Components of a Graphics Software System

Introduction to OpenGL

Basic Components of a Graphics Software System

* Examples from: Windows GDI and OpenGL
1. Output Primitives
- Building blocks for drawing pictures
- **Plotting a pixel**—most primitive
- **Windows CDC:**
  
  ```c
  COLORREF colref;
  SetPixel(x,y,colref); // Windows--plots pixel
  colref = GetPixel(x,y); // returns pixel color
  ```
- **OpenGL:**
  
  ```c
  glBegin (GL_POINTS); // OpenGL
  glVertex2f (x, y); // 2D, f==>floating pt
  glEnd(); // current drawing is color used
  ```
  - In general:
    ```c
    glVertex{234}{sifd} (TYPE coords,...);
    glVertexv{234}{sifd} (TYPE *coord array);
    // glPointSize(size); before Begin/End to set size in pixels
    ```

**Lines**

- **Windows CDC:**
  
  ```c
  MoveTo(x1,y1); // Set Curr. Pos., one endpoint
  LineTo(x2,y2); // line from CP to (x2,y2)
  // current pen is used
  ```
- **OpenGL:**
  
  ```c
  glBegin (GL_LINES); // OpenGL
  glVertex2f(x1,y1); // 2D endpoint vertices
  glVertex2f(x2,y2); // appear in pairs
  glEnd() // current glLineWidth & glColor
  ```
Polylines and Polygons

– Windows CDC:
  Polyline(lppts,num_pts); // Windows
  Polygon(lppts,num_pts);
  // parameters: POINT array, number of points

– OpenGL:
  glBegin (GL_POLYGON); // OpenGL
  glVertex2f(x1,y1); // first polygon vertex
  glVertex2f(x2,y2); // second polygon vertex
  ...            // more vertices
  glEnd();       // current glColor & glPolygonMode are used

Other primitives

– Windows CDC:
  • Lots of other primitives
  • See prior notes on Windows programming
    – Especially Help on CDC class

– OpenGL:
  • GL_TRIANGLES, GL_TRIANGLE_STRIP,
    GL_TRIANGLE_FAN, GL_LINE_STRIP,
    GL_QUADS, etc. ---- lots more
Text

Windows CDC:
    TextOut(x,y,lpszStr,cStrLength);

OpenGL:
    – Design a font set using bitmap functions in the core library
    – Use GLUT character-generation library functions
        char* str = "abcde";
        glRasterPos2i(10,10);
        for (int k=0; k<5; k++)
            glutBitmapCharacter(GLUT_BITMAP_9_BY_15, str[k]);

3-D primitives

Windows has nothing

OpenGL:
    – GLU graphics library
        • sphere, cube, cone, etc.
2. Attributes (State Variables)

- Properties of primitives
  - how they appear
  - e.g., color, line style, text style, fill patterns

- Usually modal
  - values retained until changed

- Windows –
  - see prior notes (e.g., pens, brushes)

- OpenGL -- \texttt{glProperty()};
  - ‘Property’ is state variable to set, e.g.
    \begin{verbatim}
    glColor3f (1.0, 0.0, 0.0); // bright red
    glLineWidth(3.0); // 3 pixels wide
    glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
    \end{verbatim}

3. Transformations

- Done with matrix math
- Setting windows/viewports
  - Window-to-viewport transformation

- Moving objects
  - Geometric Transformations
  - e.g., translation, rotation, scaling

- Changing coordinate system
- Changing viewpoint
- Different types of projections
Windows
- window-to-viewport transformation
  - done with Mapping Modes
- programmer must implement others

OpenGL is very rich
- glLoadMatrix(), glRotatef(), glTranslatef(), glScalef(), glViewport(), glFrustum(), glOrtho2D(), gluPerspective(), etc.

4. Segmentation
- Dividing scene into component parts for (later) manipulation
- Windows: GDI strictly immediate mode
  - But there are Metafiles (can be played back)
- OpenGL has Display lists:
  - Groups of OpenGL commands that have been stored for later execution
  - Can be hierarchical
- PHIGS uses hierarchical segments
5. Input/Interaction

- Obtain data from input devices or graphics system
  - So user can manipulate scene interactively
- Windows:
  - Built into event-driven, message-based paradigm

5. Input/Interaction in OpenGL

- Obtain data from input devices/system and respond to events
- Auxiliary libraries (GLX, WGL, AGL)
  - All use the underlying windowing system
- Or GLUT callback functions
  - All take pointers to an event handler function, e.g.
    • Window must be redrawn: glutDisplayFunc (mydisplay)
      - Then write: mydisplay ( ) function;
    • Keyboard: glutKeyboardFunc (mykey)
      - Then write: mykey (key, xmouse, ymouse) function
    • Mouse events: glutMouseFunc (mymouse)
      - Then write: mymouuse (button, action, xmouse, ymouse)
    • Mouse motion: glutMotionFunc (mymotion)
      - Then write: mymotion (xmouse, ymouse)
6. Control/Housekeeping

- Initialize system, create window, etc.
- Windows: Extensive support
  - RegisterClass(), CreateWindow(), etc.
  - Mostly hidden in MFC framework
- OpenGL:
  - Use GLUT library functions
    - glutInit(&argc,argv); glutInitDisplayMode(mode);
    - glutInitWindowSize(w,h); glutInitWindowPosition(x,y);
    - glutCreateWindow("Title"); glutMainLoop();
  - Or use WGL functions under Windows

7. Storing/retrieving/manipulating bitmapped Images

- BitBLT -- Bit Block Transfer
- Windows:
  - Device Dependent Bitmaps
    - BitBlt(), StretchBlt(), StretchDIBits() etc.
  - But very slow
  - Device Independent Bitmaps--faster
  - DirectX-- flipping surfaces--fastest!
- OpenGL:
  - glReadPixels(); glDrawPixels(); glCopyPixels();
8. Rendering/Photorealism

- Hidden surfaces, lighting, shading, reflection properties, etc.
- Windows GDI: Very little support
  - DirectX (Direct3D)--Quite a bit of support
- OpenGL: A lot of support!
  - e.g., light sources, lighting models, material properties, blending, antialiasing, fog, depth buffer (hidden surface removal), texturing, etc.

Introduction to OpenGL
The OpenGL API

- A basic library of functions for specifying 2-D and 3-D graphics primitives, attributes, transformations, viewing setups, and many other operations
- Hardware and platform independent
  - All functions in OpenGL library are device independent
  - So many operations (windowing, I/O, etc.) are not included in basic core library
  - Many auxiliary libraries for these
- Close enough to hardware so that programs written in OpenGL run efficiently
- Easy to learn and use

Three Views of OpenGL

- Programmer’s view
  - Specify a set of output primitives to render
  - Describe properties (attributes) of these objects
  - Define how these objects should be viewed
- OpenGL state machine with functions to:
  - Specify inputs to state machine
  - Change the state of the machine
  - Both determine the machine’s outputs
- The OpenGL Pipeline
The OpenGL Pipeline

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

Related Libraries
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
    - Comes with Microsoft Visual Studio
  - 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
    - Doesn’t come with Visual Studio, but easily obtained

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
  - Implies very good performance
- 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
  - Can be used from either Win32 API or MFC

- **GLUT**
  - Contains functions to create and manage windows
  - Others to set up handler functions for user-initiated events
  - Applications more easily ported to other platforms
  - Win32 API

Using the **GLUT** in OpenGL Windows Applications

**-Visual Studio 2005-**

- Download the Windows version from:

- Copy files to following directories:
  - glut32.dll to: Windows\system32
  - glut32.lib to: Program Files\Microsoft Visual Studio 8\VC\PlatformSDK\lib
  - glut.h to: Program Files\Microsoft Visual Studio 8\VC\PlatformSDK\include\gl
Using GLUT from VS 2008

- Download the GLUT libraries and header files
- Put them in the correct directories
- Go to the following website:
  - http://tempvariable.blogspot.com/2008/02/installing-freeglut-on-visual-studio.html
- Follow the instructions given there to download the files and copy them to the indicated directories:

Creating a GLUT-based Win32 API Application

- Create a Win32 API Application (Empty)
- Under Project Properties:
  - Configuration 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, time.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”);
Setting the 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 a 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);`
      - `glLoadIdentity();`
After Initialization

- Specify what to display in display window
  - Create the picture in a display “callback” function using OpenGL drawing functions
  - Pass the address of that callback function to the GLUT routine `glutDisplayFunc(callback_ftn);`
  - Subsequently `callback_ftn` gets called any time client area of display window is exposed
    - Like MFC `OnDraw()`, `callback_ftn` is called in response to `WM_PAINT` messages
    - Place code there that specifies what is to be displayed
    - End with `glFlush()` to force buffered commands to execute

- Finally start the message loop in `main()`:
  - `glutMainLoop();`
  - Must be last statement in `main()`
Example GLUT Windows Application

- See Section 2-9 of the text book (Hearn and Baker)
- Modified Program listing on page 80
  - See First OpenGL program using GLUT (ogl-pgm1-cc) link on “Example Programs” web page
  - Just draws two diagonal red straight lines
  - And some text

Using OpenGL with Microsoft Windows: WGL Approach
Steps in Using OpenGL in Windows Applications – **WGL Approach**

- Get a DC for a rendering location (window)
- Choose & set a “pixel format” for the DC
  - Describes desired HW capabilities
- Create a Rendering Context (RC) for the DC
  - Links OpenGL calls to the DC associated with a window client area
- Associate (bind) the RC with the DC
- Draw using OpenGL function calls
- Release the RC & DC

**Rendering Context (RC)**

- OpenGL equivalent of Windows GDI Device Context
- Mechanism by which OpenGL calls are rendered to the device via a DC
- Links OpenGL calls to a window client area through the associated DC
  - RC Must be compatible with a 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
Translation layer between OpenGL ftn. calls & Windows physical rendering operation
Describe things like:
– If using single or double buffering
– If direct or indirect color
– If drawing to a window or offscreen 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
– cDepthBits: depth of Z-buffer (hidden surface removal)
– etc.
☞ See online help: PIXELFORMATDESCRIPTOR
Choosing and Setting the Pixel Format

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

Creating and Using a Rendering Context

- Use WGL function to create an RC:
  - `hRC = wglCreateContext(hDC);`
  - Returns a handle to an OpenGL Rendering Context:
    - `HGLRC hRC`
  - Will have all capabilities of selected pfd
- Make the RC “Current” [bind RC to DC]
  - `wglMakeCurrent(hDC, hRC);`
  - Binds the RC to the window’s DC and 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()` in a Win32 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
  • Note we’re not using the GLUT

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 CView 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)