OpenGL Programming
What is OpenGL?

- A low level graphics API for 2D and 3D interactive graphics. OS independent.
- Descendent of GL (from SGI)
OpenGL

- Contains a library of over 200 functions
- Portable
  - Implementations available for nearly all hardware and operating systems
  - Input or windowing are *not* included in OpenGL
    - Options for Windows: GLUT, FLTK, or MFC
    - GLUT = OpenGL Utility Toolkit
    - Implementations of GLUT exist for most computing environment
- Controlled by the OpenGL Architecture Review Board
  - SGI, IBM, nVidia, ATI, … -- some major players in CG
What OpenGL isn’t:

- A windowing program or input driver, since those couldn’t be OS independent.

**Diagram:**

- **OpenGL application program**
- **GL**
- **GLU**
- **GLUT**
- **GLX**
- **Xlib, Xtk**
- **Frame buffer**

GL: core graphics capability
GLU: utilities on top of GL
GLUT: input and windowing functions
GLU

- OpenGL Utility – GLU
  - All OpenGL implementations include GLU
  - Functions are with prefix *glu*
  - Setting up viewing and projection matrices
  - Describing complex surfaces using line and polygon approximations
  - Displaying quadrics and B-spline surfaces using linear approximations
GLUT

- OpenGL Utility Toolkit – GLUT
  - Provides a library functions for interacting with any screen-window system
  - Functions are with prefix `glut`
  - The source code are available at: [http://www.opengl.org/resources/libraries/glut/](http://www.opengl.org/resources/libraries/glut/)

Installation:
1. Put header file (.h) in the GL/ include directory of Visual Studio
2. Put library file (.lib) in the lib directory
3. Put dynamic link library (.dll) in the system directory
Computer Graphics Pipeline

Geometric Model

OpenGL Rendering Algorithm

Rendered Image
The Visualization Problem
The Visualization Problem
The Visualization Problem

Need to change coordinate systems!

Camera Coordinate System (CCS)

Object Coordinate System (OCS)

World Coordinate System (WCS)
How does OpenGL work?

From the programmer’s point of view:

- Specify geometric objects
- Describe object properties
- Define how they should be viewed
- Move camera or objects around for animation
How does OpenGL work?

- State machine with input and output
  - State variables: color, current viewing position, line width, material properties,…
  - The variables (the state) then apply to every subsequent drawing command
  - Input is description of geometric object
  - Output is pixels sent to the display
How does OpenGL work?

- OpenGL pipeline:

  Model → Transformation → Projection → Rasterization → Display
How does OpenGL work?

- OpenGL pipeline:

  - Model → **Transformation** → Projection → Rasterization → Display

  - Map from WCS to CCS
  - Culling
  - Lighting
How does OpenGL work?

OpenGL pipeline:

- Model
- Transformation
- Projection
- Rasterization
- Display

- Map from WCS to CCS
- Culling
- Lighting
- Map from CCS to Screen CS
- Clipping
- Persp. division
How does OpenGL work?

- OpenGL pipeline:

  - Model
  - Transformation
  - Projection
  - Rasterization
  - Display

  - Map from WCS to CCS
  - Culling
  - Lighting
  - Map from CCS to Screen CS
  - Clipping
  - Persp. division
  - Color
  - Visibility
How does OpenGL work?

- **OpenGL pipeline:**
  - **Model** → **Transformation** → **Projection** → **Rasterization** → **Display**
  - Map from WCS to CCS
  - Culling
  - Lighting
  - Map from CCS to Screen CS
  - Clipping
  - Persp. division
  - Color
  - Visibility
  - Frame Buffer Display
How does OpenGL work?

OpenGL pipeline:

- Model
- Transformation
- Projection
- Rasterization
- Display
Basic OpenGL Syntax

- **Prefix:** `gl`

  `glBegin, glEnd, glClear, glPolygonMode, ...`

- **Constant symbols:** `GL_`

  `GL_RGB, GL_POLYGON, GL_LINES, ...`

- **Data types:** `GL`

  `GLbyte, GLshort, GLint, GLfloat, GLdouble, ...`
OpenGL Commands: a Quick Look

- Just function calls:
  \texttt{glColor3f(1.0, 1.0, 1.0);}  
  - GL prefix
  - Command name
  - Type suffix (if variable), can also end with “\texttt{v}”
  - Number of arguments (if variable)

- Same command, different arguments:
  \texttt{glColor3ub(255,255,255);}  
  -- same result
**OpenGL Data Types**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>8</td>
<td>GLbyte, GLubyte</td>
</tr>
<tr>
<td>s</td>
<td>16</td>
<td>GLshort</td>
</tr>
<tr>
<td>i</td>
<td>32</td>
<td>GLint, GLsizei, GLint</td>
</tr>
<tr>
<td>f</td>
<td>32</td>
<td>GLfloat, GLclampf</td>
</tr>
<tr>
<td>d</td>
<td>64</td>
<td>GLdouble, GLclampd</td>
</tr>
<tr>
<td>ub</td>
<td>8</td>
<td>GLubyte, Glboolean</td>
</tr>
<tr>
<td>us</td>
<td>16</td>
<td>GLushort</td>
</tr>
<tr>
<td>ui</td>
<td>32</td>
<td>GLuint, GLenum, GLbytefield</td>
</tr>
</tbody>
</table>
OpenGL as a State Machine

- **OpenGL has a state**
  - And associated state variables (line width, color, … etc.)
    - eg. GL_FLAT & GL_SMOOTH are two states of GL_SHADE_MODEL
  - Some states are binary states that are either GL_FALSE or GL_TRUE
  - Each state variable has a default value.

- **Changing states**
  - For binary state variables, use glEnable/glDisable (eg. GL_LIGHTING)
  - Mode state variables require specific commands (eg. glShadeModel)
  - Value state variables require specific commands too. (eg. glColor3f)

- **Remains in effect until changed!**
State Queries

- **Checking if enabled: `glIsEnabled(GLenum cap)`**
  - Cap: symbolic constant indicating an OpenGL capability (e.g., `GL_DEPTH_TEST`)
  - Returns `Glboolean`

- **Getting state variable value**
  - `glGetIntegerv(pname, params)`
  - Stores value of `pname` at location `params`
  - Similarly, `glGetBooleanv`, `glGetFloatv`, etc.

- **More specific queries for some important variables**
  - `glGetString(GL_VERSION)`
  - `glGetLight()`
  - `glGetError()`
Primitives: drawing a polygon

- Put GL into draw-polygon state:
  \[ \text{glBegin (GL\_POLYGON);} \]
- Send it the points making up the polygon:
  \[ \text{glVertex2f (x0, y0);} \]
  \[ \text{glVertex2f (x1, y1);} \]
  \[ \text{glVertex2f (x2, y2);} ... \]
- Tell it we’re finished
  \[ \text{glEnd();} \]
Specifying Primitives

- Code for all of the examples are available from: http://www.xmission.com/~nate/tutors.html
OpenGL Primitives

- Geometric object is described by the type of the primitive to be drawn and a set of vertices.
  - GL_POINTS
  - GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP
  - GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN
  - GL_QUADS, GL_QUAD_STRIP
  - GL_POLYGON
OpenGL Primitives

- **Why triangles, quads, and strips?**

  ![Diagram showing triangle strips and quad strips]

  - Hardware may be more efficient for triangles
  - Strips require processing less data
    - fewer glVertex calls
Primitives: Light Material Properties

- Ambient: same at every point on the surface
- Diffuse: scattered light independent of angle (rough)
- Specular: dependent on angle (shiny)
Primitives: Light Sources

- Point light sources are common:
Transformation

- glTranslate
- glRotate
- glScale
- glPushMatrix(); glPopMatrix();
Camera Views

Different views of an object in the world
Camera Views

- Lines from each point on the image are drawn through the center of the camera lens (the center of projection – COP)
Camera Views

- Many camera parameters...
- For a physical camera
  - Position
  - Orientation
  - Lens (field of view)
Camera Projections

- Orthographic projection
  - Long telephoto lens
- Flat but preserving distances and shapes. All the projectors are parallel.

- `glOrtho(left, right, bottom, top, near, far);`
Camera Projections

- Perspective projection
- Example: pin hole camera
- Objects farther away are smaller in size
Camera Transformations

- Camera positioning just results in more transformations on the objects:
  - Transformations that position the object relative to the camera
- Example:
  - `void gluLookAt (eyex, eyey, eyez, centerx, center y, centerz, upx, upy, upz);`
Clipping

- Not everything is visible on the screen
**Rasterizer**

- Transform pixel values in the projected coordinates to pixel values in screen (raster) coordinates -- `glViewport(x, y, width, height)`