

CS-4202
Computer Graphics

Part 5

Pixel Data
to
Image Display Tools

Notes and Programs by:

Dr. Michael J. Zyda

Pixel Data 3
Higher Level Image Support 6
NPSimage.h 7
NPSimage.C 9
How to Use the NPSimage Methods 16
Available Image Display Tools 17
Also Available Image Display Tools (Tools programmed in IRIS GL) 18

Pixel Data

-- So far in this class we have only been concerned with geometric figures.

-- Each time we turned geometric figures into pixels but we tried to avoid knowing/playing with pixel data...

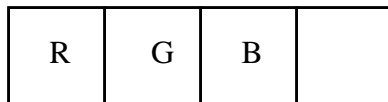
-- Sometimes we need to know what our pixel data looks like...

-- So I'm going to show you how to read/write pixel data...

Pixel Formats

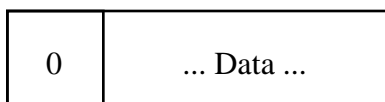
(1) RGB Data

-- read/written as 32 bit integers.



(2) Z-Buffer Data

-- 24 bits of valid data, from bits 0 to 23.



Pixel Data continued

Pixel Reading

```
void glReadPixels(GLint x, GLint y, GLsizei width, GLsizei height,  
                 GLenum format, GLenum type, GLvoid *pixels);
```

-- Reads pixel data from the framebuffer rectangle whose lower left corner is at (x, y) and whose dimensions are *width* and *height*, and stores it in the array pointed to by *pixels*.

-- *format* indicates the kind of pixel data elements that are read (GL_RGB, GL_RGBA, GL_RED, GL_GREEN, GL_BLUE, GL_ALPHA, GL_DEPTH_COMPONENT, ...).

-- *type* indicates the data type for each element (GL_UNSIGNED_BYTE, GL_BYTE, GL_BITMAP, GL_UNSIGNED_SHORT, GL_SHORT, GL_UNSIGNED_INT, GL_INT, GL_FLOAT).

Pixel Drawing

```
void glDrawPixels(GLsizei width, GLsizei height,  
                 GLenum format, GLenum type, const GLvoid *pixels);
```

-- Draw a rectangle of pixel data with dimensions *width* and *height*. The pixel rectangle is drawn with its lower left corner at the current raster position (use glRasterPos2i(ix, iy) ...).

-- The *format* and *type* parameters have the same meaning as for glReadPixels().

-- The array pointed at by *pixels* contains the pixel data to be drawn.

Copying Pixels

`glCopyPixels(GLint x, GLint y, GLsizei width, GLsizei height, GLenum type);`

-- Copies pixel data from the framebuffer rectangle whose lower left corner is at (x, y) and whose dimensions are *width* and *height*. The data is copied to a new position whose lower left corner is given by the current raster position.

-- type is either `GL_COLOR`, `GL_STENCIL`, or `GL_DEPTH`.

Magnifying or Reducing an Image

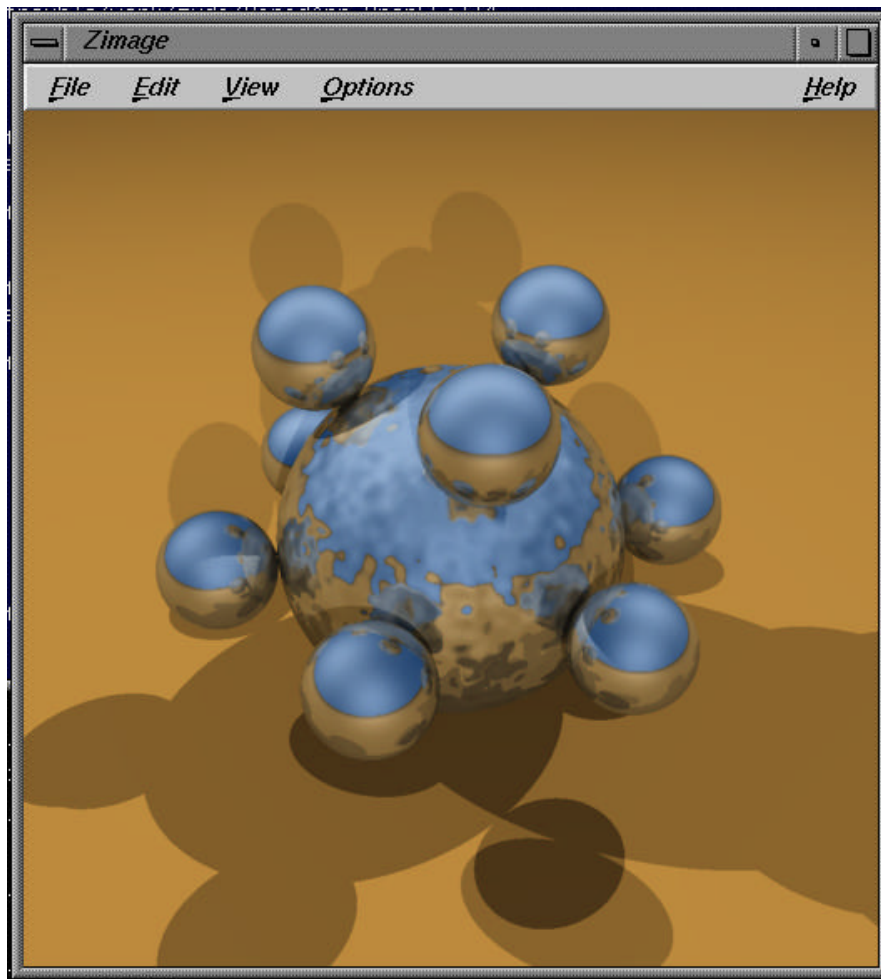
`glPixelZoom(GLfloat xzoom, GLfloat yzoom);`

-- Sets the magnification or reduction factors for pixel write operations.

-- By default, `xzoom` and `yzoom` are 1.0.

-- Fractional magnification or reduction factors are allowed, as are negative factors.

RapidApp-OpenGL/Zimage



-- I've written a support library for somewhat higher level image support...

-- This library reads/writes SGI formatted images.

-- A sample program using this library is program Zimage.

Zimage Files

Form.C - generated and edited.
Form.h - generated and edited.
Form.o
FormUI.C
FormUI.h
FormUI.o
Makedepend
Makefile - edited
NPSimage.C - image support library.
NPSimage.h - image support library.
NPSimage.o
VkwindowMainWindow.C
VkwindowMainWindow.h
VkwindowMainWindow.o
Zimage - resources
Zimage.cvdb/
b2.rgb
b4.rgb
balls.rgb
balls2.rgb
cvstatic.fileset
desktop.ftr
icon.fti
image.h
images/
main.C
main.o
makeRasterFont.C
makeRasterFont.h
makeRasterFont.o
note.to.myself.t
unimplemented.C
unimplemented.o
zimage* executable
zimage.idb
zimage.spec
zimage.uil
zimage.uil~

note.to.myself.t

Program Draw Notes: 11 Feb 96

1. I attempted to print out the filename generate by the save() callback in Form.C. I discovered that the saveas() callback had a filename but that the save() callback did not. The save() callback mentioned that there was a filename parameter but it wasn't there!

2. So I modified
 - > the save() callback in Form.C so it had a filename parameter.
 - > the save() definition in Form.h
 - > I then got a compile error in VkMainWindow.C
 - in VkMainWindow::save() I had to copy the code from
VkMainWindow::saveas() so that there was a filename passed to Form::save().
 - > I looked everywhere for a higher level definition I could change so that I did not have to hack VkMainWindow.C. I could not find one. This means the code generator that generates VkMainWindow.C is broken. This is an SGI problem.

Form.h

```
////////////////////////////////////
//
// Header file for Form
//
// This file is generated by RapidApp
//
// This class is derived from FormUI which
// implements the user interface created in
// the interface builder. This class contains virtual
// functions that are called from the user interface.
//
// When you modify this header file, limit your changes to adding
// members below the "//--- End generated code section" markers
//
// This will allow the builder to integrate changes more easily
//
// This class is a ViewKit user interface "component".
// For more information on how components are used, see the
// "ViewKit Programmers' Manual", and the RapidApp
// User's Guide.
////////////////////////////////////
#ifndef FORM_H
#define FORM_H
#include "FormUI.h"

#include <Vk/VkWindow.h>
#include <Vk/VkMenuBar.h>
#include <Vk/VkSubMenu.h>
//---- End generated headers

// Here are some define constants I need for the program.

// Here are some more includes I added.

#include <Xm/Xm.h> // Get the Motif stuff

#include <X11/StringDefs.h>
#include <X11/keysym.h>

#include <GL/gl.h> // Get GL required includes.
#include <GL/glu.h>
#include <GL/glx.h>

#include <iostream.h> // Get the C++ IO stuff.
#include <stdlib.h> // Get the exit() function.

#include "makeRasterFont.h" // Get the font support routines.

#include "NPSImage.h" // Get the NPSImage class def.
```

```

//---- Form class declaration

class Form : public FormUI
{

public:

    Form(const char *, Widget);
    Form(const char *);
    ~Form();
    const char * className();
    virtual void setParent(VkWindow *);
    virtual void copy();
    virtual void cut();
    virtual void newFile();
    virtual void openFile(const char *);
    virtual void paste();
    virtual void print(const char *);
    virtual void save(const char *);
    virtual void resetCB(Widget, XtPointer);
    virtual void saveas(const char *);

    //---- End generated code section

protected:

    // These functions will be called as a result of callbacks
    // registered in FormUI

    virtual void exposeCB ( Widget, XtPointer );
    virtual void initCB ( Widget, XtPointer );
    virtual void inputCB ( Widget, XtPointer );
    virtual void resizeCB ( Widget, XtPointer );

    VkWindow * _parent;
    //---- End generated code section

    // The following items were global but were placed here by Zyda.

    GLXContext glx_context;    // A GL context (see initCB).

    GLuint fontHandle;        // Base of font display lists.

    Display *global_display;  // Global display.

    Window global_window;     // Global window.

    int window_x_size, window_y_size; // Size of the window in pixels.

```

```
NPSimage globalImage;      // This is the global image we are displaying.

private:

void draw_the_scene ( ); // draw the picture!

// A function to change the window size ...
void changeWindowSize( Widget glwindow, int width, int height);

};
#endif
```

Form.C

```
////////////////////////////////////
//
// Source file for Form
//
// This file is generated by RapidApp
//
// This class is derived from FormUI which
// implements the user interface created in
// the interface builder. This class contains virtual
// functions that are called from the user interface.
//
// When you modify this source, limit your changes to
// modifying the empty virtual functions. You can also add
// new functions below the "//--- End generated code section" markers
//
// This will allow the builder to integrate changes more easily
//
// This class is a ViewKit user interface "component".
// For more information on how components are used, see the
// "ViewKit Programmers' Manual", and the RapidApp
// User's Guide.
////////////////////////////////////

#include "Form.h"
#include <Vk/VkEZ.h>
#include <GL/GLwMDrawA.h>
#include <Xm/Form.h>
#include <Vk/VkResource.h>
#include <Vk/VkWindow.h>
#include <Vk/VkMenuBar.h>
#include <Vk/VkSubMenu.h>

extern void VkUnimplemented(Widget, const char *);

////////////////////////////////////
/
// The following non-container widgets are created by FormUI and are
// available as protected data members inherited by this class
//
// GLwMDrawingArea _glwidget
//
////////////////////////////////////
/

//---- End generated headers
```

```

//---- Form Constructor

Form::Form(const char *name, Widget parent) :
    FormUI(name, parent)
{
    // This constructor is called after the component's interface has been built.

    //--- Add application code here:

    // Let's display a default image here ...
    globalImage.read_from("balls.rgb");

    // Set the window to the size of this image ...
    changeWindowSize( _glwidget, globalImage.size[0], globalImage.size[1]);
} // End Constructor

Form::Form(const char *name) :
    FormUI(name)
{
    // This constructor calls FormUI(name)
    // which does not create any widgets. Usually, this
    // constructor is not used

    //--- Add application code here:
} // End Constructor

Form::~Form()
{
    // The base class destructors are responsible for
    // destroying all widgets and objects used in this component.
    // Only additional items created directly in this class
    // need to be freed here.

    //--- Add application destructor code here:
}

const char * Form::className() // classname
{
    return ("Form");
} // End className()

```

```
void Form::exposeCB ( Widget w, XtPointer callData )
{
    XmAnyCallbackStruct *cbs = (XmAnyCallbackStruct*) callData;

    //--- Comment out the following line when Form::exposeCB is implemented:

    // ::VkUnimplemented ( w, "Form::exposeCB" );

    //--- Add application code for Form::exposeCB here:

    GLwDrawingAreaCallbackStruct *glptr = (GLwDrawingAreaCallbackStruct*) call-
Data;

    // Set the window into which GL drawing should be done.
    GLwDrawingAreaMakeCurrent(w, glx_context);

    // Draw the scene.
    draw_the_scene();

} // End Form::exposeCB()
```

```

void Form::initCB ( Widget w, XtPointer callData )
{

    //--- Comment out the following line when Form::initCB is implemented:

    // ::VkUnimplemented ( w, "Form::initCB" );

    //--- Add application code for Form::initCB here:

    GLwDrawingAreaCallbackStruct *glptr = (GLwDrawingAreaCallbackStruct*) call-
Data;

    Arg wargs[1];        // Arg temp.

    XVisualInfo *vi;     // Pointer to XVisualInfo.

    // Get the visual info...
    XtSetArg(wargs[0], GLwNvisualInfo, &vi);
    XtGetValues(w, wargs, 1);

    // Create a new GLX rendering context.
    // GL_TRUE -> Specify direct connection to graphics system (if possible).
    glx_context = glXCreateContext(XtDisplay(w), vi, 0, GL_TRUE);

    // Make this drawing area the current one.
    GLwDrawingAreaMakeCurrent(w, glx_context);

    // Set the global window and display.
    global_display = XtDisplay(w);
    global_window  = XtWindow(w);

    // Set the global window size parameters.
    window_x_size = glptr->width;
    window_y_size = glptr->height;

    // Get a font we can use to display our messages ...
    fontHandle = makeRasterFont(w, "-*-times-medium-r-normal--17-*-*-*-p-84-
iso8859-1");

    // Set the Viewport for the first draw of the window.
    glViewport (0, 0, glptr->width, glptr->height);

} // End Form::initCB()

```

```

void Form::inputCB ( Widget w, XtPointer callData )
{
    XmAnyCallbackStruct *cbs = (XmAnyCallbackStruct*) callData;

    //--- Comment out the following line when Form::inputCB is implemented:

    // ::VkUnimplemented ( w, "Form::inputCB" );

    //--- Add application code for Form::inputCB here:

    GLwDrawingAreaCallbackStruct *glptr = (GLwDrawingAreaCallbackStruct*) call-
Data;

    char ascii[1];    // Hold for the ascii chars returned from XLookupString.

    int nchars;      // Number of characters returned from XLookupString.

    KeySym keysym;   // The X version of the returned character.

    XKeyEvent *ptr;  // ptr to the Key Event structure.

    float wx, wy;    // Location of the mouse in world coordinates.

    // We look at the type of event to see if we should pay attention...
    switch(glptr->event->type)
    {

        case KeyRelease:
            // We must convert the keycode to a KeySym before it is possible
            // to check if it is an escape. We also dump the Ascii into the
            // array ascii. The return value from XLookupString is the
            // number of characters dumped into array ascii.
            ptr = (XKeyEvent *) glptr->event;
            nchars = XLookupString(ptr, ascii, 1, &keysym, NULL);

            if(nchars == 1 && keysym == (KeySym) XK_Escape)
            {
                // We have an escape. Time to exit.
                exit(0);
            }

            break;

        case ButtonPress:
            // We have a button press from the mouse.
            // Which mouse button was it?
            switch(glptr->event->xbutton.button)
            {
                case Button1:    // The left button was pressed.
                case Button2:    // The middle button was pressed.
                case Button3:    // The right button was pressed.

```

```

        break;

    default:
        break;

} // end switch on which mouse button it was.
break;

case ButtonRelease:
    // We have a button release from the mouse.
    // Which mouse button was it?
    switch(glptr->event->xbutton.button)
    {
        case Button1: // The left button was released.
        case Button2: // The middle button was released.
        case Button3: // The right button was released.

            break;

        default:
            break;

    } // end switch on which mouse button it was.
    break;

case MotionNotify:
    // We have motion on the mouse.
    // We only get notified of motion when a mouse button is pressed.

    break;

default:
    break;

} // end of switch on event->type.

} // End Form::inputCB()

```

```

void Form::resizeCB ( Widget w, XtPointer callData )
{
    XmAnyCallbackStruct *cbs = (XmAnyCallbackStruct*) callData;

    //--- Comment out the following line when Form::resizeCB is implemented:

    // ::VkUnimplemented ( w, "Form::resizeCB" );

    //--- Add application code for Form::resizeCB here:

    GLwDrawingAreaCallbackStruct *glptr = (GLwDrawingAreaCallbackStruct*) call-
Data;

    // Reset the global window size parameters.
    window_x_size = glptr->width;
    window_y_size = glptr->height;

    // Set the window into which GL drawing should be done.
    GLwDrawingAreaMakeCurrent(w, glx_context);

    // Set the viewport using the window size currently set.
    glViewport (0, 0, glptr->width, glptr->height);

    // Draw the scene.
    draw_the_scene();

} // End Form::resizeCB()

void Form::setParent( VkWindow * parent )
{
    // Store a pointer to the parent VkWindow. This can
    // be useful for accessing the menubar from this class.

    _parent = parent;

} // End Form::setParent()

void Form::copy( )
{
    // This member function is called when the user has selected
    // the Copy command from the Edit menu. You should copy the
    // selected object in your program to on the clipboard.

    //--- Comment out this line when this function is implemented:

    ::VkUnimplemented ( NULL, "Form::copy" );
}

```

```

        //--- Add application code for Form::copy here:

} // End Form::copy()

void Form::cut(    )
{
    // This member function is called when the user has selected
    // the Cut command from the Edit menu. You should cut the
    // selected object in your program and place it on the clipboard.

    //--- Comment out this line when this function is implemented:

    ::VkUnimplemented ( NULL, "Form::cut" );

    //--- Add application code for Form::cut here:

} // End Form::cut()

void Form::newFile(    )
{

    //--- Comment out this line when this function is implemented:

    ::VkUnimplemented ( NULL, "Form::newFile" );

    //--- Add application code for Form::newFile here:

} // End Form::newFile()

```

```

void Form::openFile( const char * filename )
{
    // This member function is called after the user has selected a new
    // file to be opened. The name of the file is given by the
    // filename argument. You can get additional information by
    // examining the state of theFileSelectionDialog object.

    //--- Comment out this line when this function is implemented:

    // ::VkUnimplemented ( NULL, "Form::openFile" );

    //--- Add application code for Form::openFile here:

    // Get the length of the string.
    int len = strlen(filename);

    // Read the data from the file...
    if(filename[len-1] == 'b')
    {
        globalImage.read_from(filename);
    }
    else if (filename[len-1] == 'f')
    {
        globalImage.read_from_gif_file(filename);
    }

    // Let's change the size of the window here ...
    changeWindowSize( _glwidget, globalImage.size[0], globalImage.size[1]);
} // End Form::openFile()

void Form::paste(    )
{
    // This member function is called when the user has selected
    // the Paste command from the Edit menu. You should retrieve
    // the contents of the clipboard and insert it into your
    // program as appropriate.

    //--- Comment out this line when this function is implemented:

    // ::VkUnimplemented ( NULL, "Form::paste" );

    //--- Add application code for Form::paste here:

} // End Form::paste()

```

```

void Form::print( const char * filename )
{

    //--- Comment out this line when this function is implemented:

    ::VkUnimplemented ( NULL, "Form::print" );

    //--- Add application code for Form::print here:

} // End Form::print()

void Form::resetCB ( Widget w, XtPointer callData )
{
    XmAnyCallbackStruct *cbs = (XmAnyCallbackStruct*) callData;

    //--- Comment out the following line when Form::resetCB is implemented:

    // ::VkUnimplemented ( w, "Form::resetCB" );

    //--- Add application code for Form::resetCB here:

    // Zap the LineHold.

} // End Form::resetCB()

```

```

void Form::save( const char * filename )
{
    // This member function is called after the user has selected
    // a file to which to save. The name of the file is given by the
    // filename argument. You can get additional information by
    // examining the state of theFileSelectionDialog object.

    //--- Comment out this line when this function is implemented:

    // ::VkUnimplemented ( NULL, "Form::save" );

    //--- Add application code for Form::save here:

    // Save the data in the file ...
    globalImage.write_to(filename);

} // End Form::save()

void Form::saveas( const char * filename )
{
    // This member function is called after the user has selected
    // a file to which to save. The name of the file is given by the
    // filename argument. You can get additional information by
    // examining the state of theFileSelectionDialog object.

    //--- Comment out this line when this function is implemented:

    // ::VkUnimplemented ( NULL, "Form::saveas" );

    //--- Add application code for Form::saveas here:

    // Save the data in the file ...
    globalImage.write_to(filename);

} // End Form::saveas()

//---- End generated code section

```

```

//
// draw_the_scene
// This function draws the picture.
//

void Form::draw_the_scene()
{
    float xdelta, ydelta; // How much we should shift the image over.

    // Set the background color to black (RGBA).
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);

    // Set the projection matrix.
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, (double) window_x_size, 0.0, (double) window_y_size);

    // Load the ModelView matrix with a unit matrix.
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    // Compute the location of where we should display the image.
    if(window_x_size > globalImage.size[0] && window_y_size > globalImage.size[1])
    {
        // The bitmap is smaller than the displayable area.
        // Shift over and up.
        xdelta = (window_x_size - globalImage.size[0])/2.0;
        ydelta = (window_y_size - globalImage.size[1])/2.0;
    }
    else
    {
        // We have a window the same size as the image.
        xdelta = 0.0;
        ydelta = 0.0;
    }

    // Draw the image.
    glRasterPos2f(xdelta, ydelta);
    globalImage.display();
} // end of draw_the_scene

```

```

void Form::changeWindowSize( Widget glwidget, int width, int height)
{
    // Let's change the size of the window here ...
    Arg wargs[5];

    // Here we check the size to make sure its not too small...
    if(width < 300)
    {
        width = 300;
    }

    if(height < 300)
    {
        height = 300;
    }

    XtSetArg( wargs[0], XmNwidth, width);
    XtSetArg( wargs[1], XmNheight, height);

    // Resize the entire widget chain starting from the child.
    // glwidget and Form should be size of image.
    XtSetValues( glwidget, wargs, 2); // glwidget
    XtSetValues( XtParent(glwidget), wargs, 2); // Form

    // Above Form, these guys should be larger by 28 pixels.
    // (I looked this up in file zimage.uil - this is the size of the
    // menu bar.)
    XtSetArg( wargs[1], XmNheight, height + 28);

    XtSetValues( XtParent(XtParent(glwidget)), wargs, 2); // Above Form
    XtSetValues( XtParent(XtParent(XtParent(glwidget))), wargs, 2); // Above
that ...

    // Set the global window sizes from this computation ...
    // We need the size of the glwidget, not the expanded size.
    window_x_size = width;
    window_y_size = height;
}

```

Makefile

```
#!/smake
#
# Makefile for zimage
# Generated by RapidApp
#
# This makefile follows various conventions used by SGI makefiles
# See the RapidApp User's Guide for more information
#
include $(ROOT)/usr/include/make/commondefs
#
# Local Definitions
#

# Directory in which inst images are placed

IMAGEDIR= images

# The GL library being used, if needed

GLLIBS=-lGLw -lGLU -lGL -limage
COMPONENTLIBS=

#
# The ViewKit stub help library (-lvkhelp) provides a simple
# implementation of the SGI help API. Changing this to -ldesktopUtil
# switches to the full IRIS Insight help system
#

HELPLIB= -lvkhelp

# Standard ViewKit header and libraries

VIEWKITFLAGS= -I$(ROOT)/usr/include/Vk
TOOLTALKLIBS=
NETLS=

EZLIB = -lvkEZ
VIEWKITLIBS= $(TOOLTALKLIBS) $(EZLIB) -lvk $(HELPLIB) $(NETLS) -lSgm -lXpm

# Local C++ options.
# woff 3262 shuts off warnings about arguments that are declared
# but not referenced.

WOFF= -woff 3262

LCXXOPTS = -nostdinc -I$(ROOT)/usr/include $(SAFLAG) $(WOFF) $(VIEWKITFLAGS)

# Add Additional libraries to USERLIBS:

USERLIBS=
```

```

LLDLIBS = -L$(ROOT)/usr/lib $(USERLIBS) $(COMPONENTLIBS) $(VIEWKITLIBS)
$(GLLIBS) -lXm -lXt -lX11 -lgen

# While developing, leave OPTIMIZER set to -g.
# For delivery, change to -O2

OPTIMIZER= -g

# SGI makefiles don't recognize all C++ suffixes, so set up
# the one being used here.

CXXO3=$(CXXO2:.C=.o)
CXXOALL=$(CXXO3)

#
# Source Files generated by the builder. If files are added
# manually, add them to USERFILES
#

BUILDERFILES = main.C\
               Form.C\
               FormUI.C\
               VkwindowMainWindow.C\
               unimplemented.C\
               $(NULL)

#
# Add any files added outside the builder here
#

USERFILES = makeRasterFont.C NPSimage.C

C++FILES = $(BUILDERFILES) $(USERFILES)

#
# The program being built
#

TARGETS=zimage
APPDEFAULTS=Zimage
default all: $(TARGETS)

$(TARGETS): $(OBJECTS)
               $(C++) $(OPTIMIZER) $(OBJECTS) $(LDFLAGS) -o $@

#
# The VkUnimplemented function used for Fix+Continue support
#

unimplemented.o: unimplemented.C
                  $(C++) -c -O unimplemented.C

#

```

```

# These flags instruct the compiler to output
# analysis information for cvstatic
# Uncomment to enable
# Be sure to also disable smake if cvstatic is used

#SADIR= Zimage.cvdb
#SAFLAG= -sa,$(SADIR)
#$(OBJECTS):$(SADIR)/cvdb.dbd
#$(SADIR)/cvdb.dbd :
#      [ -d $(SADIR) ] || mkdir $(SADIR)
#      cd $(SADIR); initcvdb.sh

#LDIRT=$(SADIR) vista.taf

#
# To install on the local machine, do 'make install'
#

install: all
        $(INSTALL) -F /usr/lib/X11/app-defaults Zimage
        $(INSTALL) -F /usr/sbin zimage

#
# To create inst images, do 'make image'
# An image subdirectory should already exist
#

$(IMAGEDIR):
        @mkdir $(IMAGEDIR)
image: $(TARGETS) $(IMAGEDIR)
        /usr/sbin/gendist -rbase / -sbase / -idb zimage.idb \
        -spec zimage.spec \
        -dist /work2/zyda/RapidApp-OpenGL/Zimage/images -all

include $(COMMONRULES)
# DO NOT DELETE

```

NPSImage.h

```
//  
// This is file NPSImage.h  
//  
// This file holds the class definitions for an NPSImage.  
//  
  
#ifndef NPSIMAGE_CLASS  
  
#define NPSIMAGE_CLASS 1  
  
class NPSImage  
{  
  
    public:  
  
        // We need to be able to get at most of the data.  
  
        unsigned int *imagedata;    // Ptr to the image data  
  
        int size[3];                // Image sizes (x, y & z).  
  
        char *name;                 // Name of the image.  
  
        // Here are the functions to access the NPSImage.  
  
        NPSImage();                // Constructor.  
  
        ~NPSImage();              // Destructor.  
  
        void del();               // Delete the NPSImage.  
  
        // Create an image of a particular size.  
        NPSImage(char *name_img, int *size_img);  
  
        // Create an image of a particular size.  
        NPSImage(char *name, int xsize, int ysize, int zsize);  
  
        // Read an NPSImage from a particular file.  
        void read_from(const char *filename);  
  
        // Read a gif image as an NPSImage.  
        void read_from_gif_file(const char *filename);  
  
        // Write an NPSImage to a particular file.  
        void write_to(const char *filename);  
  
        // Display the image.  
        void display();  
  
        // Specify an operator of = (copy) for an NPSImage.  
        NPSImage& operator=(NPSImage&);  
};
```

```
// Specify an operator of == for 2 NPSimages.  
int is_equal(NPSimage& img);  
  
// Replace all colors that look like this...  
void color_replace(unsigned int oldcor, unsigned int newclr);  
  
private:  
  
};  
  
#endif
```

NPSImage.C

```
//
// This is file NPSImage.C
//
// This file contains the methods that implement the
// NPSImage class.
//

#include "NPSImage.h"

#include <string.h>

#include <iostream.h>      // Get the I/O stuff.
#include <fstream.h>      // Get the streams file I/O stuff.

#include <GL/gl.h>        // Get the SGI routines.
#include <GL/glu.h>

#include "image.h"        // SGI image structures.

// External def for SGI image routines.

extern "C" IMAGE *iopen(const char *filename, char *access, ...);

extern "C" void getrow(IMAGE *image, unsigned short *buf, int y, int z);

extern "C" void putrow(IMAGE *image, unsigned short *buf, int y, int z);

extern "C" void iclose(IMAGE *);

// temp arrays to hold scratch info for processing an sgi image.
// RGBA order for the indices.
unsigned short buf[4][4096];

// Here is a constructor with no specs.
NPSImage::NPSImage()
{

    // We haven't been given an image size.

    // Create a default image.
    size[0] = 8;
    size[1] = 8;
    size[2] = 4;

    // Pt the image data at a simple 8 x 8 x 4 image.
    imagedata = new unsigned int [size[0] * size[1]];
}
```

```

    // No name for the image yet.
    name = new char [8];
    strcpy(name,"default");
}

// Here is the destructor for an NPSImage.
NPSImage::~NPSImage()
{
    // Call the del() function.
    del();
}

// Delete the image.
void NPSImage::del()
{
    // Delete the image data if it exists.
    if(imagedata != 0)
    {
        delete imagedata;
    }

    // Delete the image's name if it exists.
    if(name != 0)
    {
        delete name;
    }
}

```

```

// Create an empty image of a particular size.
// We pass in the name and size of the image to create.
NPSImage::NPSImage(char *name_img, int *size_img)
{
    // Create an image.

    // Copy the size.
    size[0] = size_img[0];
    size[1] = size_img[1];
    size[2] = size_img[2];

    // Allocate an image of the appropriate size.
    imagedata = new unsigned int [size[0] * size[1]];

    // No name for the image yet.
    name = new char [strlen(name_img) + 1];
    strcpy(name, name_img);
}

// Create an empty image of a particular size.
// We pass in the name and size of the image to create.
NPSImage::NPSImage(char *name_img, int xsize, int ysize, int zsize)
{
    // Create an image.

    // Copy the size.
    size[0] = xsize;
    size[1] = ysize;
    size[2] = zsize;

    // Allocate an image of the appropriate size.
    imagedata = new unsigned int [size[0] * size[1]];

    // No name for the image yet.
    name = new char [strlen(name_img) + 1];
    strcpy(name, name_img);
}

```

```

// We need to read in an image from a particular file.
void NPSImage::read_from(const char *filename)
{
    IMAGE *image;           // a ptr to an SGI image structure.

    int x,y,z;             // Loop temps.

    unsigned int *ptr;     // Ptr to the image longs.

    // open an sgi image
    image = iopen(filename,"r");
    if(image == NULL)
    {
        cerr << "NPSImage::read_from: can't open input file " << filename << endl;
        return;
    }

    // If an NPSImage is previously defined, delete it.
    NPSImage::~NPSImage();

    // I have commented this out. We really ought to try and delete
    // any old image data BUT with the new compiler, there is no
    // reliable way to make sure there was a previously valid pointer ...

    // We have the image open and the old image deleted...

    // Create an empty image of the right size.

    // Copy the size.
    size[0] = image->xsize;
    size[1] = image->ysize;
    size[2] = image->zsize;

    // Allocate an image of the appropriate size.
    imagedata = new unsigned int [size[0] * size[1]];

    // Copy the filename into the image.
    name = new char [strlen(filename) + 1];
    strcpy(name, filename);

    // get a pointer to the NPSImage longs.
    ptr = imagedata;

    // for each row of the image.
    for(y=0; y < size[1]; y=y+1)
    {
        for(z=0; z < size[2]; z=z+1)
        {
            // Read each row in reds, greens, blues, alpha order.

```

```

        getrow(image, buf[z], y, z);
    }

    // we must now step across the row and set each long integer
    // of the NPSimage format by combining the info from the sgi
    // rows.
    for(x=0; x < size[0]; x=x+1)
    {
        // Someday we should generalize the below code to make it useable
        // for images with only 1 and 2 channels.

        // compute the RGBa long to plug in and plug it in.
        if(size[2] == 4)
        {
            // Data reversed for OpenGL implementation.
            *ptr = (buf[0][x] << 24) + (buf[1][x] << 16) + (buf[2][x] << 8) +
(buf[3][x]);
        }
        else
        {
            // RGBA image with alpha forced to 0xff
            // Originally: *ptr = buf[0][x] + (buf[1][x] << 8) + (buf[2][x] << 16)
+ (0xff << 24);
            // Images are stored in memory differently in OpenGL (RGBA) instead of
            // ABGR as in IRISGL.
            *ptr = (buf[0][x] << 24) + (buf[1][x] << 16) + (buf[2][x] << 8) + (0xff);
        }

        // step the ptr to the next long.
        ptr++;
    }
}

// Close the image file.
fclose(image);
}

```

```

// We need to write out the image in SGI format.
void NPSImage::write_to(const char *filename)
{

    register IMAGE *image;    // a ptr to an SGI image structure.

    int x,y,z;                // Loop temps.

    unsigned int *ptr;        // Ptr to the image longs.

    // open an sgi rgb image for writing.
    image=ipopen(filename,"w",RLE(1),size[2],size[0],size[1],size[2]);

    // get a pointer to the NPSImage longs.
    ptr = imagedata;

    // for each row of the image ...
    for(y=0; y < size[1]; y=y+1)
    {

        // we must now step across the row and decode each integer
        // of the NPSImage format into the 16 bit shorts sgi requires.
        for(x=0; x < size[0]; x=x+1)
        {

            // Get the colors from the longs. Reversed for OpenGL.
            buf[0][x] = (unsigned short)((*ptr & 0xff000000) >> 24);
            buf[1][x] = (unsigned short)((*ptr & 0x00ff0000) >> 16);
            buf[2][x] = (unsigned short)((*ptr & 0x0000ff00) >> 8);
            buf[3][x] = (unsigned short)(*ptr & 0x000000ff);

            // step the ptr to the next long.
            ptr++;

        }

        // Write out the rows of the image.
        for(z=0; z < size[2]; z=z+1)
        {

            // Write each row in reds, greens, blues, alpha order.
            putrow(image, buf[z], y, z);

        }

    }

    // we must close the output sgi image file.
    iclose(image);

}

```

```

// We need to display the image.
void NPSImage::display()
{
    // Here is the OpenGL call to send the image to the open window.
    glDrawPixels(size[0], size[1], GL_RGBA, GL_UNSIGNED_BYTE, imagedata);
}

```

```

// The purpose of the = operator is for copying an NPSImage.

```

```

NPSImage&
NPSImage::operator=(NPSImage &img)
{
    int i; // Loop temp

    // Copy the size.
    size[0] = img.size[0];
    size[1] = img.size[1];
    size[2] = img.size[2];

    // Allocate an image of the appropriate size.
    imagedata = new unsigned int [size[0] * size[1]];

    // Copy the image data from img to "this".
    for(i=0; i < (size[0] * size[1]); i=i+1)
    {
        imagedata[i] = img.imagedata[i];
    }

    // Copy the filename into the image.
    name = new char [strlen(img.name) + 1];
    strcpy(name, img.name);

    // Return a ptr to the new image.
    return *this;
}

```

```

// The purpose of the is_equal function is for comparing 2 NPSimages.
// We return 1 if the images are equal, otherwise 0.
int NPSImage::is_equal(NPSImage &img)
{

    int i; // Loop temp

    // Compare the sizes.
    if(size[0] != img.size[0] || size[1] != img.size[1]
        || size[2] != img.size[2])
    {
        // The sizes are not equal, return 0.
        return 0;
    }

    // Compare the image data.
    for(i=0; i < (size[0] * size[1]); i=i+1)
    {
        if(imagedata[i] != img.imagedata[i])
        {
            // There is a difference between the images.
            return 0;
        }
    }

    // If we get here, the images are equal.
    // Note: we don't check to see if the names are equal.
    return 1;
}

```

```

// The purpose of the color_replace function is for
// changing all occurrences of the old color to the new color.
void NPSImage::color_replace(unsigned int oldclr, unsigned int newclr)
{

    int i; // Loop temp

    // Compare the image data.
    for(i=0; i < (size[0] * size[1]); i=i+1)
    {
        if(imagedata[i] == oldclr)
        {
            // We have the old color, replace it.
            imagedata[i] = newclr;
        }
    }
}

```

```
// We need to read in an image from a particular gif file.

void NPSImage::read_from_gif_file(const char *filename)
{

    char tmp[256];    // temp char array.

    // If an NPSImage is previously defined, delete it.
    NPSImage::~NPSImage();

    // Convert the input file to a .rgb file!
    // fromgif file.gif ztemp.rgb is sent to the system ...
    strcpy(tmp, "fromgif ");
    strcat(tmp, filename);
    strcat(tmp, " ztemp.rgb");
    system(tmp);

    read_from("ztemp.rgb");
}
}
```

Zimage

```
!  
! Generated by Silicon Graphic's RapidApp.  
!  
!  
! RapidApp 1.0.  
!  
!  
!  
!Activate schemes and sgi mode by default  
!  
Zimage*useSchemes: all  
Zimage*sgiMode: true  
!  
!SGI Style guide specifies pointer focus for applications  
!  
Zimage*keyboardFocusPolicy: pointer  
  
Zimage*vkwindow.title: Zimage  
Zimage*filePane.labelString: File  
Zimage*filePane.mnemonic: F  
Zimage*newButton.labelString: New  
Zimage*newButton.mnemonic: N  
Zimage*newButton.accelerator: Ctrl<Key>N  
Zimage*newButton.acceleratorText: Ctrl+N  
Zimage*openButton.labelString: Open...  
Zimage*openButton.mnemonic: O  
Zimage*openButton.accelerator: Ctrl<Key>O  
Zimage*openButton.acceleratorText: Ctrl+O  
Zimage*saveButton.labelString: Save  
Zimage*saveButton.mnemonic: S  
Zimage*saveButton.accelerator: Ctrl<Key>S  
Zimage*saveButton.acceleratorText: Ctrl+S  
Zimage*saveasButton.labelString: Save As...  
Zimage*saveasButton.mnemonic: A  
Zimage*printButton.labelString: Print  
Zimage*printButton.mnemonic: P  
Zimage*printButton.accelerator: Ctrl<Key>P  
Zimage*printButton.acceleratorText: Ctrl+P  
Zimage*closeButton.labelString: Close  
Zimage*closeButton.mnemonic: C  
Zimage*closeButton.accelerator: Ctrl<Key>W  
Zimage*closeButton.acceleratorText: Ctrl+W  
Zimage*exitButton.labelString: Exit  
Zimage*exitButton.mnemonic: x  
Zimage*exitButton.accelerator: Ctrl<Key>Q  
Zimage*exitButton.acceleratorText: Ctrl+Q  
Zimage*editPane.labelString: Edit  
Zimage*editPane.mnemonic: E  
Zimage*undoButton.labelString: Undo  
Zimage*undoButton.mnemonic: U  
Zimage*undoButton.accelerator: Ctrl<Key>Z  
Zimage*undoButton.acceleratorText: Ctrl+Z  
Zimage*cutButton.labelString: Cut
```

```
Zimage*cutButton.mnemonic: t
Zimage*cutButton.accelerator: Ctrl<Key>X
Zimage*cutButton.acceleratorText: Ctrl+X
Zimage*copyButton.labelString: Copy
Zimage*copyButton.mnemonic: C
Zimage*copyButton.accelerator: Ctrl<Key>C
Zimage*copyButton.acceleratorText: Ctrl+C
Zimage*pasteButton.labelString: Paste
Zimage*pasteButton.mnemonic: P
Zimage*pasteButton.accelerator: Ctrl<Key>V
Zimage*pasteButton.acceleratorText: Ctrl+V
Zimage*viewPane.labelString: View
Zimage*viewPane.mnemonic: V
Zimage*viewControl1.labelString:
Zimage*viewControl2.labelString: Reset Picture
Zimage*viewControl3.labelString:
Zimage*optionsPane.labelString: Options
Zimage*optionsPane.mnemonic: O
Zimage*helpPane.labelString: Help
Zimage*helpPane.mnemonic: H
Zimage*help_click_for_help.labelString: Click For Help
Zimage*help_click_for_help.mnemonic: C
Zimage*help_click_for_help.accelerator: Shift<Key>F1
Zimage*help_click_for_help.acceleratorText: Shift+F1
Zimage*help_overview.labelString: Overview
Zimage*help_overview.mnemonic: O
Zimage*help_index.labelString: Index
Zimage*help_index.mnemonic: I
Zimage*help_keys_and_short.labelString: Keys and Shortcuts
Zimage*help_keys_and_short.mnemonic: K
Zimage*help_prod_info.labelString: Product Information
Zimage*help_prod_info.mnemonic: P
```

```
!
! The following resources are for classes
! and instances of classes.
!
```

```
!!-- End generated defaults
```

zimage.uil

```
!0x000102EFFFF
! Generated by Silicon Graphic's RapidApp.
!
!
! RapidApp 1.0.
!
module main_uil
version = 'V1.0'
names = case_sensitive
!(BX) bx_info("include_path", ".",
!(BX)          "/usr/include/Mrm",
!(BX)          "/usr/include/uil", false )
!(BX) bx_info("c_information", true )
!(BX) bx_info("app_class", "Zimage", true )
!(BX) bx_info("app_name", "zimage", true )
!(BX) bx_info("c++_pixmaps", "pixmaps.h", true )
!(BX) bx_info("c++_baseclass", "VkComponent", false )
!(BX) bx_info("c++_main", "main.C", true )
!(BX) bx_info("c++_makefile", "Makefile", true )
!(BX) bx_info("dont_merge_files", false )
!(BX) bx_info("strip_dimensions", false )
!(BX) bx_info("use_ez", true )
!(BX) bx_info("no_standalone_classes", true )
!(BX) bx_info("generate_default_resources", false )
!(BX) bx_info("use_cvstatic", false )
!(BX) bx_info("use_runonce", false )
!(BX) bx_info("use_license", false )
!(BX) bx_info("use_tooltalk", false )
!(BX) bx_info("uil_utilities", "bxutil-uil.c", true )
!(BX) bx_info("uil_constants", "main-uil.h", true )
!(BX) bx_info("uil_callbacks", "callbacks-uil.c", true )
!(BX) bx_info("uil_uil", "zimage.uil", true )
!(BX) bx_info("uil_main", "main-uil.c", true )
!(BX) bx_info("uil_imakefile", "Imakefile", true )
!(BX) bx_info("uil_makefile", "makefile-uil", true )
!(BX) bx_info("c_utilities", "bxutil-c.c", true )
!(BX) bx_info("c_pixmaps", "pixmaps.h", true )
!(BX) bx_info("c_callbacks", "callbacks-c.c", true )
!(BX) bx_info("c_creation", "creation-c.c", true )
!(BX) bx_info("c_main", "main-c.c", true )
!(BX) bx_info("c_imakefile", "Imakefile", true )
!(BX) bx_info("c_makefile", "Makefile", true )
!(BX) bx_info("app_defaults", "app-defaults", true )

list BaseStyle : arguments {
};

list BaseStyleReasons : callbacks {
};

!(BX)object topLevelShell : TopLevelShell widget {
```

```

!(BX) arguments {
!(BX)     XmNtitle = 'Zimage';
!(BX)     XmNiconName = 'Zimage';
!(BX)     XmNx = 314;
!(BX)     XmNy = 327;
!(BX)     XmNwidth = 750;
!(BX)     XmNheight = 600;
!(BX) };
!(BX) controls {
!(BX)     managed VkWindow vkwindow;
!(BX) };
!(BX) callbacks {
!(BX) };
!(BX)};

object vkwindow : VkWindow widget {
    arguments {
!(BX)     _XmNtitle = 'Zimage';
!(BX)     XmNdisableWindowResize = true;
!(BX)     XmNwidth = 750;
!(BX)     XmNheight = 600;
    };
    controls {
        managed XmMenuBar menuBar;
        managed XmForm form;
    };
    callbacks {
    };
};

object menuBar : XmMenuBar widget {
    arguments {
        XmNmenuHelpWidget = XmCascadeButton helpPane;
        XmNwidth = 750;
        XmNheight = 28;
    };
    controls {
        managed XmCascadeButton filePane;
        managed XmCascadeButton editPane;
        managed XmCascadeButton viewPane;
        managed XmCascadeButton optionsPane;
        managed XmCascadeButton helpPane;
    };
    callbacks {
    };
};

object filePane : XmCascadeButton widget {
    arguments {
!(BX)     _XmNlabelString = compound_string("File");
!(BX)     _XmNmnemonic = keysym("F");
!(BX)     XmNwidth = 43;
!(BX)     XmNheight = 24;
    };
};

```

```

    controls {
        unmanaged XmPulldownMenu pulldownMenu;
    };
    callbacks {
    };
};

object pulldownMenu : XmPulldownMenu widget {
    arguments {
        XmNwidth = 156;
        XmNheight = 176;
    };
    controls {
        managed XmPushButtonGadget newButton;
        managed XmPushButtonGadget openButton;
        managed XmPushButtonGadget saveButton;
        managed XmPushButtonGadget saveasButton;
        managed XmPushButtonGadget printButton;
        managed XmSeparatorGadget separator1;
        managed XmPushButtonGadget closeButton;
        managed XmPushButtonGadget exitButton;
    };
    callbacks {
    };
};

object newButton : XmPushButtonGadget widget {
    arguments {
        !(BX) _XmNlabelString = compound_string("New");
        !(BX) _XmNmnemonic = keysym("N");
        !(BX) _XmNaccelerator = 'Ctrl<Key>N';
        !(BX) _XmNacceleratorText = compound_string("Ctrl+N");
    };
    controls {
    };
    callbacks {
        XmNactivateCallback = procedure newFile();
    };
};

object openButton : XmPushButtonGadget widget {
    arguments {
        !(BX) _XmNlabelString = compound_string("Open...");
        !(BX) _XmNmnemonic = keysym("O");
        !(BX) _XmNaccelerator = 'Ctrl<Key>O';
        !(BX) _XmNacceleratorText = compound_string("Ctrl+O");
    };
    controls {
    };
    callbacks {
        XmNactivateCallback = procedure openFile();
    };
};

```

```

object saveButton : XmPushButtonGadget widget {
  arguments {
    !(BX)  _XmNlabelString = compound_string("Save");
    !(BX)  _XmNmnemonic = keysym("S");
    !(BX)  _XmNaccelerator = 'Ctrl<Key>S';
    !(BX)  _XmNacceleratorText = compound_string("Ctrl+S");
  };
  controls {
  };
  callbacks {
    XmNactivateCallback = procedure save();
  };
};

object saveasButton : XmPushButtonGadget widget {
  arguments {
    !(BX)  _XmNlabelString = compound_string("Save As...");
    !(BX)  _XmNmnemonic = keysym("A");
  };
  controls {
  };
  callbacks {
    XmNactivateCallback = procedure saveas();
  };
};

object printButton : XmPushButtonGadget widget {
  arguments {
    !(BX)  _XmNlabelString = compound_string("Print");
    !(BX)  _XmNmnemonic = keysym("P");
    !(BX)  _XmNaccelerator = 'Ctrl<Key>P';
    !(BX)  _XmNacceleratorText = compound_string("Ctrl+P");
  };
  controls {
  };
  callbacks {
    XmNactivateCallback = procedure print();
  };
};

object separator1 : XmSeparatorGadget widget {
  arguments {
  };
  controls {
  };
  callbacks {
  };
};

object closeButton : XmPushButtonGadget widget {
  arguments {
    !(BX)  _XmNlabelString = compound_string("Close");
    !(BX)  _XmNmnemonic = keysym("C");
    !(BX)  _XmNaccelerator = 'Ctrl<Key>W';
  };
};

```

```

!(BX)  _XmNacceleratorText = compound_string("Ctrl+W");
};
controls {
};
callbacks {
    XmNactivateCallback = procedure close();
};
};

object exitButton : XmPushButtonGadget widget {
    arguments {
!(BX)  _XmNlabelString = compound_string("Exit");
!(BX)  _XmNmnemonic = keysym("x");
!(BX)  _XmNaccelerator = 'Ctrl<Key>Q';
!(BX)  _XmNacceleratorText = compound_string("Ctrl+Q");
};
controls {
};
callbacks {
    XmNactivateCallback = procedure quit();
};
};

object editPane : XmCascadeButton widget {
    arguments {
!(BX)  _XmNlabelString = compound_string("Edit");
!(BX)  _XmNmnemonic = keysym("E");
        XmNwidth = 47;
        XmNheight = 24;
};
controls {
    unmanaged XmPulldownMenu pulldownMenu1;
};
callbacks {
};
};

object pulldownMenu1 : XmPulldownMenu widget {
    arguments {
        XmNwidth = 125;
        XmNheight = 102;
};
controls {
    managed XmPushButtonGadget undoButton;
    managed XmPushButtonGadget cutButton;
    managed XmPushButtonGadget copyButton;
    managed XmPushButtonGadget pasteButton;
};
callbacks {
};
};

object undoButton : XmPushButtonGadget widget {
    arguments {

```

```

!(BX)  _XmNlabelString = compound_string("Undo");
!(BX)  _XmNmnemonic = keysym("U");
!(BX)  _XmNaccelerator = 'Ctrl<Key>Z';
!(BX)  _XmNacceleratorText = compound_string("Ctrl+Z");
};
controls {
};
callbacks {
    XmNactivateCallback = procedure undo();
};
};

object cutButton : XmPushButtonGadget widget {
    arguments {
!(BX)  _XmNlabelString = compound_string("Cut");
!(BX)  _XmNmnemonic = keysym("t");
!(BX)  _XmNaccelerator = 'Ctrl<Key>X';
!(BX)  _XmNacceleratorText = compound_string("Ctrl+X");
};
controls {
};
callbacks {
    XmNactivateCallback = procedure cut();
};
};

object copyButton : XmPushButtonGadget widget {
    arguments {
!(BX)  _XmNlabelString = compound_string("Copy");
!(BX)  _XmNmnemonic = keysym("C");
!(BX)  _XmNaccelerator = 'Ctrl<Key>C';
!(BX)  _XmNacceleratorText = compound_string("Ctrl+C");
};
controls {
};
callbacks {
    XmNactivateCallback = procedure copy();
};
};

object pasteButton : XmPushButtonGadget widget {
    arguments {
!(BX)  _XmNlabelString = compound_string("Paste");
!(BX)  _XmNmnemonic = keysym("P");
!(BX)  _XmNaccelerator = 'Ctrl<Key>V';
!(BX)  _XmNacceleratorText = compound_string("Ctrl+V");
};
controls {
};
callbacks {
    XmNactivateCallback = procedure paste();
};
};

```

```

object viewPane : XmCascadeButton widget {
    arguments {
!(BX)  _XmNlabelString = compound_string("View");
!(BX)  _XmNmnemonic = keysym("V");
        XmNwidth = 52;
        XmNheight = 24;
    };
    controls {
        unmanaged XmPulldownMenu viewPulldownMenu;
    };
    callbacks {
    };
};

object viewPulldownMenu : XmPulldownMenu widget {
    arguments {
        XmNwidth = 122;
        XmNheight = 46;
    };
    controls {
        managed XmPushButtonGadget viewControl1;
        managed XmPushButtonGadget viewControl2;
        managed XmPushButtonGadget viewControl3;
    };
    callbacks {
    };
};

object viewControl1 : XmPushButtonGadget widget {
    arguments {
!(BX)  _XmNlabelString = compound_string("");
    };
    controls {
    };
    callbacks {
    };
};

object viewControl2 : XmPushButtonGadget widget {
    arguments {
!(BX)  _XmNlabelString = compound_string("Reset Picture");
    };
    controls {
    };
    callbacks {
        XmNactivateCallback = procedure resetCB();
    };
};

object viewControl3 : XmPushButtonGadget widget {
    arguments {
!(BX)  _XmNlabelString = compound_string("");
    };
    controls {

```

```

};
callbacks {
};
};

object optionsPane : XmCascadeButton widget {
    arguments {
!(BX)  _XmNlabelString = compound_string("Options");
!(BX)  _XmNmnemonic = keysym("O");
        XmNwidth = 74;
        XmNheight = 24;
    };
    controls {
        unmanaged XmPulldownMenu pulldownMenu2;
    };
    callbacks {
};
};

object pulldownMenu2 : XmPulldownMenu widget {
    arguments {
        XmNwidth = 84;
        XmNheight = 54;
    };
    controls {
        managed XmToggleButtonGadget option1;
        managed XmToggleButtonGadget option2;
    };
    callbacks {
};
};

object option1 : XmToggleButtonGadget widget {
    arguments {
};
    controls {
};
    callbacks {
};
};

object option2 : XmToggleButtonGadget widget {
    arguments {
};
    controls {
};
    callbacks {
};
};

object helpPane : XmCascadeButton widget {
    arguments {
!(BX)  _XmNlabelString = compound_string("Help");
!(BX)  _XmNmnemonic = keysym("H");
    };

```

```

        XmNwidth = 50;
        XmNheight = 24;
    };
    controls {
        unmanaged XmPulldownMenu pulldownMenu3;
    };
    callbacks {
    };
};

object pulldownMenu3 : XmPulldownMenu widget {
    arguments {
        XmNwidth = 235;
        XmNheight = 128;
    };
    controls {
        managed XmPushButtonGadget help_click_for_help;
        managed XmPushButtonGadget help_overview;
        managed XmPushButtonGadget help_index;
        managed XmPushButtonGadget help_keys_and_short;
        managed XmSeparatorGadget separator;
        managed XmPushButtonGadget help_prod_info;
    };
    callbacks {
    };
};

object help_click_for_help : XmPushButtonGadget widget {
    arguments {
        !(BX) _XmNlabelString = compound_string("Click For Help");
        !(BX) _XmNmnemonic = keysym("C");
        !(BX) _XmNaccelerator = 'Shift<Key>F1';
        !(BX) _XmNacceleratorText = compound_string("Shift+F1");
    };
    controls {
    };
    callbacks {
    };
};

object help_overview : XmPushButtonGadget widget {
    arguments {
        !(BX) _XmNlabelString = compound_string("Overview");
        !(BX) _XmNmnemonic = keysym("O");
    };
    controls {
    };
    callbacks {
    };
};

object help_index : XmPushButtonGadget widget {
    arguments {
        !(BX) _XmNlabelString = compound_string("Index");
    };

```

```

!(BX)  _XmNmnemonic = keysym("I");
};
controls {
};
callbacks {
};
};

object help_keys_and_short : XmPushButtonGadget widget {
arguments {
!(BX)  _XmNlabelString = compound_string("Keys and Shortcuts");
!(BX)  _XmNmnemonic = keysym("K");
};
controls {
};
callbacks {
};
};

object separator : XmSeparatorGadget widget {
arguments {
};
controls {
};
callbacks {
};
};

object help_prod_info : XmPushButtonGadget widget {
arguments {
!(BX)  _XmNlabelString = compound_string("Product Information");
!(BX)  _XmNmnemonic = keysym("P");
};
controls {
};
callbacks {
};
};

object form : XmForm widget {
arguments {
XmNresizePolicy = XmRESIZE_GROW;
XmNwidth = 750;
XmNheight = 572;
};
controls {
managed GLwMDrawingArea glwidget;
};
callbacks {
};
};

object glwidget : GLwMDrawingArea widget {
arguments {

```

```

    GLwNrgba = true;
    GLwNdoublebuffer = false;
    XmNtopAttachment = XmATTACH_FORM;
    XmNbottomAttachment = XmATTACH_FORM;
    XmNleftAttachment = XmATTACH_FORM;
    XmNrightAttachment = XmATTACH_FORM;
    XmNtopPosition = 0;
    XmNtopOffset = 0;
    XmNbottomOffset = 0;
    XmNleftOffset = 0;
    XmNrightOffset = 0;
    XmNx = 0;
    XmNy = 0;
    XmNwidth = 750;
    XmNheight = 572;
};
controls {
};
callbacks {
    GLwNginithCallback = procedure initCB();
    GLwNinputCallback = procedure inputCB();
    GLwNresizeCallback = procedure resizeCB();
    GLwNexposeCallback = procedure exposeCB();
};
};
end module;

```

VkMainWindow.C

```
////////////////////////////////////
//
// Source file for VkwindowMainWindow
//
// This class is a ViewKit VkWindow subclass
//
//
// Normally, very little in this file should need to be changed.
// Create/add/modify menus using the builder.
//
// Try to restrict any changes to the bodies of functions
// corresponding to menu items, the constructor and destructor.
//
// Add any new functions below the "//--- End generated code"
// markers
//
// Doing so will allow you to make changes using the builder
// without losing any changes you may have made manually
//
// Avoid gratuitous reformatting and other changes that might
// make it difficult to integrate changes made using the builder
////////////////////////////////////
#include "VkwindowMainWindow.h"
#include <Vk/VkApp.h>
#include <Vk/VkFileSelectionDialog.h>
#include <Vk/VkSubMenu.h>
#include <Vk/VkRadioSubMenu.h>
#include <Vk/VkMenuItem.h>
#include "Form.h"
//---- End generated headers

// These are default resources for widgets in objects of this class
// All resources will be prepended by *<name> at instantiation,
// where <name> is the name of the specific instance, as well as the
// name of the baseWidget. These are only defaults, and may be overridden
// in a resource file by providing a more specific resource name

String  VkwindowMainWindow::_defaultVkwindowMainWindowResources[] = {
    (char*)NULL
};

//---- Class declaration

VkwindowMainWindow::VkwindowMainWindow( const char *name,
                                         ArgList args,
                                         Cardinal argCount) :
    VkWindow (name, args, argCount)
{
    // Create the view component contained by this window
```

```

_form= new Form("form",mainWindowWidget());

XtVaSetValues ( _form->baseWidget(),
                XmNwidth, 750,
                XmNheight, 572,
                (XtPointer) NULL );

// Add the component as the main view

addView (_form);
_form->setParent(this);

// Set up the window manager properties for this window.
// This window cannot be resized by the user.

VkConfigureWM ( this,
                TRUE, //quit
                TRUE, // close
                TRUE, // border
                TRUE, // title
                FALSE, // allow resize
                TRUE, // allow iconify
                TRUE); // Show window manager menu

// Create the panes of this window's menubar. The menubar itself
// is created automatically by ViewKit

// The filePane menu pane

_filePane = addMenuPane("filePane");

_newButton = _filePane->addAction ( "newButton",
                                   &VkwindowMainWindow::newFileCallback,
                                   (XtPointer) this );

_openButton = _filePane->addAction ( "openButton",
                                     &VkwindowMainWindow::openFileCallback,
                                     (XtPointer) this );

_saveButton = _filePane->addAction ( "saveButton",
                                     &VkwindowMainWindow::saveCallback,
                                     (XtPointer) this );

_saveasButton = _filePane->addAction ( "saveasButton",
                                       &VkwindowMainWindow::saveasCallback,
                                       (XtPointer) this );

_printButton = _filePane->addAction ( "printButton",
                                      &VkwindowMainWindow::printCallback,
                                      (XtPointer) this );

_separator1 = _filePane->addSeparator();

```

```

_closeButton = _filePane->addAction ( "closeButton",
                                       &VkwindowMainWindow::closeCallback,
                                       (XtPointer) this );

_exitButton = _filePane->addAction ( "exitButton",
                                       &VkwindowMainWindow::quitCallback,
                                       (XtPointer) this );

// The editPane menu pane

_editPane = addMenuPane("editPane");

_editPane->add ( theUndoManager );

_cutButton = _editPane->addAction ( "cutButton",
                                       &VkwindowMainWindow::cutCallback,
                                       (XtPointer) this );

_copyButton = _editPane->addAction ( "copyButton",
                                       &VkwindowMainWindow::copyCallback,
                                       (XtPointer) this );

_pasteButton = _editPane->addAction ( "pasteButton",
                                       &VkwindowMainWindow::pasteCallback,
                                       (XtPointer) this );

// The viewPane menu pane

_viewPane = addMenuPane("viewPane");

_viewControl1 = _viewPane->addAction ( "viewControl1",
                                       NULL, // No callback function given
                                       (XtPointer) this );

_viewControl2 = _viewPane->addAction ( "viewControl2",
                                       &VkwindowMainWindow::resetCBCallback,
                                       (XtPointer) this );

_viewControl3 = _viewPane->addAction ( "viewControl3",
                                       NULL, // No callback function given
                                       (XtPointer) this );

// The optionsPane menu pane

_optionsPane = addMenuPane("optionsPane");

_option1 = _optionsPane->addToggle ( "option1",
                                       NULL, // No Callback given
                                       (XtPointer) this );

_option2 = _optionsPane->addToggle ( "option2",
                                       NULL, // No Callback given
                                       (XtPointer) this );

```

```

        //---- End generated code section

} // End Constructor

VkwindowMainWindow::~VkwindowMainWindow()
{
    delete _form;
} // End destructor

const char *VkwindowMainWindow::className()
{
    return ("VkwindowMainWindow");
} // End className()

Boolean VkwindowMainWindow::okToQuit()
{
    // This member function is called when the user quits by calling
    // theApplication->terminate() or uses the window manager close protocol
    // This function can abort the operation by returning FALSE, or do some.
    // cleanup before returning TRUE. The actual decision is normally passed on
    // to the view object

    // Query the view object, and give it a chance to cleanup

    return ( _form->okToQuit() );
} // End okToQuit()

////////////////////////////////////
// The following functions are static member functions used to
// interface with Motif.
////////////////////////////////////

void VkwindowMainWindow::closeCallback ( Widget      w,
                                         XtPointer clientData,
                                         XtPointer callData )
{
    VkwindowMainWindow* obj = ( VkwindowMainWindow * ) clientData;
    obj->close ( w, callData );
}

void VkwindowMainWindow::copyCallback ( Widget      w,
                                        XtPointer clientData,
                                        XtPointer callData )
{
    VkwindowMainWindow* obj = ( VkwindowMainWindow * ) clientData;

```

```

    obj->copy ( w, callData );
}

void VkwindowMainWindow::cutCallback ( Widget    w,
                                       XtPointer clientData,
                                       XtPointer callData )
{
    VkwindowMainWindow* obj = ( VkwindowMainWindow * ) clientData;
    obj->cut ( w, callData );
}

void VkwindowMainWindow::newFileCallback ( Widget    w,
                                           XtPointer clientData,
                                           XtPointer callData )
{
    VkwindowMainWindow* obj = ( VkwindowMainWindow * ) clientData;
    obj->newFile ( w, callData );
}

void VkwindowMainWindow::openFileCallback ( Widget    w,
                                           XtPointer clientData,
                                           XtPointer callData )
{
    VkwindowMainWindow* obj = ( VkwindowMainWindow * ) clientData;
    obj->openFile ( w, callData );
}

void VkwindowMainWindow::pasteCallback ( Widget    w,
                                         XtPointer clientData,
                                         XtPointer callData )
{
    VkwindowMainWindow* obj = ( VkwindowMainWindow * ) clientData;
    obj->paste ( w, callData );
}

void VkwindowMainWindow::printCallback ( Widget    w,
                                         XtPointer clientData,
                                         XtPointer callData )
{
    VkwindowMainWindow* obj = ( VkwindowMainWindow * ) clientData;
    obj->print ( w, callData );
}

void VkwindowMainWindow::quitCallback ( Widget    w,
                                        XtPointer clientData,
                                        XtPointer callData )
{
    VkwindowMainWindow* obj = ( VkwindowMainWindow * ) clientData;
    obj->quit ( w, callData );
}

void VkwindowMainWindow::resetCBCallback ( Widget    w,
                                           XtPointer clientData,
                                           XtPointer callData )

```

```

{
    VkwindowMainWindow* obj = ( VkwindowMainWindow * ) clientData;
    obj->resetCB ( w, callData );
}

void VkwindowMainWindow::saveCallback ( Widget      w,
                                       XtPointer clientData,
                                       XtPointer callData )
{
    VkwindowMainWindow* obj = ( VkwindowMainWindow * ) clientData;
    obj->save ( w, callData );
}

void VkwindowMainWindow::saveasCallback ( Widget      w,
                                         XtPointer clientData,
                                         XtPointer callData )
{
    VkwindowMainWindow* obj = ( VkwindowMainWindow * ) clientData;
    obj->saveas ( w, callData );
}

////////////////////////////////////
// The following functions are called from the menu items
// in this window.
////////////////////////////////////

void VkwindowMainWindow::close ( Widget, XtPointer )
{
    // To close a window, just delete the object
    // If this is the last window, ViewKit will exit

    delete this;
}

void VkwindowMainWindow::copy ( Widget w, XtPointer callData )
{
    // This virtual function is called from copyCallback.
    // By default this member function passes control
    // to the component contained by this window

    _form->copy();
} // End VkwindowMainWindow::copy()

void VkwindowMainWindow::cut ( Widget w, XtPointer callData )
{
    // This virtual function is called from cutCallback.

```

```

    // By default this member function passes control
    // to the component contained by this window

    _form->cut();

} // End VkwindowMainWindow::cut()

void VkwindowMainWindow::newFile ( Widget w, XtPointer callData )
{
    // This virtual function is called from newFileCallback.
    // By default this member function passes control
    // to the component contained by this window

    _form->newFile();

} // End VkwindowMainWindow::newFile()

void VkwindowMainWindow::openFile ( Widget, XtPointer )
{
    // This virtual function is called from openFileCallback
    // Use the blocking mode of the file selection
    // dialog to get a file before calling view object's openFile()

    if(theFileSelectionDialog->postAndWait() == VkDialogManager::OK)
    {
        _form->openFile(theFileSelectionDialog->fileName());
    }
}

void VkwindowMainWindow::paste ( Widget w, XtPointer callData )
{
    // This virtual function is called from pasteCallback.
    // By default this member function passes control
    // to the component contained by this window

    _form->paste();

} // End VkwindowMainWindow::paste()

void VkwindowMainWindow::print ( Widget, XtPointer )
{
    // This virtual function is called from printCallback
    _form->print(NULL);

}

void VkwindowMainWindow::quit ( Widget, XtPointer )
{

```

```

    // Exit via quitYourself() to allow the application
    // to go through its normal shutdown routines, checking with
    // all windows, views, and so on.

    theApplication->quitYourself();

}

void VkwindowMainWindow::resetCB ( Widget w, XtPointer callData )
{
    // This virtual function is called from resetCBCallback.
    // By default this member function passes control
    // to the component contained by this window

    _form->resetCB( w, callData);

} // End VkwindowMainWindow::resetCB()

void VkwindowMainWindow::save ( Widget w, XtPointer callData )
{
    // This virtual function is called from saveCallback.
    // By default this member function passes control
    // to the component contained by this window

    // _form->save(); This is the original line here!

    // Zyda added code below.
    if(theFileSelectionDialog->postAndWait() == VkDialogManager::OK)
    {
        _form->save(theFileSelectionDialog->fileName());
    }

} // End VkwindowMainWindow::save()

void VkwindowMainWindow::saveas ( Widget, XtPointer )
{
    // This virtual function is called from saveasCallback
    // Use the blocking mode of the file selection
    // dialog to get a file before calling view object's save()

    if(theFileSelectionDialog->postAndWait() == VkDialogManager::OK)
    {
        _form->saveas(theFileSelectionDialog->fileName());
    }

}

//--- End generated member functions

```

How to Use the NPSImage Methods

```
//  
// This is file read_an_image.C  
//  
// This is a test program to show how to read an NPSImage.  
//  
  
#include <iostream.h>  
  
#include "NPSImage.h"  
  
void main()  
{  
  
    NPSImage myimage; // Define an image.  
  
    NPSImage mycopy; // Define a copy holder.  
  
    // Read an image from a file.  
    myimage.read_from("bud1.rgb");  
  
    // We have the image in memory.  
    // Print out the values associated with the image.  
    cout << "image size = " << myimage.size[0] << " " << myimage.size[1]  
        << " " << myimage.size[2] << "Name = " << myimage.name << endl;  
  
    // Write out the image.  
    myimage.write_to("bud1.copy");  
  
    // Copy the image.  
    mycopy = myimage;  
  
    // Write out the copy.  
    mycopy.write_to("bud1.again");  
  
    // Compare the 2 images.  
    if(myimage.is_equal(mycopy))  
    {  
        cout << "The images are equal! " << endl;  
    }  
    else  
    {  
        cout << "The images are different!" << endl;  
    }  
  
}
```

Available Image Display Tools

- Try xv, which has been installed on all IRISes.
- xv displays most image file formats.
- There are some other, older image support tools that I wrote under ~zyda/imagesupport.

Also Available Image Display Tools (Tools programmed in IRIS GL)

Usage: epslfromgif infile.gif 127 outfile.eps
-- the 127 value is the bwcutoff value.
-- the outfile.eps file is the output Encapsulated
Postscript made from your picture.

the program displays a gif image specified
on the command line as rgb color, as bw and bw-1bit.

the purpose of this program is to allow intelligent selection
of the cutoff value for the 1bit preview.

the eps output is generated from the 1-bit preview, instead
of the 8-bit image as in epsfromgif.

the mouse buttons exit the program

Usage: epslfromsgi infile.rgb 127 outfile.eps
-- the 127 value is the bwcutoff value.
-- the outfile.eps file is the output Encapsulated
Postscript made from your picture.

the program displays an sgi image specified
on the command line as rgb color, as bw and bw-1bit.

the purpose of this program is to allow intelligent selection
of the cutoff value for the 1bit preview.

the eps is generated from the 1-bit image, unlike program
epsfromsgi, which generates the eps from the 8-bit b/w image.

the mouse buttons exit the program

Usage: epsfromgif infile.gif 127 outfile.eps
-- the 127 value is the bwcutoff value.
-- the outfile.eps file is the output Encapsulated
Postscript made from your picture.

the program displays a gif image specified
on the command line as rgb color, as bw and bw-1bit.

the purpose of this program is to allow intelligent selection
of the cutoff value for the 1bit preview.

the mouse buttons exit the program

Usage: epsfromsgi infile.rgb 127 outfile.eps
-- the 127 value is the bw cutoff value.
-- the outfile.eps file is the output Encapsulated
Postscript made from your picture.

the program displays an sgi image specified
on the command line as rgb color, as bw and bw-1bit.

the purpose of this program is to allow intelligent selection
of the cutoff value for the 1bit preview.

the mouse buttons exit the program

Usage: gifttoeps file.gif outfile.eps

This program reads a gif image and generates Encapsulated Postscript.
Encapsulated Postscript is importable into Framemaker and Pagemaker.
The image input format expected is a gif file.

Usage: giftosgi file.gif file.rgb

This program reads in a gif file and outputs an sgi formatted
image file.

Usage: rletosgi file.rle file.rgb

This program reads in an rle file and outputs an sgi formatted
image file.

Usage: sgitoeps file.rgb outfile.eps

This program reads an SGI formatted RGB image taken on the IRIS and
generates Encapsulated Postscript.
Encapsulated Postscript is importable into Framemaker and Pagemaker.

Usage: sgitorle file.rgb file.rle

This program reads in an sgi file and outputs an rle formatted
image file.

Usage: showgif file.gif (understands wildcards)
the program displays gif images specified
on the command line.

the mouse buttons advance to the next image.

the esckey exits immediately.

Usage: showgif2 file.gif (understands wildcards)

the program displays gif images specified
on the command line as color mapped images.

the mouse buttons advance to the next image.

the esckey exits immediately.

Usage: showgifbw file.gif (understands wildcards)

the program displays gif images specified
on the command line as rgb color and as bw.

the mouse buttons advance to the next image.

the esckey exits immediately.

Usage: showrle file.rle (understands wildcards)

the program displays rle images specified
on the command line.

rle images are those output in Univ. of Utah rle format.

the mouse buttons advance to the next image.

the esckey exits immediately.

Usage: showsgi file.rgb (understands wildcards)

the program displays sgi images specified
on the command line.

the mouse buttons advance to the next image.

the esckey exits immediately.

Usage: showsgibw file.rgb (understands wildcards)

the program displays sgi images specified
on the command line as rgb color and as bw.

the mouse buttons advance to the next image.

the esckey exits immediately.

Usage: slidegif *.gif (understands wildcards)

the program displays gif images specified on the command line and advances each new image after a few seconds automatically.

the mouse buttons also advance to the next image.

the esckey exits immediately.

Usage: snapsgi file.rgb

This program uses the window manager to place a surround box around an area of the screen. The depression of the left mouse then places the 24 bit rgb image under that surround box into the specified file in sgi image format.

Usage: zoomgif file.gif

This program zooms a gif formatted image file.

Left mouse: Pan the image around.

Middle Mouse: + Zoom the image.

Right Mouse: - Zoom the image..

Usage: zoomrle file.rle

This program zooms an rle formatted image file.

Left mouse: Pan the image around.

Middle Mouse: + Zoom the image.

Right Mouse: - Zoom the image..

Usage: zoomsgi file.rgb

This program zooms an sgi formatted image file.

Left mouse: Pan the image around.

Middle Mouse: + Zoom the image.

Right Mouse: - Zoom the image..

Usage: shrinksgi infile.rgb factor outfile.rgb

This program shrinks the sgi formatted image file by the specified factor (try 2).