An OpenGL with wxWindows Tutorial

Andrew Van Pernis
Clemson University

A basic tutorial demonstrating how to use the wxWindows widget wxGLCanvas. The issues of basic event handling and cross platform development are discussed.

Key Words and Phrases: wxWindows, OpenGL, cross platform, graphical user interface

1. INTRODUCTION

OpenGL provides the programmer with a great deal of flexibility for drawing graphics to the screen, however a way to develop a strong graphical user interface is needed to support the power provided by OpenGL. Many OpenGL implementations support GLUT (OpenGL Utility Toolkit), which is a simple windowing API. GLUT lacks many of the features of more advanced GUI toolkits, and thus is not well suited for development of complex user interfaces. A large number of GUI libraries and APIs exist, but most are designed for a single hardware platform making it difficult to write programs that can be used on any system. Finally, several GUI APIs have been developed to be fully featured and cross platform. One of those is wxWindows.

wxWindows consists of two parts an API and a set of libraries. The wxWindows API defines a set of GUI widgets and utility functions in C++. The libraries are implementations of that API for a specific platform using a native framework. For example, wxWindows has three libraries available for Unix-based systems. The first and most commonly used is wxGTK, which runs under X windows using the GTK+ toolkit. wxMotif also runs under X, but uses the Motif toolset as its base. The third version of wxWindows for Unix, wxX11, is based on the standard X windows commands themselves. There are many other versions of wxWindows as well, including libraries for Microsoft Windows and MacOS. Furthermore, the API has been translated into other languages, such as Python, Perl, and BASIC. More information about wxWindows, as well as the libraries and API, can be found at the website, http://www.wxWindows.org.

In this paper, we will discuss a C++ program written using the wxWindows API that displays an OpenGL drawing area. Some of the issues involved with event handling will also be discussed. The program will also demonstrate the cross platform nature of wxWindows, so we will discuss how portions of the code must be developed to handle OpenGL on different platforms. Section 2 discusses some of the basic implementation details needed to write a program using the OpenGL and wxWindows APIs. Next, Section 3 covers the C++ class representing the application. The class defining the main window and its behavior is explored in

E-mail: arakel@vr.clemson.edu
Web: http://www.vr.clemson.edu/~arakel

Section 4. We will discuss the heart of the program, the OpenGL drawing region, in Section 5. Finally, Section 6 will give details about the OpenGL functions used for drawing by the application. The source code for the entire program is given in the appendices.

2. IMPLEMENTATION BASICS

In order to create a program with wxWindows, header directives will have to be added to the program to specify which portions of the wxWindows API will be used. All wxWindows programs have a basic set of files that need to be included. These files can be seen in Listing 1. In order to use OpenGL, the include directives for that API are required as well. First, we must specify that we will be adding the wxWindows widget for OpenGL drawable regions by using the first line in Listing 2. The remaining lines in Listing 2 give the directives that should be used for including the OpenGL API. Because the header files are placed in different locations under different operating sytems, directives defined by wxWindows are used to determine the operating system and specify the correct location for the header files.

Listing 1. wxWindows include directives.

```
#include "wx/wxprec.h"
#ifdef __BORLANDC__
#pragma hdrstop
#endif

#ifndef WX_PRECOMP
#include "wx/wx.h"
#endif
```

Listing 2. OpenGL include directives.

```
#include "wx/glcanvas.h"
#ifdef __WXMAC__
#ifdef __DARWIN__
#include <OpenGL/gl.h>
#include <OpenGL/glu.h>
#else
#include <gl.h>
#include <glu.h>
#endif
#else
#include <GL/gl.h>
#include <GL/glu.h>
#endif
```

3. APPLICATION CLASS

The first step in creating a wxWindows program is to develop a class to represent the application. The default class `wxApp` is provided by the API, because some changes need to made to that class we will derive a new class `DemoApp` from `wxApp`.

3.1 Application Class Declaration

Listing 3 shows the declaration for the class `DemoApp`. Because `DemoApp` inherits from `wxApp`, the first thing to be done is to redefine `OnInit`. This method is used to initialize the main window for the application, which will be an object of the class `DemoFrame`, as discussed in Section 4. Therefore, a pointer to an object of type `DemoFrame` is added as a member variable to the application class. Last, the macro, `DECLARE_APP`, is used to create a set of forward declarations needed by the wxWindows API.

Listing 3. `DemoApp` declaration block.

```
class DemoApp: public wxApp{
 public:
  bool OnInit(void);
 private:
  DemoFrame *m_main_window;
};
DECLARE_APP(DemoApp)
```

3.2 Application Class Implementation

The first part of the implementation of the class `OnInit` is the use of the macro `IMPLEMENT_APP(DemoApp)`. This macro is used as a compliment to the `DECLARE_APP` macro from the application class declaration.

The definition of the method `OnInit` can be seen in Listing 4. `OnInit` first attempts to create a new `DemoFrame` object. Because the `DemoFrame` object will be the main window for the application, if it could not be created the method returns a value of false, indicating the application should not continue. Once the main window has been created, a call to its method `Show` is used to display it. Then, `OnInit` returns a value of true indicating the main event loop for the application should begin.

4. MAIN WINDOW CLASS

wxWindows has a very robust event handling system, which we will use with the class representing the main window of the application. The main window will handle two events. The first is an event indicating that the program should exit. This event will occur whenever the window manager generates an exit event (i.e. the user clicked the close window button on the window border). The other event the main window will handle is the idle event, indicating that no other events have occurred and the application is idle.

Listing 4. Implementation of `DemoApp::OnInit`.

```
bool DemoApp::OnInit(void){

  //Create the main window
  m_main_window = new DemoFrame(NULL, "wxWindows OpenGL Demo",
                                wxDefaultPosition,
                                wxSize(720,480));
  if (m_main_window == NULL)
    return FALSE;

  //Show the window
  m_main_window->Show(TRUE);
  return TRUE;
}
```

4.1 Main Window Class Declaration

Because the main window will contain an OpenGL drawing region, a pointer to an object of the class `DemoGLCanvas`, which is discussed in Section 5, is added to the `DemoFrame` class as can be seen in Listing 5. Notice the macro `DECLARE_EVENT_TABLE`, that macro is added to the protected portion of any class declaration that will be using the event handling capabilities of wxWindows. The declaration also specifies the prototypes for two event handling methods, `OnExit` and `OnIdle`. A constructor is prototyped for the class as well.

Listing 5. `DemoFrame` declaration block.

```
class DemoFrame: public wxFrame{
 public:
  DemoFrame(wxFrame *frame, const wxString& title,
            const wxPoint& pos = wxDefaultPosition,
            const wxSize& size = wxDefaultSize,
            long style = wxDEFAULT_FRAME_STYLE);
  void OnExit(wxCloseEvent& event);
  void OnIdle(wxIdleEvent& event);
 private:
  DemoGLCanvas *m_canvas;
 protected:
  DECLARE_EVENT_TABLE()
};
```

4.2 Main Window Class Implementation

The constructor for `DemoFrame` makes use of a constructor inherited from the class `wxFrame`, as a basis for initializing the main window. Next a `DemoGLCanvas` object is created. The size of the main window is set to 720×480, but the size of the `DemoGLCanvas` widget is not specified, rather it is set to fill the available space within

Listing 6. `DemoFrame` constructor implementation.

```
DemoFrame::DemoFrame(wxFrame *parent, const wxString& title,
                     const wxPoint& pos, const wxSize& size,
                     long style):
  wxFrame(parent, -1, title, pos, size, style){
  int width, height;

  //Define attributes for the GL drawable
#ifdef __WXMSW__
  int *attributes = NULL;
#else
  int attributes[] = {WX_GL_RGBA, WX_GL_MIN_RED, 1,
                      WX_GL_MIN_GREEN, 1, WX_GL_MIN_BLUE, 1,
                      WX_GL_DEPTH_SIZE, 1, WX_GL_DOUBLEBUFFER,
#ifdef __WXMAC__
                      GL_NONE};
#else
                      None};
#endif
#endif

  //Create the GL drawable
  m_canvas = new DemoGLCanvas(this, -1, wxDefaultPosition,
                              wxDefaultSize, 0, "DemoGLCanvas",
                              attributes);
  m_canvas->GetClientSize(&width, &height);

  //Initialize OpenGL
  GL_Init(width, height);
}
```

the window. The OpenGL initialization routine needs to know the exact size of the drawable region, so a query method is used to obtain the size and pass that information to the initialization routine.

After the main window has been created, events need to be connected to the main window. In the declaration of `DemoFrame`, we specified two event handlers. A set of macros are used to define an event table, which will link the specific events to their handlers. Listing 7 shows the event table macros used for the `DemoFrame` class. The first macro, `BEGIN_EVENT_TABLE`, specifies the start of an event table for the class given as the first parameter, in this case `DemoFrame`. The second parameter is used to specify the parent of the first class, because the event table of the parent will be used to form a basis for the child's event table. The macros `EVT_CLOSE` and `EVT_IDLE` connect the event handling methods of `DemoFrame` to their events. Each event in wxWindows has a corresponding connection macro. The `END_EVENT_TABLE` macro simply declares the end of the current event table.

The exit event handler's implementation can be seen in Listing 8. The `Destroy` methods for both the OpenGL canvas widget and the main window are used to

Listing 7. Event table for the class `DemoFrame`.

```
BEGIN_EVENT_TABLE(DemoFrame, wxFrame)
  EVT_CLOSE(DemoFrame::OnExit)
  EVT_IDLE(DemoFrame::OnIdle)
END_EVENT_TABLE()
```

destroy them in order. The `Destroy` method for any wxWindows widget should be used in place of the `delete` operator, because wxWindows delays the deletion of some widgets until all events have been processed.

Listing 8. Implementation of `DemoFrame::OnExit`.

```
void DemoFrame::OnExit(wxCloseEvent& event){
  m_canvas->Destroy();
  Destroy();
}
```

Idle event handling is done with the method found in Listing 9. First, the idle event handler increments a global variable indicating the current frame number, which is useful for animation. Next, the `DemoGLCanvas` widget calls its method `Draw` to draw the scene using the updated frame number. The key to the idle event handler is the final step. When an event handling function finishes, wxWindows considers that event to be processed entirely. In order to continuously increment the frame number an idle event needs to occur whenever the application is idle. The call to the method `RequestMore` by the `wxIdleEvent` object passed as a parameter indicates that more idle event processing is needed. Thus, after the event handler has finished, an idle event will be generated almost immediately, creating a continuous stream of idle events. Other events will appear within this stream as they occur, so it may be necessary for an event with a lengthy handler to increment the current frame number in order to maintain a constant frame rate.

5. OPENGL DRAWABLE AREA CLASS

The OpenGL drawing area is the main focus of the application, and therefore the most complex class involved. Four event handlers are needed for this class. A method to render a scene into the OpenGL drawing region is also needed. Section 6 will discuss the function calls used for drawing.

5.1 OpenGL Drawable Area Class Declaration

Besides the event handling methods, the class `DemoGLCanvas` will have a basic constructor and a helping method to perform OpenGL drawing. No member variables are needed for the class, but the protected section must have an event table declaration using the `DECLARE_EVENT_TABLE` macro. The event table's construction can be seen in Listing 11. Listing 10 shows the complete declaration for the `DemoGLCanvas` class.

Listing 9. Implementation of `DemoFrame::OnIdle`.

```
void DemoFrame::OnIdle(wxIdleEvent& event){

  //Increment the time step
  time_step++;

  //Avoid overflowing the time step variable
  if (time_step >= MAX_STEPS)
    time_step = 0;

  //Redraw the OpenGL canvas
  m_canvas->Draw();

  //Request more idle events otherwise the main loop will
  //consider idle events to be processed until a non-idle
  //event is generated
  event.RequestMore();
}
```

Listing 10. `DemoGLCanvas` declaration block.

```
class DemoGLCanvas: public wxGLCanvas{
 public:
  DemoGLCanvas(wxWindow *parent, const wxWindowID = -1,
               const wxPoint& pos = wxDefaultPosition,
               const wxSize& size = wxDefaultSize,
               long style = 0,
               const wxString& name = "DemoGLCanvas",
               int *gl_attrib = NULL);

  //Event handling
  void OnPaint(wxPaintEvent& event);
  void OnSize(wxSizeEvent& event);
  void OnEraseBackground(wxEraseEvent& event);
  void OnChar(wxKeyEvent& event);

  //Helper functions
  void Draw(void);
 protected:
  DECLARE_EVENT_TABLE()
};
```

Listing 11. Event table for the class `DemoGLCanvas`.

```
BEGIN_EVENT_TABLE(DemoGLCanvas, wxGLCanvas)
  EVT_SIZE(DemoGLCanvas::OnSize)
  EVT_PAINT(DemoGLCanvas::OnPaint)
  EVT_CHAR(DemoGLCanvas::OnChar)
  EVT_ERASE_BACKGROUND(DemoGLCanvas::OnEraseBackground)
END_EVENT_TABLE()
```

5.2 OpenGL Drawable Area Class Implementation

As can be seen in Listing 12, the constructor starts by using a constructor inherited from it's parent, `wxGLCanvas`. This inherited constructor will use the attributes passed in as a parameter to create an OpenGL context for the widget. Once the constructor from `wxGLCanvas` is finished, the `DemoGLCanvas` constructor will continue by grabbing focus for the keyboard through the method `SetFocus`. By default, the main window would have keyboard focus, but as can be seen below, this widget will have the event handler for keyboard events. The last thing to be done by the constructor is to make the OpenGL context for the widget current, but this should not be done until the widget has been realized. In order to insure that the OpenGL drawing area has been realized, a call is made to its parent's `Show` method.

Listing 12. `DemoGLCanvas` constructor implementation.

```
DemoGLCanvas::DemoGLCanvas(wxWindow *parent, wxWindowID id,
                           const wxPoint& pos,
                           const wxSize& size, long style,
                           const wxString& name,
                           int *gl_attrib):
  wxGLCanvas(parent, id, pos, size, style, name, gl_attrib){

  //Set the canvas to receive keyboard input
  SetFocus();

  //Check to make sure the main window has been shown
  if (parent != NULL)
    parent->Show(TRUE);

  //Set the canvas to be the current GL drawable
  SetCurrent();
}
```

5.2.1 *Resize Event Handler.* Any time the OpenGL drawing region or the window containing it changes size, a size event will be generated, triggering the resize event handler. Listing 13 displays the implementation of that event handler. First, the event handler for size events from `wxGLCanvas` is used. This is another undocumented part of `wxGLCanvas`, which will handle changing the size of the widget. If

Listing 13. Implementation of `DemoGLCanvas::OnSize`.

```
void DemoGLCanvas::OnSize(wxSizeEvent& event){

  //Use the parent class size event handler
  wxGLCanvas::OnSize(event);

  //Check to see if there's a GL context for drawing
#ifndef __WXMOTIF__
  if (!GetContext())
    return;
#endif

  //Resize the GL viewport
  int width, height;
  GetClientSize(&width, &height);
  SetCurrent();
  glViewport(0, 0, width, height);

  //Adjust the perspective matrix
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(VIEW_ANGLE, (float)width/(float)height,
                 NEAR_CLIP, FAR_CLIP);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}
```

a context exists for the widget, the new size of the widget will be determined and used to resize the OpenGL viewport. After that, the perspective matrix is changed to reflect the new aspect ratio of the widget.

5.2.2 *Erase Background Event Handler.* Technically, a method for handling erase background events is not needed, because one is already defined for the parent class of `DemoGLCanvas`. Unfortunately, that event handler may cause the widget to flicker as the background is erased. Because the widget is an OpenGL drawing area, the flicker can be eliminated by having the erase background event handler do nothing and the paint event handler use `glClear` to erase the color buffer, which is equivalent to erasing the background.

Listing 14. Implemenation of `DemoGLCanvas::OnEraseBackground`.

```
void DemoGLCanvas::OnEraseBackground(wxEraseEvent& event){
  //Do nothing, but we need to define this to avoid flashing
  //caused by parent class's event handler for erase
  //background events
}
```

5.2.3 *Paint Event Handler.* The paint event handler is also quite simple. First the method creates a `wxPaintDC` object, which is a required step for all paint event handlers. In this case, the `wxPaintDC` object will not be used. Next, the paint event handler uses the method `Draw` to perform OpenGL rendering.

Listing 15. Implementation of `DemoGLCanvas::OnPaint`.

```
void DemoGLCanvas::OnPaint(wxPaintEvent& event){

  //This is a placeholder, but needs to be created as part of
  //the paint event handler
  wxPaintDC paint_context(this);

  //Draw using OpenGL
  Draw();
}
```

Before drawing to the OpenGL widget, it is important to check and make sure an OpenGL context exists. This can be done with the undocumented method `GetContext`, which returns a pointer to the `wxGLContext` associated with the drawing region. Note, that the class `wxGLContext` is also undocumented. Also, the `GetContext` method is currently not available in the Motif version of wxWindows. Once it is determined that a context is available for drawing, the `SetCurrent` method make that the current context for OpenGL commands. The function `GL_Draw` executes all of the actual rendering commands. Finally, because the OpenGL context was created using double buffering, a call to the method `SwapBuffers` moves the back buffer to the front and displays it. The source code for the helping method `Draw` can be seen in Listing 16.

Listing 16. Implementation of `DemoGLCanvas::Draw`.

```
void DemoGLCanvas::Draw(void){

  //Check to see if there's a GL context for drawing
#ifndef __WXMOTIF__
  if (!GetContext())
    return;
#endif

  //Use that context to draw
  SetCurrent();
  GL_Draw();
  SwapBuffers();
}
```

5.2.4 *Keyboard Input Event Handler.* Because the window manager may or may not add a close window button to the main window, a way to exit the application is needed. The key event handler will check to see if one the keys, escape, Q, or X, has been pressed. If any one of them has been pressed then the program should exit. To do so, the `Close` method needs to be called by the object representing the main window, which is retreived with the method, `GetParent`, as the main window is the parent of the OpenGL drawing region. If any other key is pressed the application should do nothing. A call to the `Skip` method for the `wxKeyEvent` object indicates that this event is not needed, and thus no other event handler should process it.

Listing 17. Implementation of `DemoGLCanvas::OnChar`.

```
void DemoGLCanvas::OnChar(wxKeyEvent& event){
  switch(event.KeyCode()) {
  case WXK_ESCAPE:
  case 'q': case 'Q':
  case 'x': case 'X':
    GetParent()->Close();
    break;
  default:
    event.Skip();
    return;
  }
}
```

6. OPENGL DRAWING FUNCTIONS

Two functions are used for OpenGL drawing. The first, `GL_Init`, is a basic initialization routine. As can be seen in Listing 18, first the function sets several OpenGL state variables. Next, the perspective matrix is set using the width and height parameters passed into the function. Finally, the modelview matrix is set to the identity.

The second OpenGL drawing function, `GL_Draw`, performs the rendering of a simple animated scene. The function begins by setting up a camera for viewing the scene. Then a rectangle is drawn so that it ripples using a sine wave. The rippling effect is animated through the use of the time step variable discussed in 4.2. Listing 19 shows the details of the rendering function. It is important to note that a value for Pi is used in the computation of the sine wave function, while some system math libraries include a definition of constant representing Pi, others do not. The preprocessor statements shown in Listing 20 check to see if a Pi constant exists, creating it if it does not.

7. CONCLUSION

It is easy to create a basic OpenGL application using wxWindows. Furthermore, wxWindows provides the advantages of being cross platform and extensible. Finally, unlike GLUT, wxWindows is a fully functional GUI toolkit, making it a good choice for developing with OpenGL.

Listing 18. OpenGL initialization routine.

```
void GL_Init(int width, int height){

  //Basic GL initialization
  glClearColor(0.2, 0.2, 0.2, 0);
  glEnable(GL_DEPTH_TEST);
  glEnable(GL_NORMALIZE);

  //Enable GL optimizations
  glEnable(GL_CULL_FACE);

  //Initialize the GL matrices
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(VIEW_ANGLE, (float)width/(float)height,
                 NEAR_CLIP, FAR_CLIP);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}
```

REFERENCES

Braem, F. 2001-2002. *wxWindows 2: Programming cross-platform GUI applications in C++*. http://users.skynet.be/saw/download/wxWindows/wxTutorial.pdf.

Shreiner, D., Ed. 2000. *OpenGL Reference Manual*, Third ed. Addison-Wesley. The Official Reference Document to OpenGL, Version 1.2.

Woo, M., Neider, J., Davis, T., and Shreiner, D. 1999. *OpenGL Programming Guide*, Third ed. Addison-Wesley. The Official Guide to Learning OpenGL, Version 1.2.

Listing 19. OpenGL rendering function.

```
void GL_Draw(void){
  float height, x;
  int i;
  glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

  //Set the camera
  glPushMatrix();
  gluLookAt(5.0, 10.0, 5.0, 5.0, 0.0, 0.0, 0.0, 1.0, 0.0);

  //Transform the quad
  glPushMatrix();
  glTranslatef(5.0, 0.0, 0.0);
  glScalef(8.0, 2.0, 6.0);
  glTranslatef(-0.5, 0.0, 0.5);

  //Draw the rippling quad
  glColor4f(1.0, 1.0, 1.0, 1.0);
  glBegin(GL_QUAD_STRIP);
  for (i=0; i<=RIPPLE_STRIPS; i++){
    height = sin(4*M_PI*(((time_step+i)%RIPPLE_STRIPS)/
                         (float)RIPPLE_STRIPS));
    x = (float)i/(float)RIPPLE_STRIPS;
    if (x > 1.0)
      x = 1.0;
    glTexCoord2f(x, 1.0);
    glVertex3f(x, height*0.2, -1.0);
    glTexCoord2f(x, 0.0);
    glVertex3f(x, height*0.2, 0.0);
  }
  glEnd();
  glPopMatrix();

  //Pop the camera matrix (technically I don't have to do
  //this, but I might want to add flight controls one day)
  glPopMatrix();
}
```

Listing 20. Cross platform Pi definition.

```
#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif
```

APPENDIX

A. WXWINDOWS_DEMO.H

```
//File:         wxWindows_demo.h
//Author:       Andrew Van Pernis <arakel@vr.clemson.edu>

#ifndef UBER_WXWINDOWS_DEMO_H
#define UBER_WXWINDOWS_DEMO_H 1

//OpenGL drawable region class
class DemoGLCanvas: public wxGLCanvas{
 public:
  DemoGLCanvas(wxWindow *parent, const wxWindowID = -1,
               const wxPoint& pos = wxDefaultPosition,
               const wxSize& size = wxDefaultSize,
               long style = 0,
               const wxString& name = "DemoGLCanvas",
               int *gl_attrib = NULL);

  //Event handling
  void OnPaint(wxPaintEvent& event);
  void OnSize(wxSizeEvent& event);
  void OnEraseBackground(wxEraseEvent& event);
  void OnChar(wxKeyEvent& event);

  //Helper functions
  void Draw(void);
 protected:
  DECLARE_EVENT_TABLE()
};

//Main application window class
class DemoFrame: public wxFrame{
 public:
  DemoFrame(wxFrame *parent, const wxString& title,
            const wxPoint& pos = wxDefaultPosition,
            const wxSize& size = wxDefaultSize,
            long style = wxDEFAULT_FRAME_STYLE);

  //Event handling
  void OnExit(wxCloseEvent& event);
  void OnIdle(wxIdleEvent& event);
 private:
  DemoGLCanvas *m_canvas;
 protected:
  DECLARE_EVENT_TABLE()
};

//Application class
class DemoApp: public wxApp{
```

```
48  public:
49   bool OnInit(void);
50  private:
51   DemoFrame *m_main_window;
52 };
53 DECLARE_APP(DemoApp)
54
55 #endif
```

B. WXWINDOWS_DEMO.CPP

```
1  //File:         wxWindows_demo.cpp
2  //Author:       Andrew Van Pernis <arakel@vr.clemson.edu>
3
4  //wxWindows includes
5  #include "wx/wxprec.h"
6  #ifdef __BORLANDC__
7  #pragma hdrstop
8  #endif
9
10 #ifndef WX_PRECOMP
11 #include "wx/wx.h"
12 #endif
13
14 #if !wxUSE_GLCANVAS
15 #error Please enable OpenGL support for wxWindows.
16 #endif
17
18 //OpenGL includes
19 #include "wx/glcanvas.h"
20 #ifdef __WXMAC__
21 #ifdef __DARWIN__
22 #include <OpenGL/gl.h>
23 #include <OpenGL/glu.h>
24 #else
25 #include <gl.h>
26 #include <glu.h>
27 #endif
28 #else
29 #include <GL/gl.h>
30 #include <GL/glu.h>
31 #endif
32
33 //Application includes
34 #include "wxWindows_demo.h"
35 #include "draw.h"
36 #include <iostream>
37
38 //Application class definitions
39 IMPLEMENT_APP(DemoApp)
40
```

```
41 bool DemoApp::OnInit(void){
42
43   //Create the main window
44   m_main_window = new DemoFrame(NULL, "wxWindows OpenGL Demo",
45                                 wxDefaultPosition,
46                                 wxSize(720,480));
47   if (m_main_window == NULL)
48     return FALSE;
49
50   //Show the window
51   m_main_window->Show(TRUE);
52   return TRUE;
53 }
54
55 //Main window class definitions
56 BEGIN_EVENT_TABLE(DemoFrame, wxFrame)
57   EVT_CLOSE(DemoFrame::OnExit)
58   EVT_IDLE(DemoFrame::OnIdle)
59 END_EVENT_TABLE()
60
61 DemoFrame::DemoFrame(wxFrame *parent, const wxString& title,
62                      const wxPoint& pos, const wxSize& size,
63                      long style):
64   wxFrame(parent, -1, title, pos, size, style){
65   int width, height;
66
67   //Define attributes for the GL drawable
68 #ifdef __WXMSW__
69   int *attributes = NULL;
70 #else
71   int attributes[] = {WX_GL_RGBA, WX_GL_MIN_RED, 1,
72                       WX_GL_MIN_GREEN, 1, WX_GL_MIN_BLUE, 1,
73                       WX_GL_DEPTH_SIZE, 1, WX_GL_DOUBLEBUFFER,
74 #ifdef __WXMAC__
75                       GL_NONE};
76 #else
77                       None};
78 #endif
79 #endif
80
81   //Create the GL drawable
82   m_canvas = new DemoGLCanvas(this, -1, wxDefaultPosition,
83                               wxDefaultSize, 0, "DemoGLCanvas",
84                               attributes);
85   m_canvas->GetClientSize(&width, &height);
86
87   //Initialize OpenGL
88   GL_Init(width, height);
89 }
90
91 void DemoFrame::OnExit(wxCloseEvent& event){
```

```
92    m_canvas->Destroy();
93    Destroy();
94  }
95
96  void DemoFrame::OnIdle(wxIdleEvent& event){
97
98    //Increment the time step
99    time_step++;
100
101   //Avoid overflowing the time step variable
102   if (time_step >= MAX_STEPS)
103     time_step = 0;
104
105   //Redraw the OpenGL canvas
106   m_canvas->Draw();
107
108   //Request more idle events otherwise the main loop will
109   //consider idle events to be processed until a non-idle
110   //event is generated
111   event.RequestMore();
112 }
113
114
115 //OpenGL drawing region definitions
116 BEGIN_EVENT_TABLE(DemoGLCanvas, wxGLCanvas)
117   EVT_SIZE(DemoGLCanvas::OnSize)
118   EVT_PAINT(DemoGLCanvas::OnPaint)
119   EVT_CHAR(DemoGLCanvas::OnChar)
120   EVT_ERASE_BACKGROUND(DemoGLCanvas::OnEraseBackground)
121 END_EVENT_TABLE()
122
123 DemoGLCanvas::DemoGLCanvas(wxWindow *parent, wxWindowID id,
124                            const wxPoint& pos,
125                            const wxSize& size, long style,
126                            const wxString& name,
127                            int *gl_attrib):
128   wxGLCanvas(parent, id, pos, size, style, name, gl_attrib){
129
130   //Set the canvas to receive keyboard input
131   SetFocus();
132
133   //Check to make sure the main window has been shown
134   if (parent != NULL)
135     parent->Show(TRUE);
136
137   //Set the canvas to be the current GL drawable
138   SetCurrent();
139 }
140
141 void DemoGLCanvas::OnPaint(wxPaintEvent& event){
142
```

```
143   //This is a placeholder, but needs to be created as part of
144   //the paint event handler
145   wxPaintDC paint_context(this);
146 
147   //Draw using OpenGL
148   Draw();
149 }
150 
151 void DemoGLCanvas::OnSize(wxSizeEvent& event){
152 
153   //Use the parent class size event handler
154   wxGLCanvas::OnSize(event);
155 
156   //Check to see if there's a GL context for drawing
157 #ifndef __WXMOTIF__
158   if (!GetContext())
159     return;
160 #endif
161 
162   //Resize the GL viewport
163   int width, height;
164   GetClientSize(&width, &height);
165   SetCurrent();
166   glViewport(0, 0, width, height);
167 
168   //Adjust the perspective matrix
169   glMatrixMode(GL_PROJECTION);
170   glLoadIdentity();
171   gluPerspective(VIEW_ANGLE, (float)width/(float)height,
172                  NEAR_CLIP, FAR_CLIP);
173   glMatrixMode(GL_MODELVIEW);
174   glLoadIdentity();
175 }
176 
177 void DemoGLCanvas::OnEraseBackground(wxEraseEvent& event){
178   //Do nothing, but we need to define this to avoid flashing
179   //caused by parent class's event handler for erase
180   //background events
181 }
182 
183 void DemoGLCanvas::OnChar(wxKeyEvent& event){
184   switch(event.KeyCode()) {
185   case WXK_ESCAPE:
186   case 'q': case 'Q':
187   case 'x': case 'X':
188     GetParent()->Close();
189     break;
190   default:
191     event.Skip();
192     return;
193   }
```

```
}

void DemoGLCanvas::Draw(void){

  //Check to see if there's a GL context for drawing
#ifndef __WXMOTIF__
  if (!GetContext())
    return;
#endif

  //Use that context to draw
  SetCurrent();
  GL_Draw();
  SwapBuffers();
}
```

C. DRAW.H

```
//File:         draw.h
//Author:       Andrew Van Pernis <arakel@vr.clemson.edu>

#ifndef UBER_WXWINDOWS_DEMO_DRAW_H
#define UBER_WXWINDOWS_DEMO_DRAW_H 1

//Constants
#define VIEW_ANGLE 45.0
#define NEAR_CLIP 0.1
#define FAR_CLIP 100.0
#define RIPPLE_STRIPS 360
#define MAX_STEPS (RIPPLE_STRIPS*16)

//Globals
extern int time_step;

//Prototypes for standard OpenGL functions
void GL_Init(int width, int height);
void GL_Draw(void);

#endif
```

D. DRAW.CPP

```
//File:         draw.cpp
//Author:       Andrew Van Pernis <arakel@vr.clemson.edu>

//OpenGL includes
#include "wx/glcanvas.h"
#ifdef __WXMAC__
#ifdef __DARWIN__
#include <OpenGL/gl.h>
#include <OpenGL/glu.h>
#else
```

```
#include <gl.h>
#include <glu.h>
#endif
#else
#include <GL/gl.h>
#include <GL/glu.h>
#endif

//Application includes
#include <math.h>
#include "draw.h"

//Constants
#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif

//Global variables
int time_step = 0;

//Functions
void GL_Init(int width, int height){

  //Basic GL initialization
  glClearColor(0.2, 0.2, 0.2, 0);
  glEnable(GL_DEPTH_TEST);
  glEnable(GL_NORMALIZE);

  //Enable GL optimizations
  glEnable(GL_CULL_FACE);

  //Initialize the GL matrices
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(VIEW_ANGLE, (float)width/(float)height,
                 NEAR_CLIP, FAR_CLIP);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}

void GL_Draw(void){
  float height, x;
  int i;
  glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

  //Set the camera
  glPushMatrix();
  gluLookAt(5.0, 10.0, 5.0, 5.0, 0.0, 0.0, 0.0, 1.0, 0.0);

  //Transform the quad
  glPushMatrix();
```

```
62   glTranslatef(5.0, 0.0, 0.0);
63   glScalef(8.0, 2.0, 6.0);
64   glTranslatef(-0.5, 0.0, 0.5);
65
66   //Draw the rippling quad
67   glColor4f(1.0, 1.0, 1.0, 1.0);
68   glBegin(GL_QUAD_STRIP);
69   for (i=0; i<=RIPPLE_STRIPS; i++){
70     height = sin(4*M_PI*(((time_step+i)%RIPPLE_STRIPS)/
71                          (float)RIPPLE_STRIPS));
72     x = (float)i/(float)RIPPLE_STRIPS;
73     if (x > 1.0)
74       x = 1.0;
75     glTexCoord2f(x, 1.0);
76     glVertex3f(x, height*0.2, -1.0);
77     glTexCoord2f(x, 0.0);
78     glVertex3f(x, height*0.2, 0.0);
79   }
80   glEnd();
81   glPopMatrix();
82
83   //Pop the camera matrix (technically I don't have to do
84   //this, but I might want to add flight controls one day)
85   glPopMatrix();
86 }
```