CMSC 498M: Chapter 2a
Overview of OpenGL

Reading:
- An excellent resource is the *OpenGL Programming Guide*, (also known as "The Red Book") by the OpenGL Arch. Rev. Board, Shreiner, Woo, Neider, and Davis.

Overview:
- Graphics Pipeline and General Structure
- OpenGL Drawing Primitives
- OpenGL Rendering Process

The Graphics Pipeline

Geometry Processing
- Model (geometric description)
- Transform (eye/camera space)
- Clip (viewing volume)
- Light (illumination and shading)

Image Processing
- Scan (primitives to pixels)
- Visibility (hidden surface removal)
- Texture (texture map)
- Composite (masks, other images)
- Frame Buffer (final image)

Display
OpenGL

Standard: Most widely-used/supported 2D/3D graphics API
- Windows NT/95/98/00, UNIX, Linux, MacOS, OS/2, Python, ...
- Bindings for C, C++, Java, Fortran, Ada
- ATI, HP/Compaq, E&S,IBM, Intel, Intergraph, NVIDIA, Microsoft, SGI

Independent: of hardware, OS, window system.

Windowing not included: Does not include commands for windowing tasks or user interaction.

OpenGL Resources

Online Documentation:
http://www.opengl.org/documentation/
http://msdn.microsoft.com/library/

Tricks/Tips/Examples/Tutorials:
http://nehe.gamedev.net/
http://www.gamedev.net/

Sample Code:
http://www.opengl.org/code/

GLUT: (GL toolkit)
http://www.opengl.org/resources/libraries/glut/
http://www.xmission.com/~nate/glut.html
Display List:
OpenGL drawing commands, pre-compiled for efficiency.

Evaluator:
Vertex preprocessing.

Per-Vertex Operations:
Geometric transformations applied to each vertex.

Primitive Assembly:
Vertices are grouped together to form triangles, polygons, etc.

Rasterization:
Primitives converted into pixels, called fragments, and colored.

Per-Fragment Operations:
Tests (e.g. depth/stencil test) to determine fragment visibility.

Pixel Operations and Texture Memory:
Pixels can be copied, texture mapped, or saved.
OpenGL Drawing Primitives

OpenGL Type Naming Conventions

Functions: begin with gl
Constants: begin with GL_
Types: begin with GL

<table>
<thead>
<tr>
<th>Suffix</th>
<th>OpenGL Datatype</th>
<th>C/C++ Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>GLbyte</td>
<td>signed char</td>
</tr>
<tr>
<td></td>
<td>GLshort</td>
<td>short</td>
</tr>
<tr>
<td></td>
<td>GLint</td>
<td>int</td>
</tr>
<tr>
<td>s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>i</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GLubyte</td>
<td>unsigned char</td>
</tr>
<tr>
<td></td>
<td>GLushort</td>
<td>unsigned short</td>
</tr>
<tr>
<td></td>
<td>GLuint</td>
<td>unsigned int</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ub</td>
<td></td>
<td></td>
</tr>
<tr>
<td>us</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GLLint</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>GLfloat</td>
<td>float</td>
</tr>
<tr>
<td>d</td>
<td>GLdouble</td>
<td>double</td>
</tr>
</tbody>
</table>

Chapter 2, Slide 8
Copyright © David Mount and Amitabh Varshney
Specifying Vertices

**Vertex Arguments:** All objects in OpenGL are constructed from convex polygons, which are represented by their vertex coordinates. The argument type is specified by the suffix to the OpenGL function name:

```
<func_name> <dim> <type> (argument list )
```

**Examples:**
- 2D point in GLint (int) coordinates:
  ```
glVertex2i (200, -150);
  ```
- 3D point in GLfloat (float) coordinates:
  ```
glVertex3f (200.3f, -150f, 40.75f);
  ```

**Vector (array) Arguments:** Add suffix "v" to the function name

- 3D point in GLdouble (double) coordinates given as a vector:
  ```
  GLdouble pt[3] = { 200.3, -150, 40.75 };
  glVertex3dv (pt);
  ```

OpenGL Vertices and Primitives

OpenGL supports a number of different drawing primitives. Each primitive is specified by enumerating the vertices that define it.

**General Form:**
```
glBegin (object type);
  glVertex... ( ... );
  glVertex... ( ... );
  glVertex... ( ... );
glEnd ();
```

**Note:** There are a number of other attributes that can be placed within the glBegin...glEnd pair. These affect things like color, texture, and surface normals. We will discuss these later.

**Vertices:** May be 2D (x,y), 3D (x,y,z), or 4D (x,y,z,w), where the w coordinate is usually 1. Called homogeneous coordinates.
**Isolated Points**

**GL_POINTS:** Draws a set of isolated points.

```c
glBegin ( GL_POINTS );
    glVertex2i ( 0, 0 );
    glVertex2i ( 0, 1 );
    glVertex2i ( 1, 0 );
    glVertex2i ( 1, 1 );
glEnd ( );
```

(0,0) (1,0) (0,1) (1,1)

---

**Line Loop (Polyline)**

**GL_LINE_LOOP:** Draws a closed polygonal line (segments joined end to end).

```c
glBegin ( GL_LINE_LOOP );
    glVertex2i ( 0, 0 );
    glVertex2i ( 0, 1 );
    glVertex2i ( 1, 1 );
    glVertex2i ( 1, 0 );
glEnd ( );
```

Variants:
- **GL_LINE_STRIP:** Polygonal line, but not closed off to form a loop.
- **GL_LINES:** A sequence of line segments, not connected to each other.
Polygon

**GL_POLYGON:** Draws a filled convex polygon.

```c
glBegin ( GL_POLYGON );
    glVertex2i ( 0, 0 );
    glVertex2i ( 0, 1 );
    glVertex2i ( 1, 1 );
    glVertex2i ( 1, 0 );
glEnd();
```

**Note:** OpenGL assumes that all polygons are convex, meaning that all interior angles are at most 180 degrees. OpenGL is largely "silent" about errors, so be careful.

---

Triangles

**GL_TRIANGLES:** Draws a series of filled triangles. Each sequence of three vertices defines a separate triangle.

```c
glBegin ( GL_TRIANGLES );
    glVertex2i ( 0, 1 );
    glVertex2i ( 0, 0 );
    glVertex2i ( 1, 1 ); // abc
    glVertex2i ( 1, 1 );
    glVertex2i ( 0, 0 );
    glVertex2i ( 1, 0 ); // cde
    glVertex2i ( 2, 0 );
glEnd ();
```

**Note:** List vertices in a consistent order, say, counterclockwise. This is used to distinguish front (CCW) and back (CW) sides.
**Triangle Strip**

**GL_TRIANGLE_STRIP**: Draws a series of triangles by joining the next vertex to the previous two.

```c
glBegin ( GL_TRIANGLE_STRIP );
glVertex2i ( 0, 1 ); // a
  glVertex2i ( 0, 0 ); // b
  glVertex2i ( 1, 1 ); // c→abc
  glVertex2i ( 1, 0 ); // d→bcd
  glVertex2i ( 2, 0 ); // e→cde
  glEnd ( );
```

Orientation: The first triangle is oriented counterclockwise. OpenGL will orient all remaining triangles in the strip in the same way.

**Attributes**

Attributes: affect the manner in which objects are drawn.
- These can be placed within each `glBegin…glEnd` pair.
- Once set, they affect subsequent objects, until changed again.

Point:
- Point size: `glPointSize ( 2.0 );`
- Point color: `glColor3f ( 0.0, 0.0, 1.0 );`
  (Sets RGB color components: Red, Green, Blue.)

Line:
- Line width: `glLineWidth ( 2.0 );`
- Line color: `glColor3f ( 0.0, 0.0, 1.0 );`

Face:
- Front and/or back: `GL_FRONT, GL_BACK, GL_FRONT_AND_BACK`
- Face color: `glColor3f ( 0.0, 0.0, 1.0 );`
OpenGL Rendering Process

OpenGL Buffers

Color Buffers: The buffers that are normally drawn into.

Double-buffered system: front and back buffers. (The front buffer is visible, and drawing take place into the back buffer.)

Other examples: stereoscopic buffers (for left and right images).

Depth Buffer:

Used for hidden surface removal by storing a depth value for each pixel. Only the closest pixel is drawn.

Stencil Buffer:

This allows drawing to be restricted to certain portions of the screen.

Accumulation Buffer:

Used for accumulating a series of images into a final, composite image. In particular it can be used for antialiasing and effects like motion blur.
Lighting and Clipping

**Lighting:**
The color of each vertex is determined based on the object's **material properties** and the relationship to light sources.

**Clipping:**
Once a primitive has been assembled, it is **clipped** so that it lies within a 3-dimensional region, called the **view frustum**.
When a polygon is **partially inside**, new vertices are created as needed and vertex attributes are interpolated.

Projection and Rasterization

**Projection:**
Vertices are projected (either **perspective** or **orthogonal**).

**Viewport:**
The projected vertices are mapped to the visible portion of the screen (**viewport**).

**Rasterization:**
Primitives are converted into pixels (**fragments**).

**Per-Fragment Operations:**
Determine fragment visibility.
**Depth Test:** Hidden by a closer fragment?
**Stencil Test:** Used to restrict drawing to selected portions of the window.
Texturing and Fog

Texture:
When enabled, a fragment’s texture coordinates are used to index into a texture image, generating a texel. The texel modifies the fragment’s color based on the current texture environment, which may involve blending with the existing color.

Maxification/Minification: Different rules can be applied to interpolate values when the texel is smaller (requiring maxification) or larger (requiring minification).

Fog:
After texturing, a fog function may be applied to the fragments. This blends a fog color based on the distance of the viewer from the fragment.

Immediate Mode and Display Lists

Immediate Mode:
- By default, OpenGL operates in immediate mode, where the drawing commands are executed immediately.

Display Lists:
- Frequently executed commands that are stored for later execution.
- Display lists reside on the server (the GPU).
- Once created they cannot be modified, but they can be transformed geometrically.
- They store both geometric information (vertices) and other attributes (color, texture coordinates, surface normals).
- Display lists can reference other display lists. This is useful for the display of composite objects (e.g., the wheels of a car).
Summary

Topics Covered:
- Graphics Pipeline and General Structure
- OpenGL Drawing Primitives
- OpenGL Rendering Process