Binghamton University

EngiNet™

State University of New York

Thomas J. Watson

School of Engineering
and Applied Science

EngiNet™

WARNING

CS 560

Computer Graphics

Professor Richard Eckert

Lecture # 6

February 7, 2001

Lecture 5

Wednesday, 2-7-01

Computer Graphics Software

1. Lowest Level (earliest)-- Assembly/machine language

- **Programs drive hardware directly**
  - **Fast, but non-portable**
  - **Difficult to program**
  - **Prone to errors**

2. Medium Level (General Programming Packages)

- A. Extensions to high level languages-- graphics libraries
  - **e.g., Borland's BGI, Windows' GDI, Silicon Graphics GL, Microsoft's DirectX**
  - **Still have platform dependencies**
  - **Easier to program**
  - **Usually slower, but with optimized compilers, not so bad**

- B. Standard Graphics Packages
  - **Sets of specifications**
  - **Supposedly language/platform independent**
  - **Usually with bindings for many high level languages**
  - **Syntax for accessing graphics functions**
  - **Examples: GKS, PHIGS, OpenGL**

3. Special-purpose Application packages

- **e.g., Corel Draw, 3D Studio, Harvard Graphics, Photoshop**
- **Good for what they do, but specific uses**

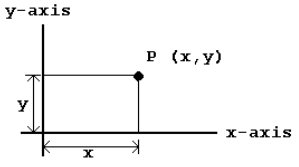
We'll be working at level 2 for most of this course

Describing positions of objects

- Need coordinate systems
- 2-D and 3-D Cartesian systems used universally

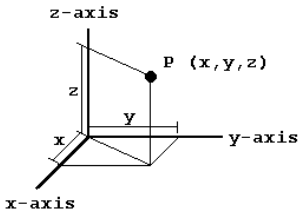
2-D Cartesian System

- Measure distance along two mutually perpendicular axes
- Position given by 2 numbers (x,y)



3-D Cartesian System

- Measure distance along three mutually perpendicular axes
- Position given by three numbers (x,y,z)



Other commonly-used systems

- Depend on symmetry
  - Spherical (rho, theta, phi)
  - Cylindrical (r, theta, z)
- Conversion formulas used

Types of coordinate systems

1. Modeling coordinate system (MCS)

- System used by programmer (modeler) to describe a single object
- Can be 2-D or 3-D
- Depends on object being modeled
- Origin, scale picked according to object
- Varies from object to object

2. World Coordinate System (WCS)

- Reference system used to position objects in a scene
- Can be 2-D or 3-D
- Origin, scale, units specific to scene
- Coordinate transformations map from an MCS to the WCS
  - Effectively positions objects in scene
  - Called the "Modeling Transformation"

3. Device Coordinate System (DCS)

- Coordinate system used by output device
- Units usually pixels
- Always 2-D
- Varies according to HW platform
- Graphics SW maps from WCS to DCS
  - Called the "Viewing Transformation"
- If WCS is 3-D, a projection transformation is also involved

4. Normalized Device Coordinates (NDCS)

- 2-D system
- 0 <= x <= 1; 0<= y <= 1
- Intermediate between WCS and DCS
- Hardware-independent

Graphics Transformation Pipeline

- MCS ------> WCS ------> NDCS ------> DCS
  modeling xform viewing xform HW-depend. xform
- The first two transformations are device independent

A Window

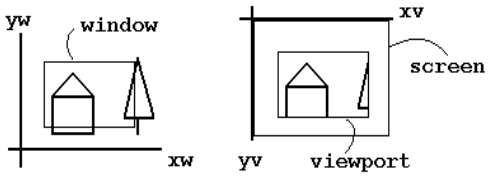
- A rectangular area (2-D)
  or Rectangular parallelepiped (3-D)
- Expressed in WC
  - e.g.,
    xwmin,xwmax,ywmin,ywmin,zwmin,zwmax
- Specifies a part of the scene of interest

A Viewport

- A rectangular portion of the screen
- Expressed in device coordinates
  - e.g., xvmin, yvmin, xvmax, yvmax
- A window is mapped to a viewport

Window to Viewport Transformation

– mapping from a window to a viewport



Clipping

- Removal of parts of scene outside a window or viewport
- Graphics system may perform clipping with respect to a window or a viewport

Animation Techniques using windows/viewports/clipping

- Zooming
  - Change window size from frame to frame, clip, and transform
    - scene appears to approach or recede from viewer
- Panning
  - Translate window from frame to frame, clip, and transform
    - scene appears to move across the screen

BASIC COMPONENTS OF A GRAPHICS SOFTWARE SYSTEM

- Examples from: Windows & OpenGL

1. Output Primitives

- Building blocks for drawing pictures
- Plotting a pixel--most primitive

```
SetPixel(x,y,colref); // Windows--plots pixel
colref = GetPixel(x,y); // returns pixel  color

glBegin (GL_POINTS);  // OpenGL
   glVertex2f (x, y);  // 2==>2D, f==>floating pt
glEnd();   // current drawing color used
```

- Lines

```
Windows:
MoveTo(x,y);      // Set Curr. Pos.
LineTo(x,y);      // line from CP to (x,y)

glBegin (GL_LINES);  //OpenGL
   glVertex2f(x1,y1);  //endpoint vertices
   glVertex2f(x2,y2);  //appear in pairs
glEnd()
```

- Polylines and Polygons

```
Polyline(lppts,num_pts); // Windows--use pts
                         // array, number of points
Polygon(lppts,num_pts);

glBegin (GL_POLYGON); // OpenGL
   glVertex2f(x1,y1); // polygon vertex
   …                  // more vertices
glEnd();
```

- Other primitives
  - **Windows**
    - **Lots of other primitives**
    - **See prior notes on Windows programming**
  - **OpenGL**
    - **GL_TRIANGLES, GL_LINE_STRIP, GL_QUADS, etc. -- lots more**

- Text

Windows:
TextOut(x,y,lpszStr,cStrLngth);

OpenGL-- Use display lists and/or GLUT library functions

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**
- **OpenGL -- gl*Property*();**
  - **'*Property*' is state variable to set**
    - **e.g., glColor3f (R,G,B);**

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(), glOrtho(), gluPerspective(), etc.

4. Segmentation

- Dividing scene into component parts for (later) manipulation
- Windows--strictly immediate mode
  - (DirectX has support for retained mode)
- 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

Input/Interaction in OpenGL

- Auxiliary libraries (aux, GLX, GLUT)
  - Use underlying windowing system
  - Provide input-handling callback ftns
    - e.g., auxMouseFunc()
    - glutMouseFunc()
  - take pointer to callback function
    - programmer must write these

6. Control/Housekeeping

- Initialize graphics system, put window up, etc.
- Windows--Extensive support
  - RegisterClass(), CreateWindow(), etc.
- OpenGL
  - Use of auxiliary libraries
    - e.g., auxInitWindow(), auxInitDisplayMode(), auxMainLoop(ptr to ftn. that draws scene)

7. Storing/retrieving/ manipulating bitmapped Images

- BitBlT--Bit Block Transfer
- Windows--
  - Device Dependent Bitmaps
    - BitBlt(), StretchBlt(), etc.
  - But very slow
  - Device Independent Bitmaps--faster
  - DirectX-- flipping surfaces--fastest!
- OpenGL--
  - glReadPixels(); glDrawPixels(); glCopyPixels();

8. Photorealism

- Hidden surfaces, lighting, shading, reflection properties, etc.
- Windows--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, etc.

Intro. to OpenGL for 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 to Windows

Steps in using OpenGL in Windows Applications

- Get a DC for rendering location (window)
- Choose & select a "pixel format" for DC
- Create a Rendering Context (RC) for 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
- Must be compatible with window's DC
- Keeps track of current values of OpenGL state variables

Pixel Format

- 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
  - Others

PIXELFORMATDESCRIPTOR

- Data structure used to set Pixel Format
- Some fields:
  - dwFlags: "OR" of properties constants, e.g.
    - doublebuffered, stereo, window or bitmap, etc.
  - iPixelType
    - color type (RGBA or indexed)
  - cColorBits: # of bitplanes
  - cRedBits: # of red
  - cRedShift: where red bits are
  - etc.

Choosing and setting the Pixel Format

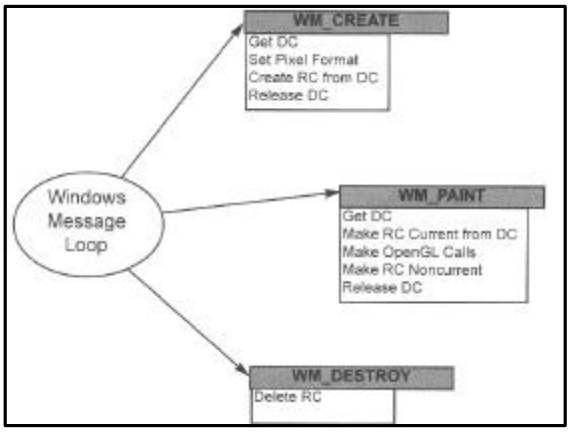
- Set up a PIXELFORMATDESCRIPTOR variable (e.g., pfd)
- ChoosePixelFormat(hDC,&pfd)
  - gets DC's pixel format closest to one desired
  - returns an integer pf_index
- SetPixelFormat(hDC, pf_index, &pfd)
  - Set that pixel format in the DC

Creating and using an RC

- Use "wgl" function to create an RC:
  - wglCreateContext(hDC)
  - Returns a handle to an OpenGL Rendering Context (HGLRC)
- Make the RC "Current" (bind RC to DC)
  - wglMakeCurrent(hDC, hRC)
- Now we can draw with OpenGL calls

Cleanup

- Make RC noncurrent (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

- 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'
  - 'Project'|'Settings'|'Link tab'

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)