Introduction to OpenGL

- A basic library of functions for specifying 2-D and 3-D graphics primitives, attributes, transformations, viewing setups, and many other operations
  - All functions in OpenGL library begin with gl
- Designed to be hardware independent
  - All functions in OpenGL library are device independent
  - So many operations (windowing, I/O, etc.) not included in basic library
  - Many auxiliary libraries for these

Related Libraries

- GLU: utility library provides routines for working with viewing/projection matrices, approximating complex 3D objects with polygons, displaying quadrics & splines, surface rendering, and much more
  - GLU functions begin with glu
  - All OpenGL implementations include the GLU library

Windowing Support Libraries

- Windowing systems are platform dependent
- Support libraries:
  - GLX: OpenGL Extension to the X Window System, functions begin with glX
  - WGL: Microsoft Windows-to-OpenGL interface, functions begin with wgl
  - AGL: Apple GL, functions begin with agl
  - GLUT: OpenGL Utility Toolkit
    - A library of functions for interacting with screen-windowing system, functions begin with glut
    - Works with many different platforms

OpenGL for Microsoft Windows

- Industry standard for high-quality 3-D graphics applications
- Available on many HW and OS platforms
- "Thin" software interface to underlying graphics HW
- Implementing on Windows brings workstation-class graphics to PC
- Real 3-D graphics for Windows

Using OpenGL from Microsoft Windows

- Two approaches:
  - WGL
    - Underlying Windows functionality does most of the work
    - Easy to use from either Win32 API or MFC
  - GLUT
    - Contains functions to create and manage windows
    - Others to set up handler functions for user-initiated events
    - Harder to program, but applications more easily ported to other platforms
Steps in Using OpenGL in Windows Applications – WGL Approach

- Get a DC for rendering location (window)
- Choose & set a “pixel format” for the DC
- Create a Rendering Context (RC) for the DC
- Associate (bind) the RC with the DC
- Draw using OpenGL function calls
- Release the RC & DC

Rendering Context (RC)

- OpenGL equivalent of Windows GDI DC
- Mechanism by which OpenGL calls are rendered to the device
- Links OpenGL calls to a Windows window client area
- Must be compatible with window’s DC
- Keeps track of current values of OpenGL state variables
  - Just like DC does for GDI state variables
    - Attributes, drawing objects, etc.

Pixel Format

- The translation layer between OpenGL calls and the rendering operation Windows performs
- Holds attributes for device drawing surface
- Describes things like:
  - Using single or double buffering
  - Direct or indirect color
  - Drawing to a window or a bitmap
  - Color depth (# of bit planes)
  - ZBuffer depth
  - Lots of others

PIXELFORMATDESCRIPTOR

- Data structure used to set the Pixel Format
- Some fields:
  - dwFlags: “OR” of properties constants, e.g.
    - doublebuffered, stereo, window or bitmap, etc.
  - iPixelFormat
    - color type (RGBA or indexed)
  - cColorBits: # of bitplanes
  - cRedBits: # of bits in red color channel
  - cRedShift: where red bits are
  - etc.
- See online help: PIXELFORMATDESCRIPTOR

Choosing and Setting the Pixel Format

- Set up a PIXELFORMATDESCRIPTOR variable (e.g., pfd)
- ChoosePixelFormat(hDC, &pfd)
  - gets DC’s pixel format that’s the closest match to the PFD desired
  - returns an integer pf_index
- SetPixelFormat(hDC, pf_index, &pfd)
  - Set that pixel format in the DC

Other Pixel Format Descriptor Functions

- DescribePixelFormat()
  - Specify index & DC, returns pointer to filled PFD structure describing the PF
- GetPixelFormat()
  - Gets index of current PFD for the specified DC
Creating and using an RC

- Use WGL function to create an RC:
  - wglCreateContext(hDC);
  - Returns a handle to an OpenGL Rendering Context:
    - HGLRC hRC
- Make the RC “Current” (bind RC to DC)
  - wglMakeCurrent(hDC, hRC);
  - Binds the RC to the current thread of execution
- Now we can draw with OpenGL calls

Cleanup

- Make RC non-current (Unbind RC from DC)
  - wglMakeCurrent(hDC, NULL);
- Get rid of the DC
  - ReleaseDC() or EndPaint() in API app.
    - Done automatically in MFC when OnDraw() returns
- Get rid of the RC
  - wglDeleteContext(hRC);

Building a Windows/OpenGL App using the WGL Interface

- Includes in .h file:
  - <gl/gl.h>     // OpenGL interface
  - <gl/glu.h>   // OpenGL utility library interface
- Must add opengl32.lib & glu32.lib to Linker’s Object library modules
  - Under .NET:
    - Type in: opengl32.lib glu.lib

MINOGL Example Program

- Displays a rectangle in different shades of red
- See online listing of view class of minogl example OpenGL program
  - Look on CS-460/560 “Sample Programs” Page
  - Link:
    - MINOGL: A Simple OpenGL Example Program for Windows MFC (minoglView.cpp)

Using the GLUT in OpenGL Windows Win32 API Apps

- Download the Windows version from:
- Copy files to following directories:
  - Glut32.dll to: Windows\system32
  - glut32.lib to: Program Files\Microsoft Visual Studio.NET\VC7\PlatformSDK\lib
  - glut.h to: Program Files\Microsoft Visual Studio.NET\VC7\PlatformSDK\include\gl
Create a Win32 API Application (Empty)

- Under Project Properties:
  - Linker / Input / Additional Dependencies, add:
    - opengl32.lib glu32.lib glut32.lib
  - Under Linker / Advanced / Entry Point, set to:
    - mainCRTStartup

Header Files

- #include <GL/glut.h>
  - gl.h and glu.h not needed if we’re using the GLUT
  - May need other C/C++ standard header files:
    - stdio.h, stdlib.h, math.h, etc.

Main Program

- Just like regular C/C++ app, entry point is:
  - void main(int &argc, char** argv)

  - In main() do following:
    1. Initialize the GLUT with
      - glutInit(&argc, argv);
    2. Set the display mode
      - Specify kind of buffering, color mode, etc:
        - glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    3. Set initial window position on screen:
      - glutInitWindowPosition(x,y);
    4. Set initial window size on screen
      - glutInitWindowSize(w,h);
    5. Create the window:
      - glutCreateWindow("title");

Further Initialization: Background Color

- Set background color for display window
  - glClearColor(1.0,1.0,1.0,0.0); //white
    - Assigns a color, but does not paint it, Use:
      - glClear(GL_COLOR_BUFFER_BIT);
        - Causes values in color buffer to be set to values given in glClearColor()

More Initialization: Projection Type, Viewing Transformation, Clipping

- OpenGL designed for 3D graphics
- Must project onto 2D window
- Also do window to viewport transformation
  - with clipping
- For 2D graphics, use an orthogonal projection
  - gluOrtho2D(xmin,xmax,ymin,ymax)
    - Equivalent to taking z=0 & setting a "window" with clipping boundaries: xmin<=x<=xmax ymin<=y<=ymax
      - Will be mapped to entire client area of physical window
    - Since projection transformations are done with matrices, must first set the matrix mode and initialize the matrix:
      - glMatrixMode(GL_PROJECTION);

After Initialization

- Must specify what is to be displayed in the display window
  - Create the picture in a "callback" function using OpenGL drawing functions
  - Pass the address of that callback function to the GLUT routine glutDisplayFunction(callback_fnt);
  - Subsequently callback_fnt gets called any time client area of display window is exposed
    - Like MFC OnDraw() is called in response to WM_PAINT messages
- Finally start the message loop:
  - glutMainLoop();
    - Must be last statement in program
Example GLUT Windows Application

- See Section 2-9 of the text book (Hearn and Baker)
- Program listing on page 80
  - Just draws a red straight line