Introduction to OpenGL Part II

CS 351-50

OpenGL: Open Graphics Library

- Graphics API
  (Application Programming Interface)
  - Software library
  - Layer between programmer and graphics hardware (and other software
- Several hundred procedures and functions

What is OpenGL

- Configurable state machine
  - Input is 2D or 3D data
  - Output is framebuffer
  - Modify state to modify functionality

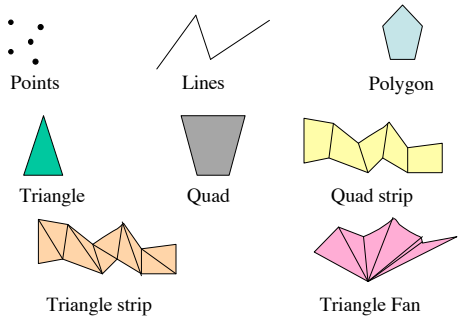
How do I render a geometric primitive?

- To Framebuffer
- OpenGL primitives
  - A group of one or more vertices
  - Vertex defines:
    - A point
    - An endpoint of an edge
    - A corner of a polygon where two edges meet

OpenGL Rendering

- Data consists of
  - Positional coordinates
  - Colors
  - Normals
  - Texture Coordinates
- Each vertex is processed
  - independely
  - In order
  - In the same way

OpenGL Primitives



OpenGL drawing

- To draw a primitive, call glBegin()
- glEnd() encloses a list of vertices and their attributes
- Coordinates of a primitive are given counter-clockwise order

Function calls to draw a primitive

```
glBegin(GL_POINTS);
    glVertex3f(0.0f, 0.0f, 0.0f);
glEnd();
```

Draw a triangle:

```
glBegin(GL_TRIANGLES);
        glVertex3f(0.0f, 1.0f, 0.0f);
        glVertex3f(-1.0f, -1.0f, 0.0f);
        glVertex3f(1.0f, -1.0f, 0.0f);
glEnd();
```

Draws a triangle with different colors at each vertex

```
glBegin(GL_TRIANGLES);
        glColor3f(1.0f, 0.0f, 0.0f); //pure red
        glVertex3f(0.0f, 1.0f, 0.0f);

        glColor3f(0.0f, 1.0f, 0.0f); //pure green
        glVertex3f(-1.0f, -1.0f, 0.0f);

        glColor3f(0.0f, 0.0f, 1.0f); //pure blue
        glVertex3f(1.0f, -1.0f, 0.0f);
glEnd();
```

Types of rendering

- Wireframe
  - Display only lines and curves
  - No filled objects
- Flat Shading
  - Compute a single color for each polygon
  - Fill the polygon with that constant color

Types of rendering

- Smooth (Gouraud shading)
  - Interpolate vertex colors across polygon
  - Vertex colors can be specified explicitly or computed using lighting calculations
- Texture Mapping
  - Modify color of each pixel with colors from an image
- 1991 Pixar Shutterbug Example
  - http://www.siggraph.org/education/curriculum/projects/slide_sets/slides91/91_01_2.htm

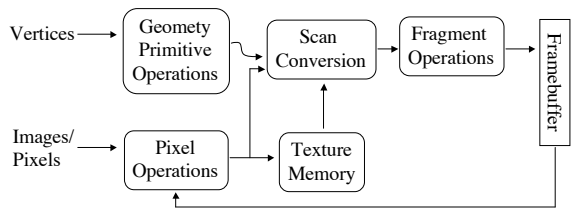
How are these options configured?

- Functions
  - glEnable, glDisable,
  - glCullMode, glPolygonMode, glLightModel, etc.

What is the OpenGL Pipeline

- Process thru which OpenGL processes data
- Geometric data and pixel data processed separately
  - Geometry subject to per-vertex operations
  - Pixel data subject to per-pix operations
  - Both share the same final step
    - Rasterization and per-fragement ops
    - (fragment is general term for pixel or vertex)

OpenGL Pipeline



OpenGL Pipeline

- Understanding Pipeline is important!
  - Helps design algorithms
  - Helps find bottlenecks
    - Which parts are supported by hardware

Pixel Primitives

- Provide a way of manipulating rectangles of pixels
- glDrawPixels, glReadPixels, glCopyPixels move pixel rectangles to and from the framebuffer
- glBitmap takes a binary image and renders the current color in framebuffer positions corresponding to 1s in image (useful for drawing fonts)
- glRasterPos defines where pixels go in the framebuffer

Hidden Surface Removal

- When we draw a fragment, record the z (distance to the eye) in the depth buffer
- If the z stored in the depth buffer is greater than the z for the fragment about to be drawn, draw it
- Otherwise, the fragment is behind something that has already been drawn, so throw it away

Hidden Surface Removal

- When seeing up your window, specify a depth buffer:
  - glutInitDisplayMode(GLUT_DEPTH);
- When clearing, make sure to:
  - glClear(GL_DEPTH_BUFFER_BIT);
- glEnable(GL_DEPTH_TEST);
- Set the depth test comparison operation:
  - glDepthFunc(GL_LESS);
  - //this is the default, don't really need to specify

Demos

- Shapes (Nate Robin's Tutors)
  - Ctrl and mouse to change primitive

OpenGL Color Models

- RGBA color
  - Red, Green, Blue, Alpha
  - Separate channel for each
  - 8 bits/channel = 16 million colors
- Indexed Color
  - Small number of colors accessed by indices into a color look up table
    - 8 bits = 256 colors

OpenGL Transparency

- Use the fourth component (alpha) of RGBA color
  - Alpha = 0 is fully transparent
  - Alpha = 1 is fully opaque
- Objects are composited as they are rendered
  - Example: C = alpha*Cs + (1 - alpha) Cf
  - Cs is the color of the incoming fragment
  - Cf is the color already in the framebuffer

Viewing in OpenGL

- Viewing consists of two parts
  - Object positioning
    - model view transformation matrix
  - View projection
    - projection transformation matrix
  - OpenGL support both perspective and orthographic viewing transformations
  - OpenGL camera is always at the origin, pointing in the -z direction
  - Transformations move objects relative to camera

ModelView Matrix

- Positions objects in world coordinates
- Usually formed by concatenating simple transformations
  - glRotate(theta, x,y,z)
  - glTranslate(x,y,z)
  - glScale(x,y,z)
- Order is important

Modeling Transformations

```
glTranslatef(0.0f, 0.0f, -10.0f);
glRotatef(45.0f, 0.0f, 1.0f, 0.0f);

glBegin(GL_TRIANGLES);
glColor3f(1.0f, 0.0f, 0.0f); //pure red
glVertex3f(0.0f, 1.0f, 0.0f);

glColor3f(0.0f, 1.0f, 0.0f); //pure green
glVertex3f(-1.0f, -1.0f, 0.0f);

glColor3f(0.0f, 0.0f, 1.0f); //pure blue
glVertex3f(1.0f, -1.0f, 0.0f);
glEnd();
```

Viewing in OpenGL

- Orthographic projection
  - Boundaries of the parallel viewing volume
  - Objects are clipped to specified viewing cube
  - glOrtho(left, right, bottom, top, front, back)
- Perspective Projection
  - glFrustum, gluPerspective
  - Clipping volume is a frustum
- Make sure near and far clipping planes aren't too far apart
- Make sure near plane isn't too close to eye

Positioning the Camera

- Use gluLookAt to specify
  - Eye location
  - "Look-at" point
  - "up" vector
- gluLookAt(10,10,10,1,2,3,0,0,1);
  - Eye is (10,10,10)
  - Look at point is (1,2,3)
  - Up is (0,0,1)
- Usually done in GL_PROJECTION matrix and combined with perspective matrix

Complete Viewing Example

```
//Projection first
glMatrixMode( GL_PROJECCTION);
glLoadIdentity();
gluPerspective(60, 1, 1, 100);
gluLookAt(10,10,10,1,2,3,0,0,1)

//Now object transformations
glMatrixMode(GL_MODELVIEW)
glLoad Identity();
glTranslate(1,1,1);
glRotatef(90, 1,0,0);
DrawObject();
```

Matrix Stacks

- OpenGL has multiple matrix "stacks"
- glPushMatrix pushes a copy of the top fo the stack matrix
- glPopMatrix throws away the top of the stack
- Very useful for hierachically defined figures
- Demo:
  - http://www.cs.princeton.edu/~min/cs426/jar/transform.html

Column versus row order

- OpenGL uses column, C uses row
- We will use column

From OpenGL maual (pg 103 of Programmers 1.1)

- "If you are programming in C and declare a matrix as m[4][4] then the element m[*i*][*j*] is in the *i*th column and *j*th row of the OpenGL transformation matrix. This is the reverse of the standard C convention in which m[*i*][*j*] is in row *i* and column *j*. To avoid confusion you should declare your matrices as m[16]."

OpenGL Matrix:

- (zero based)

$$\begin{pmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{pmatrix}$$

Demos

- Transformations (Nate Robin's Tutors)
  - The transformation tutorial program (shown at left) demonstrates how the basic transformations of rotate, translate and scale operate in OpenGL. The order of the transforms can be changed to see how that effects rendering.

Slide Credits

- These slides were pieced together based upon information in the following valuable sources:
  - Greg Humphreys, 2000 (pdf of Intro to OpenGL, CS148 Lecture 7)
  - http://www.csie.nctu.edu.tw/~jllin/D_Documents.htm
  - Michael Mack,
    - http://www.acm.wwu.edu/~mackm/pres/index.php