
                                        XmNwidth, 20,
                                        XmNspacing, 5,
                                        XmNentryAlignment, XmALIGNMENT_BEGINNING,
                                        NULL);

        /* Interaction boxes */

        for (i = 0; i < XtNumber(text_labels); i++)
        {
          XtVaCreateManagedWidget (text_labels[i], xmLabelWidgetClass,
                                   rowcol, NULL);
          /*sprintf (buf, "text_%d", i);*/
          w = XtVaCreateManagedWidget (buf, xmTextFieldWidgetClass, rowcol, NULL)

          /* Get appropriate user's input */

          switch (i)
          {
            case 0:  XtAddCallback (w, XmNactivateCallback, ServerHost_cb, i+1);
            case 1:  XtAddCallback (w, XmNactivateCallback, ClientHost_cb, i+1);
            case 2:  XtAddCallback (w, XmNactivateCallback, WindowName_cb, i+1);
            case 3:  XtAddCallback (w, XmNactivateCallback, LowResSample_cb, i+1)
            case 4:  XtAddCallback (w, XmNactivateCallback, LowResFrame_cb, i+1);
          }
          XtAddCallback (w, XmNactivateCallback, XmProcessTraversal,
                         XmTRAVERSE_NEXT_TAB_GROUP);
        }

        /* Wrap up Xt */

        XtManageChild (rowcol);
        XtManageChild (main_w);
        XtRealizeWidget (toplevel);

        XtAppMainLoop(app);
}
```

```
/*-------------------------------------------------*/
/*          Over the Shoulder Application          */
/*                                                 */
/*            The University of Kansas             */
/*       Telecommunications and Information        */
/*                 Sciences Lab                    */
/*                                                 */
/*  Module:        OTS_client.c                    */
/*  Author:        Marcia G. Ramos                 */
/*  Date:          April 14, 1994                  */
/*  Version:       2.0                             */
/*-------------------------------------------------*/

  /* C libraries */

#include <stdio.h>
#include <string.h>

  /* X libraries */

#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>

  /* Motif libraries */

#include <Xm/LabelG.h>
#include <Xm/RowColumn.h>
#include <Xm/Text.h>
#include <Xm/TextF.h>
#include <Xm/MainW.h>
#include <Xm/MessageB.h>
#include <Xm/Label.h>
#include <Xm/FileSB.h>

  /* For debug */

#include <errno.h>

  /* System libraries */

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <netdb.h>
#include <netinet/in.h>
#include <signal.h>
#include <time.h>

  /* Data structures used */

#include "Packets.h"

  /* Window structure */

typedef union   window_id
{
        Window  id;
        char    wid[8];
} window_struct;

  /* Bitmap file for drawing box cursor */

#include "64cursor.xbm"
```

```
  /* Port numbers */

#define TCPServerPort           21000
#define TCPClientPort           22000

/*-------------------------------------------------------------------------*/

        /* Define global variables */

        Display                 *display;       /* X disp */
        window_struct           LowResWindow;   /* Low resolution window */
        window_struct           HighResWindow;  /* High resolution window */
        int                     screen;
        XEvent                  event;      /* Event struct for LowRes images */
        XEvent                  event_high; /* Event struct for HighRes images *,

        unsigned long           foreground;
        unsigned long           background;

        Cursor                  cursor;
        XColor                  color;
        Colormap                colormap;
        XImage                  *ximage;    /* Image struct for LowRes image */
        XImage                  *ximageH;   /* Image struct for HighRes image */
        Visual                  *visual;
        GC                      gc2, gc3;       /* Graphics contexts */
        int                     xdim = 512;     /* Image dimensions */
        int                     ydim = 512;

        Data                    data;               /* Packet transmitted */
        unsigned char           LRImage[512*512];
        unsigned char           HRImage[512*512];

        int                     TimeOut;
        int                     RequestTime = 0;
        void                    handler();
        void                    handle_requests();

        int                     done = 0;
        int                     doneH = 0;

        /* Socket variables */

        struct sockaddr_in      TCPServerAddr;
        struct sockaddr_in      TCPClientAddr;

        int                     TCPSocket;

        /* Request image structure */

        ImageRequest            RequestPacket;

        /* Timing variables */

        long                    start;
        long                    timediff;
        FILE                    *fd;

        /*#########################################################*/
        /*      MAKE THIS EQUAL TO 1 FOR DEBUG                     */
        /*#########################################################*/

        int                     debug = 0;

/*-------------------------------------------------------------------------*/
```

```
void handler ()
{
   /*-----------------------------------------*/
   /* Sets timeout variable when alarm goes off */
   /*-----------------------------------------*/

   /* It will timeout when the recvfrom is sleeping */

   TimeOut = 1;
}

/*-------------------------------------------------------------------------------*/

void handle_requests ()
{
   /*-----------------------------------------*/
   /* Sets timeout variable when alarm goes off */
   /*-----------------------------------------*/

   RequestTime = 1;
}

/*-------------------------------------------------------------------------------*/

void    CreateWindow ()
{

        /*-----------------------------------------------------------------*/
        /* Creates window and does the graphics set up                     */
        /*-----------------------------------------------------------------*/

        char            *DisplayName = NULL;
        char            *WindowName = "Low Resolution Image";
        char            *WindowNameHigh = "High Resolution Image";

        unsigned int            width, height;

        int                     i;

        XWindowAttributes       Wattr;
        XGCValues               values;

        /* Connect to server */

        if (debug == 1)
          printf ("client: Connect to display...\n");
        if ((display = XOpenDisplay (DisplayName)) == NULL)
        {
           fprintf (stderr, "client:Could not open display! \n");
           exit (1);
        }

        /* Create simple windows */

        screen = DefaultScreen (display);
        background = BlackPixel (display, screen);
        foreground = WhitePixel (display, screen);

        /* Low resolution window */

        if (debug == 1)
          printf ("Open LowRes window\n");
        LowResWindow.id = XCreateSimpleWindow (display,
                                               DefaultRootWindow(display),
                                               80, 200, 512, 512, 0,
                                               foreground, background);
```

```
if (debug == 1)
  printf ("LowResWindowID = %#x \n",. LowResWindow.id);

XStoreName (display, LowResWindow.id, WindowName);

/* High resolution window */

if (debug == 1)
  printf ("Open HighRes window\n");
HighResWindow.id = XCreateSimpleWindow (display,
                                        DefaultRootWindow(display),
                                        650, 200, 512, 512, 0,
                                        foreground, background);

if (debug == 1)
  printf ("HighResWindowID = %#x \n", HighResWindow.id);

XStoreName (display, HighResWindow.id, WindowNameHigh);

/* Select event types wanted */

if (debug == 1)
  printf ("Select events...\n");
XSelectInput(display, LowResWindow.id, ExposureMask | ButtonPressMask |
             ButtonReleaseMask | ButtonMotionMask |
             PointerMotionHintMask);

/* Create GC for each window */

if (debug == 1)
  printf ("CreateGCs\n");
gc2 = XCreateGC (display, LowResWindow.id, 0, 0);
gc3 = XCreateGC (display, HighResWindow.id, 0, 0);

/* Adjust background and foreground pixels for each window */

if (debug == 1)
  printf ("Adjust background...\n");
XSetBackground (display, gc2, background);
XSetForeground (display, gc2, foreground);
XSetBackground (display, gc3, background);
XSetForeground (display, gc3, foreground);

/* Create the visual as the default visual */

visual = DefaultVisual (display, screen);

/* Display windows */

if (debug == 1)
  printf ("Display windows...\n");
XMapRaised (display, LowResWindow.id);
XMapRaised (display, HighResWindow.id);

/* Create colormap for 8 bit display */

if (debug == 1)
  printf ("Colormap..\n");
colormap = XCreateColormap (display, RootWindow (display, screen),
                            visual, AllocAll);
if (debug == 1)
  printf ("aftger Colormap..\n");

for (i = 0; i <= 255; i++)
{
```

```
            color.pixel = i;
            XQueryColor (display, DefaultColormap(display, screen), &color);
            color.flags = DoRed | DoGreen | DoBlue;
            XStoreColor (display, colormap, &color);
          }

          /* Scale colors */

          if (debug == 1)
            printf ("Scale colors\n");

          for (i = 50; i <= 250; i++)
          {
            color.pixel= i;
            color.blue = color.green = color.red = ((i - 50)*65535)/200;
            XStoreColor(display, colormap, &color);
          }

          if (debug == 1)
            printf ("Setcolormap\n");
          XSetWindowColormap (display, LowResWindow.id, colormap);
          XSetWindowColormap (display, HighResWindow.id, colormap);
}

/*-------------------------------------------------------------------------*/

Cursor  CreateCursor ()
{
          /*---------------------------------------------------------------*/
          /* Creates a box cursor to be used at the LowRes window          */
          /*---------------------------------------------------------------*/

          Cursor          cursor;
          XColor          fcolor;
          XColor          bcolor;
          Pixmap          mask;
          Pixmap          bitmap;

          /* Get background and foreground colors */

          fcolor.pixel = foreground;
          bcolor.pixel = background;

          XQueryColor (display, colormap, &fcolor);
          XQueryColor (display, colormap, &bcolor);

          /* Create pixmap from bitmap data file */

          mask = bitmap = XCreateBitmapFromData (display, LowResWindow.id, box_bits
                                                 box_width, box_height);

          /* Create the cursor with the pixmap */

          cursor = XCreatePixmapCursor (display, bitmap, mask,
                                        &fcolor, &bcolor, box_x,
                                        box_y);

          if (cursor != (Cursor) None)
          {
            XDefineCursor (display, LowResWindow.id, cursor);
          }

          return (cursor);

}
```

```
/*-------------------------------------------------------------------------*/

void    SendRequest (ImageType, x, y, width, height)
int     ImageType;
short   x, y;
int     width, height;
{
        /*-----------------------------------------------------------------*/
        /*  This function handles the requests for images from the server */
        /*-----------------------------------------------------------------*/

        int     ns;

        /* Send request for LowRes image */

        if (ImageType == 1)
        {
            /* Assign proper values to Request structure */

            RequestPacket->ImageType = htonl(ImageType);
            RequestPacket->x = 0;
            RequestPacket->y = 0;
            RequestPacket->height = 0;
            RequestPacket->width = 0;

            /* Send request structure to the server */

            ns = write (TCPSocket, RequestPacket, sizeof(*RequestPacket));

            /* Check for errors in sending the request */

            if (ns == -1)
            {
              perror ("Client: cannot send request!");
              exit ();
            }
            if (debug == 1)
              printf ("Requested LowRes image...\n");
        }

        /* Send request for HighRes image */

        else
        {
            /* Assign proper values to Request structure */

            RequestPacket->ImageType = htonl(ImageType);
            RequestPacket->x = ntohs(x);
            RequestPacket->y = ntohs(y);
            RequestPacket->height = htonl(height);
            RequestPacket->width = htonl(width);

            /* Send request structure to the server */

            if (debug == 1)
              printf ("Before sending messagfe!\n");
            ns = write (TCPSocket, RequestPacket, sizeof(*RequestPacket));

            /* Check for errors in sendimy the request */

            if (ns == -1)
            {
              perror ("Client: cannot send request!");
              exit ();
            }
            if (debug == 1)
```

```
            {
              printf ("Requested HighRes image...\n");
              printf ("RequestPacket->x = %d\n", x);
              printf ("RequestPacket->y = %d\n", y);
            }
        }
}

/*-------------------------------------------------------------------------*/

void            CreateImageInfo (Picture, type)
unsigned char   Picture[512*512];
int             type;
{
        /*-----------------------------------------------------------------*/
        /*  Creates the picture in x - to be modified (colormap)           */
        /*-----------------------------------------------------------------*/

        int             i;

        /* Create image structure */

        ximage = XCreateImage (display, visual, 8, ZPixmap, 0, Picture, xdim,
                               ydim, 8, 0);

}

/*-------------------------------------------------------------------------*/

void HandleHighResEvents ()
{
        /*-----------------------------------------------------------------*/
        /*  Handles the graphics events such as draw boxes and XPutImage   */
        /*-----------------------------------------------------------------*/

        int             i, j = 0;
        int             first = 0;
        int             width, height;
        int             index = 1;
        int             x, y;
        Window          root, child;
        int             root_x, root_y;
        unsigned int    keys_buttons;
        XPoint          point;
        time_t          tstart;
        XImage          *buffer;
        int             imgx = 258, imgy = 258;
        char            *mesg[2];

        int             FrameInterval = 10;
        int             HighFrames = 0;

        /* Initialize default coordinates */

        point.x = point.y = 0;

        /* Start timer for event loop */

        tstart = time(NULL);

        /* Loop to handle the events for the high-resolution image displays */

        while (doneH == 0)
        {
          /*printf ("wait for an event\n");*/
```

```
/*-------------------------------------------------------------------*/
/* Check number of events in the queue. If there are no events,      */
/* then update the HighRes image. If there is an event in the queue */
/* such as select a new area in the LowRes image, then process the   */
/* event.  Note that if 15 secs have passed, an interrupt will be    */
/* caused and the LowRes image will be updated again.                */
/*-------------------------------------------------------------------*/

if (XEventsQueued (display, QueuedAfterFlush) > 0)
{
      XNextEvent (display, &event_high);
      if (debug == 1)
        printf ("event-received!\n");
      switch (event_high.type)
      {
          j++;
          printf ("loop = %d\n", j);
          case Expose:
              if (debug == 1)
              {
                printf ("Expose\n");
                printf ("do-nothing!\n");
              }
              break;

          /* Mouse button is pressed, get coordinates */

          case ButtonPress:
              if (debug == 1)
                printf ("ButtonPress\n");
              point.x = event_high.xbutton.x;
              point.y = event_high.xbutton.y;
              /* Send request for HighRes image */
              if (debug == 1)
                printf ("Send HIGHRES requests!\n");
              start = time(NULL);
              if (debug == 1)
                printf ("Before send-request!\n");
              SendRequest (2, point.x, point.y, 64, 64);
              if (debug == 1)
                printf ("After send-request!\n");
              timediff = time(NULL) - start;
              start = time(NULL);
              timediff = time(NULL) - start;
              break;

          /* Mouse button is released, display HighRes image */

          case ButtonRelease:
              if (debug == 1)
                printf ("ButtonRelease!\n");
              break;

          case MotionNotify:
              if (debug == 1)
                printf ("MotionNotify\n");
              break;

          /* Receive current HighRes image from the server */

          default:
              break;
      }

   }
}
```

```
        if (debug == 1)
          printf ("out-of-loop\n");
}

/*-------------------------------------------------------------------------*/

void HandleEvents ()
{
       /*-------------------------------------------------------------------*/
       /*  Handles the events for the LowRes image and for the HighRes    */
       /*-------------------------------------------------------------------*/

        /* Loop to handle the events for the high-resolution image displays */

        while (done == 0)
        {
          XNextEvent (display, &event);
          switch (event.type)
          {
            /* Display image */

            case Expose:
              if (debug == 1)
                printf ("EXPOSE!!!!!!!!!!!!!\n");
              start = time (NULL);
              SendRequest (1, 0, 0, 0, 0);
              timediff = time(NULL) - start;
              CreateImageInfo (LRImage, 1);
              start = time (NULL);
              timediff = time(NULL) - start;
              if (debug == 1)
                printf ("DISPLAYED IMAGE\n");

              /* Handle events for the HighRes image */

              HandleHighResEvents ();
              if (debug == 1)
                printf ("OUT*****\n");
              doneH = 0;
              done = 1;
              break;

          }
        }
}

/*-------------------------------------------------------------------------*/

void    CreateTCPServerSocket (ServerHost)
char    *ServerHost;
{

        /*-------------------------------------------------------------------*/
        /*  This function creates the TCP socket that is used for the      */
        /*  server.                                                        */
        /*-------------------------------------------------------------------*/

        struct  hostent         *hp;
        char                    HostName[64];

        /* Clear and set name/address structure */

        bzero ((char *)&TCPServerAddr, sizeof(TCPServerAddr));

        /* Convert port number to network byte order */
```

```
        TCPServerAddr.sin_port = htons (TCPServerPort);

        /* Set family to internet */

        TCPServerAddr.sin_family = AF_INET;

        /* Get server address to connect to */

        /*sleep (5);*/

        if (debug == 1)
          printf ("ServerHost = %s\n", ServerHost);

        hp = gethostbyname (ServerHost);
        if (hp == 0)
        {
          printf ("Client Error: could not obtain the address of \n");
          exit ();
        }

        /* Copy address to socket structure */

        bcopy (hp->h_addr_list[0], (caddr_t)&TCPServerAddr.sin_addr,
               hp->h_length);

        /* Connect to the server */

        sleep (5);
        if (connect (TCPSocket, (struct sockaddr *)&TCPServerAddr,
                     sizeof(TCPServerAddr)) < 0)
        {
          perror ("Client Error: trying to connect");
          exit ();
        }
}

/*-------------------------------------------------------------------------*/

void    CreateTCPClientSocket ()
{
        /*-----------------------------------------------------------------*/
        /*  This function creates the TCP socket that is used for          */
        /*  sending the requests to the server.                            */
        /*-----------------------------------------------------------------*/

        /* Clear and set name/address structure */

        bzero ((char *)&TCPClientAddr, sizeof(TCPClientAddr));

        /* Convert port number to network byte order */

        TCPClientAddr.sin_port = htons (TCPClientPort);

        /* Allow connections from any server */

        TCPClientAddr.sin_addr.s_addr = htonl (INADDR_ANY);

        /* Set socket family to internet */

        TCPClientAddr.sin_family = AF_INET;

        /* Create a socket */

        if ((TCPSocket = socket (AF_INET, SOCK_STREAM, 0)) < 0)
        {
```

```
            perror ("Client: TCP socket failed!");
            exit ();
          }

          /* Bind socket to local address */

          if (bind (TCPSocket, (struct sockaddr *)&TCPClientAddr,
                    sizeof(TCPClientAddr)) < 0)
          {
            perror ("Client: bind failed!");
            exit();
          }
}

/*-------------------------------------------------------------------------*/

void      ClientSide (ServerHost)
char      *ServerHost;
{
          /*---------------------------------------------------------------*/
          /*  Initializes sockets and handles requests - main routine      */
          /*---------------------------------------------------------------*/

          int               j, ns;
          char              HostName[64];

          if (debug == 1)
            printf ("got here!\n");

          /* Check if host is proper host */

          if (gethostname(HostName, sizeof(HostName)-1) < 0)
          {
            perror("Gethostname problem: check the Host name!\n");
            exit();
          }

          if (debug == 1)
            printf("\nYour Host is : %s \n", HostName);

          /* Create TCP sockets */

          if (debug == 1)
            printf ("Create sockets!\n");
          CreateTCPClientSocket ();
          CreateTCPServerSocket (ServerHost);

          /* Open display */

          if (debug == 1)
            printf ("Before window...\n");
          CreateWindow ();
          if (debug == 1)
            printf ("After window...\n");

           /* Create box cursor for LowRes image */

          cursor = CreateCursor ();

          /* Allocate memory for the image structure */

          if (debug == 1)
            printf ("Allocates memory...\n");
          RequestPacket = (ImageRequest) malloc(sizeof(struct Request));
          if (debug == 1)
            printf ("After allocate\n");
```

```
        /* Send lowreswindow id */

        if (debug == 1)
          printf ("send window id!\n");
        start = time(NULL);
        ns = write (TCPSocket, LowResWindow.wid, sizeof(LowResWindow.wid));
        if (debug == 1)
          printf ("write!\n");
        ns = write (TCPSocket, HighResWindow.wid, sizeof(HighResWindow.wid));
        timediff = time(NULL) - start;
        if (debug == 1)
          printf ("Before loop!\n");

        /* Loop forever */

        j = 1;
        for (;;)
        {
          /* Send an expose event to event queue */

          XSendEvent (display, LowResWindow.id, False, Expose, &event);
          if (debug == 1)
            printf ("EVENT SENT\n");

          /* Process events - if mouse pressed, request and display HighRes */

          HandleEvents ();
          done = 0;
          RequestTime = 0;
          if (debug == 1)
            printf ("OUT OF EVENTS LOOP\n");
          j++;
          if (j == 3)
            fclose (fd);
        }

        /* Closes the socket descriptors */

        close (TCPSocket);
}

/*-------------------------------------------------------------------------*/
```

```
/*-------------------------------------------------*/
/*        Over the Shoulder Application            */
/*                                                 */
/*           The UniversiTY Of Kansas              */
/*      Telecommunications and Information         */
/*                Sciences Lab                     */
/*                                                 */
/*  Module:       img_server                       */
/*  Author:       Marcia G. Ramos                  */
/*  Date:         March 07, 1994                   */
/*  Version:      2.0                              */
/*-------------------------------------------------*/

  /* C libraries */

#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>

  /* X libraries */

#include <X11/Xlib.h>
#include <X11/Xutil.h>

  /* System libraries */

#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <sys/param.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <limits.h>
#include <time.h>

  /* Define port numbers for TCP service */

#define TCPServerPort           21000
#define TCPClientPort           22000

  /* Data structures used */

#include "Packets.h"

  /* Window structure */

typedef union   window_id
{
        Window  id;
        char    wid[8];
} window_struct;

/*-------------------------------------------------------------------------*/

        /* Display devices */

        FILE *fp;

        Display                 *ClientDisplay;
        Display                 *AppDisplay;
```

```
/* Window information */

GC                          gc1, gc2;
Visual                      *visual;
int                         screen;
unsigned long               foreground;
unsigned long               background;

/* Window IDs */

Window                      wid;
window_struct               Appwindow;
window_struct               lowreswindow;
window_struct               highreswindow;

/* Image structures */

XImage                      *ximage;
XImage                      *LowResImage;
XImage                      *HighResImage;

/* Size of the TerraVision image */

unsigned int                width;
unsigned int                height;

/* Packet with request from client */

ImageRequest                RequestPacket;

/* TCP addresses and socket structures */

struct  sockaddr_in         TCPServerAddr;
struct  sockaddr_in         TCPClientAddr;

/* TCP sockets */

int                         TCPServerSocket;
int                         TCPClientSocket;

/* Timing variables for stats */

long                        start, timediff;
FILE                        *fd;

/*#################################################################*/
/*      MAKE THIS EQUAL TO 1 FOR DEBUG                             */
/*#################################################################*/

int                         sdebug = 0;

/* Sample rate for the LowRes image */

int                         SampleRate;

/* HighRes frames */

time_t                      HighResFramesTimer;
int                         FrameInterval = 10;
int                         HighResFrames = 0;
float                       HighResFramesTotal = 0;
int                         HighResFramesMult = 10;
int                         HighResFramesCount = 0;
float                       FrameRate;

char                        *ServerHost;
```

```
        char                    *ClientHost;
        char                    *WindowName;
        int                     LowResSampleRate;
        int                     LowResFrameRate;

  /* Get window ID info */

#include "xwininfo.h"

/*-----------------------------------------------------------------------*/

int     CreateTCPSocket ()
{
        /*--------------------------------------------------------------*/
        /*  This function creates the TCP socket that is used for       */
        /*  receiving the image requests from the client.               */
        /*--------------------------------------------------------------*/

        struct  sockaddr_in     TCPServerAddr;
        int                     TCPServerSocket;

        /* Clear and set name/address structure */

        bzero ((char *)&TCPServerAddr, sizeof(TCPServerAddr));

        /* Convert port number to network byte order */

        TCPServerAddr.sin_port = htons (TCPServerPort);

        /* Set family to internet */

        TCPServerAddr.sin_family = AF_INET;

        /* Allow connections from all clients */

        TCPServerAddr.sin_addr.s_addr = htonl (INADDR_ANY);

        /* Create TCP socket */

        if ((TCPServerSocket = socket (AF_INET, SOCK_STREAM, 0)) < 0)
        {
          perror ("Server: can't open TCP socket!");
          exit(1);
        }

        /* Bind socket to port */

        if (bind (TCPServerSocket, &TCPServerAddr,
                  sizeof(TCPServerAddr)) < 0)
        {
          perror ("Server: can't bind TCP socket to local address!");
          exit(1);
        }

        /* Listen to the socket for a connection */

        if (listen (TCPServerSocket, 1) < 0)
        {
          perror ("Server: listen failed!");
          exit(1);
        }

        /* Return socket */

        return TCPServerSocket;
```

```
}
/*-------------------------------------------------------------------------*/

void    CreateImageStructures ()
{
        /*---------------------------------------------------------------*/
        /*   This function creates the image strutures for the LowRes    */
        /*   and HighRes images                                          */
        /*---------------------------------------------------------------*/

        unsigned char   Image[512*512];
        int             i;

        /* Initialize buffer to 0 */

        for (i = 0; i < 512*512; i++)
          Image[i] = 1;

        if (sdebug == 1)
          printf ("Create images!\n");

        /* Define and initialize image structures */

        LowResImage = XCreateImage (AppDisplay, visual, 8, ZPixmap, 0, Image,
                                    512, 512, 8, 0);

        HighResImage = XCreateImage (AppDisplay, visual, 8, ZPixmap, 0, Image,
                                     512, 512, 8, 0);

}
/*-------------------------------------------------------------------------*/

void    GetServerImage ()
{
        /*---------------------------------------------------------------*/
        /*  Gets the whole image from TerraVision                        */
        /*---------------------------------------------------------------*/

        int             i;

        /* Grab image from TerraVision */

        if (sdebug == 1)
          fprintf (fp, "Before getimage!\n");

        if ((ximage = XGetImage (AppDisplay, Appwindow.id, 0, 0, width, height,
                                 ~0, ZPixmap)) == NULL)
        {
          fprintf (fp, "Error in XGetImage!\n");
        }

        if (sdebug == 1)
          fprintf (fp, "Image received!\n");
}
/*-------------------------------------------------------------------------*/

void    GetHighResImage ()
{
        /*---------------------------------------------------------------*/
        /*  Gets the HighRes image from TerraVision                      */
        /*---------------------------------------------------------------*/

        int             i;
```

```
int             x, y;
int             xcoord, ycoord = 0;
int             index;
int             sizex, sizey;
unsigned long   pixelvalue;


sizex = sizey = 512;

/* Get the HighRes image according to the selected area */

if ((SampleRate*RequestPacket->x < width/SampleRate) &&
    (SampleRate*RequestPacket->y < height/SampleRate))
{
  x = SampleRate*RequestPacket->x;
  y = SampleRate*RequestPacket->y;
  if (x < (width - 512))
    x = SampleRate*RequestPacket->x;
  else
    x = width - 512;
  if (y < (height - 512))
    y = SampleRate*RequestPacket->y;
  else
    y = height - 512;
}
if ((SampleRate*RequestPacket->x > width/SampleRate) &&
    (SampleRate*RequestPacket->y < height/SampleRate))
{
  x = width - 512;
  y = SampleRate*RequestPacket->y;
  if (y < (height - 512))
    y = SampleRate*RequestPacket->y;
  else
    y = height - 512;
  printf ("x = %d\n", x);
  printf ("y = %d\n", y);
}
if ((SampleRate*RequestPacket->x > width/SampleRate) &&
    (SampleRate*RequestPacket->y > height/SampleRate))
{
  x = width - 512;
  y = height - 512;
}
if ((SampleRate*RequestPacket->x < width/SampleRate) &&
    (SampleRate*RequestPacket->y > height/SampleRate))
{
  x = SampleRate*RequestPacket->x;
  y = height - 512;
  if (x < (width - 512))
    x = SampleRate*RequestPacket->x;
  else
    x = width - 512;
}

if (sdebug == 1)
{
  fprintf (fp, "Before gethighresimage!\n");
  fprintf (fp, "x = %d\n", x);
  fprintf (fp, "y = %d\n", y);
}

/* Grab HighRes image from TerraVision */

if (sdebug == 1)
{
  fprintf (fp, "sizex = %d\n", sizex);
```

```
        fprintf (fp, "sizey = %d\n", sizey);
      }
      if ((HighResImage = XGetImage (AppDisplay, Appwindow.id, x, y,
                                     sizex, sizey,
                                     ~0, ZPixmap)) == NULL)
      {
        fprintf (fp, "Error in XGetImage!\n");
      }

      if (sdebug == 1)
        fprintf (fp, "BEFORE PUT HIGH RES!\n");

      /* Send HighRes image to client */

      start = time(NULL);
      if (sdebug == 1)
        fprintf (fp, "Put High!\n");
      XPutImage (ClientDisplay, highreswindow.id, gc2, HighResImage,
                 0, 0, 0, 0, sizex, sizey);
      if (sdebug == 1)
      {
        fprintf (fp, "AFTER PUT!\n");
        fprintf (fp, "Image received!\n");
        fprintf (fp, "size = %d \n", sizeof(ximage));
      }
      fclose (fp);
}

/*-------------------------------------------------------------------------*/

void    SampleLowResImage ()
{
      /*-----------------------------------------------------------------*/
      /*   This function samples the server image into a LowRes          */
      /*   image.                                                        */
      /*-----------------------------------------------------------------*/

      unsigned long   pixelvalue;
      int             x, y;
      int             xcoord, ycoord;
      int             index = 0;

      /* Initialize LowRes image coordinates */

      xcoord = ycoord = 0;

      /* Sample LowRes getting pixel  according to the SampleRate selected */

      if (sdebug == 1)
      {
        fprintf (fp, "SampleRate = %d\n", SampleRate);
        fprintf (fp, "width = %d\n", width);
        fprintf (fp, "height = %d\n", height);
      }

      for (y = 0; y < height; y=y + SampleRate)
      {
        for (x = 0; x < width; x=x + SampleRate)
        {
           pixelvalue = XGetPixel (ximage, x, y);
           XPutPixel (LowResImage, xcoord, ycoord, pixelvalue);
           xcoord++;
        }
        xcoord = 0;
        ycoord++;
        /*printf ("pixel = %ld\n", pixelvalue);*/
```

```
        }

        /* Send sampled image to client display */

        if (sdebug == 1)
          fprintf (fp, "BEFORE PUTLow!\n");
        XPutImage (ClientDisplay, lowreswindow.id, gc1, LowResImage,
                   0, 0, 0, 0, width/SampleRate, height/SampleRate);
        if (sdebug == 1)
          fprintf (fp, "AFTER PUT!\n");
}

/*-------------------------------------------------------------------------*/

void    SendImage       ()
{
        /*-----------------------------------------------------------------*/
        /*  This function sends the appropriate image to the client by     */
        /*  calling the appropriate functions defined earlier.             */
        /*-----------------------------------------------------------------*/

        /* Get image from TerraVision and sample it */

        if (ntohl(RequestPacket->ImageType) == 1)
        {
          GetServerImage ();
          if (sdebug == 1)
            printf ("sampling...\n");
          if ((width > 512) || (height > 512))
            SampleLowResImage ();
          if (sdebug == 1)
            printf ("sampling done!\n");
        }
        else
        {
          GetHighResImage ();
          if (sdebug == 1)
            printf ("gethighres...\n");
        }
}

/*-------------------------------------------------------------------------*/

Window  byteorder (window)
Window  window;
{
        /*-----------------------------------------------------------------*/
        /*  This function converts the byte order from the Alpha to the    */
        /*  onyx.                                                          */
        /*-----------------------------------------------------------------*/

        return (((((window & 0xff00) >> 8) & 0x00ff)
                  | ((window & 0x00ff) << 8)) << 16)
                | ((window & 0xffff0000) >> 24);
}

/*-------------------------------------------------------------------------*/

void    ServerSide (argc, argv)
int     argc;
char    *argv[];
{

        /*-----------------------------------------------------------------*/
        /*  This function is the main routine in the server side. It       */
        /*  creates the TCP socket for receiving image requests. The       */
```

```
        /*  socket waits for an image request and if no requests are    */
        /*  present at the queue, the server will just send a high      */
        /*  resolution image with the previous coordinates.If requests  */
        /*  are present in the queue, the server will process all of    */
        /*  them in the order they appear in the queue.                 */
        /*--------------------------------------------------------------*/

/*------------------------------------------------------------------------*/

        /* Displays names for server and client */

        char            DisplayName[50];
        char            TerraDisplay[50];

        /* Address size */

        int             TCPAddrLength;

        /* Control variables */

        int             Receive;
        int             First = 1;
        int             Block = 0;

        /* Previous coordinates requested by user */

        int             prev_x = 0;
        int             prev_y = 0;

        int             i;

        /* Timer for the LowRes updates */

        time_t          tstart;

/*------------------------------------------------------------------------*/

        /* Get LowRes sample rate from the command line */

        SampleRate = LowResSampleRate;

        if (sdebug == 1)
          fprintf (fp, "SampleRate = %d\n", SampleRate);

        if (sdebug == 1)
        {
          printf ("width = %d\n", width);
          printf ("height = %d\n", height);
        }

        /* Allocate memory for the request structure */

        RequestPacket = (ImageRequest) malloc(sizeof(struct Request));

        /* Create the TCP socket for receiving requests */

        TCPServerSocket = CreateTCPSocket ();
        TCPAddrLength = sizeof (TCPClientAddr);

        /* Accept client connection */

        if (sdebug == 1)
          fprintf (fp, "Before accept client!\n");
        TCPClientSocket = accept (TCPServerSocket, &TCPClientAddr,
                                  &TCPAddrLength);
        if (sdebug == 1)
```

```
  fprintf (fp, "After accept client!\n");
close (TCPServerSocket);

/* Machine to provide the image */

DisplayName[0] = '\0';
strcat (DisplayName, ServerHost);
strcat (DisplayName, ":0.0");
printf ("DisplayName = %s\n", DisplayName);
if (sdebug == 1)
  fprintf (fp, "Before open display!\n");
if ((AppDisplay = XOpenDisplay (DisplayName)) == NULL)
{
   fprintf (stderr, "server:Could not open display! \n");
   exit (1);
}

/* Get TerraVision's window info using the xwininfo function */

if (sdebug == 1)
  fprintf (fp, "Name = %s\n", WindowName);

if (WindowName == "Out_The_Window_View")
  WindowName = "Out The Window View";
if (WindowName == "Overhead_View")
  WindowName = "Overhead View";
xwininfo (argc, argv, WindowName);

/* Client display */

TerraDisplay[0] = '\0';
strcat(TerraDisplay, ClientHost);
strcat (TerraDisplay, ":0.0");
printf ("TerraDisplay = %s\n", TerraDisplay);
if ((ClientDisplay = XOpenDisplay (TerraDisplay)) == NULL)
{
  fprintf (stderr, "server:Could not get other display! \n");
  exit (1);
}

if (sdebug == 1)
  fprintf (fp, "before GC!\n");

/* Define GC */

screen = DefaultScreen (ClientDisplay);
visual = DefaultVisual (ClientDisplay, screen);
gc1 = XCreateGC (ClientDisplay, RootWindow(ClientDisplay, screen), 0, NUI
gc2 = XCreateGC (ClientDisplay, RootWindow(ClientDisplay, screen), 0, NUI

XSetBackground (ClientDisplay, gc1, background);
XSetForeground (ClientDisplay, gc1, foreground);
XSetBackground (ClientDisplay, gc2, background);
XSetForeground (ClientDisplay, gc2, foreground);
if (sdebug == 1)
  fprintf (fp, "After GC!\n");

Appwindow.id = wid;
if (sdebug == 1)
  fprintf (fp, "Appwindow.id = %#x\n", Appwindow.id);

/* Initialize image structures */

CreateImageStructures ();

/* Get LowRes and HighRes window IDs */
```

```
if (sdebug == 1)
  fprintf (fp, "Get window.id \n");

Receive = read (TCPClientSocket, lowreswindow.wid, sizeof(lowreswindow.wi
Receive = read (TCPClientSocket, highreswindow.wid, sizeof(highreswindow.
if (sdebug == 1)
{
  fprintf (fp, "LowResWindowID = %#x \n", lowreswindow.id);
  fprintf (fp, "HighResWindowID = %#x \n", highreswindow.id);
}

/* Change the byte order if necessary */

lowreswindow.id = byteorder (lowreswindow.id);
highreswindow.id = byteorder (highreswindow.id);

if (sdebug == 1)
{
  fprintf (fp, "LowResWindowID = %#x \n", lowreswindow.id);
  fprintf (fp, "HighResWindowID = %#x\n", highreswindow.id);
}

/* Set socket to be non-blocking */

fcntl (TCPClientSocket, F_SETFL, FNDELAY);

/* Initialize timer for LowResFrames */

tstart = time(NULL);

/* Initialize timer for HighResFrames */

HighResFramesTimer = time(NULL);

while (1)
{
  /* Gets request if there is one */

  if (sdebug == 1)
    fprintf (fp, "Before read!\n");

  Receive = read (TCPClientSocket, RequestPacket,
                  sizeof(*RequestPacket));

  /* If there is no request in the queue, send high resolution image */

  if (Receive == -1)
  {
    /* Checks if accept failed */

    if (errno != EWOULDBLOCK)
    {
      perror ("Server: error in accepting requests!");
      exit(1);
    }

    /* There are no requests, send image */

    else
    {
      if (First == 1)
      {
        if (sdebug == 1)
          printf ("Waiting for first request...\n");
        First = 0;
```

```
      }
      if (Block != 0)
      {
        if (sdebug == 1)
          printf ("SEND_HIGH_RES!\n");

        /* Get previous coordinates of the HighRes image */

        if (Receive == -1)
        {
          RequestPacket->ImageType = htonl(2);
          RequestPacket->x = prev_x;
          RequestPacket->y = prev_y;
        }

        /* Send previous coordinates to client */

        if (sdebug == 1)
        {
          printf ("HIGHRES!\n");
          printf ("LowResFrameRate = %d\n", LowResFrameRate);
        }
        if ((time(NULL) - tstart) >= LowResFrameRate)
        {
          RequestPacket->ImageType = 1;
          SendImage ();
          tstart = time(NULL);
        }
        else
          SendImage ();
        if (sdebug == 1)
          printf ("SendImage!\n");
      }
    }
  }

  /* Process requests in the queue */

  else
  {
    Block = 1;
    if (sdebug == 1)
    {
      printf ("RequestIssued!\n");
      printf ("TCPSocket = %d\n", TCPClientSocket);
    }

    /* Gets data from TCP socket */

    if (sdebug == 1)
    {
      printf ("Request->ImageType = %d\n", ntohl(RequestPacket->ImageType
      printf ("Request->x = %d\n", htons(RequestPacket->x));
      printf ("Request->y = %d\n", htons(RequestPacket->y));
      printf ("Request->width = %d\n", ntohl(RequestPacket->width));
      printf ("Request->height = %d\n", ntohl(RequestPacket->height));
    }

    /* Update previous coordinates for HighRes image */

    if (ntohl(RequestPacket->ImageType) == 2)
    {
      prev_x = htons(RequestPacket->x);
      prev_y = htons(RequestPacket->y);
    }
```

```
            /* Sends appropriate image */

            SendImage ();

          }

        }
}

/*-------------------------------------------------------------------------*/

main    (argc, argv)
int     argc;
char    *argv[];
{

        fp = fopen ("data.dat", "w");

        if (sdebug == 1)
          fprintf (fp, "Server!\n");

        ServerHost = argv[1];
        ClientHost = argv[2];
        WindowName = argv[3];
        LowResSampleRate = atoi (argv[4]);
        LowResFrameRate = atoi (argv[5]);

        if (sdebug == 1)
        {
          fprintf (fp, "This is the server!\n");
          fprintf (fp, "ServerHost = %s\n", ServerHost);
          fprintf (fp, "ClientHost = %s\n", ClientHost);
          fprintf (fp, "WindowName = %s\n", WindowName);
          fprintf (fp, "Sample = %d\n", LowResSampleRate);
        }

        ServerSide (argc, argv);
}
```

TISL Technical Report 9770-14; "An Over-The-Shoulder Implementation"
Appendix D

```
cc -O -c OTS_client.c
cc -O -c OTS.c
cc -O -o OTSv2 OTS_client.o OTS.o -lXm -lXt -lX11 -lPW
```

```
cc -O dsimple.o -c dsimple.c
cc -O OTS_server.o -c OTS_server.c
cc -o OTS_server dsimple.o OTS_server.o -lX11 -lXt -lXmu -lXext
```