Introduction to OpenGL

What is OpenGL?

- An application programming interface (API)
- A (low-level) Graphics rendering API
- A pipe-line to generate high-quality images composed of geometric and image primitives
- A state machine

A 3D graphics API

- Separate code
  - OpenGL32.dll on Windows
  - Vendors package their own version of this library with the graphics card
  - Windows 2000 supports a software-only version of OpenGL 1.1 out of the box
A low-level 3D graphics API

• An interface to hardware
  – The library knows how to interface with the card drivers to get the hardware to handle the graphics.
  – Anything not done in hardware is done in software

• Primitive-based
  – Objects consist of points, line segments, and polygons
  – OpenGL is not aware of any connections between primitives
  – Exception
    • The GLU libraries include quadric and NURBS “objects” that encapsulate primitives for you

A Pipeline Architecture

A state machine

• Functions are global and change the state of the OpenGL environment
• State can be pushed onto stacks and popped back off
• OpenGL properties remain as you set them until you set them again
OpenGL Is Not

- A modeling language
- Compiled directly into your code
- Object-oriented

Getting Started - Syntax

- OpenGL core functions are prefixed with gl
- OpenGL utility functions are prefixed with glu
- OpenGL typedef defined types are prefixed with GL
- OpenGL constants are all caps and prefixed with GL_

History of the 3D graphics industry

- 1960s:
  - Line drawings, hidden lines, parametric surfaces (B-splines...)
  - Automated drafting & machining for car, airplane, and ships manufacturers
- 1970's:
  - Mainframes, Vector tubes (HP...)
  - Software: Solids, (CSG), Ray Tracing, Z-buffer for hidden lines
- 1980s:
  - Graphics workstations ($50K-$1M): Frame buffers, rasterizers, GL, Phigs
  - VR: CAVEs and head-mounted displays
  - CAD/CAM & GIS: CATIA, SDRC, PTC
  - Sun, HP, IBM, SGI, E&S, DEC

History of the 3D graphics industry

- 1990s:
  - PCs ($2K): Graphics boards, OpenGL, Java3D
  - CAD+Videogames+Animations: AutoCAD, SolidWorks..., Alias-Wavefront
  - Intel, many board vendors
- 2000s:
  - Laptops, PDAs, Cell Phones: Parallel graphic chips
  - Everything will be graphics, 3D, animated, interactive
  - Nvidia, Sony, Nokia
Why OpenGL?

- Cross-platform.
- Better / easier to teach.
  - Academically oriented textbooks, etc.
  - Has existed long before other API's.
- Hardware-based device drivers widely supported.
- Captures the low-level pipeline.

Other API’s?

- Microsoft’s Direct3D (DirectX)
  - Also captures the low-level pipeline.
  - I expect you to pick up a book and easily transition from OpenGL to Direct3D.
- Java3D
  - A scenegraph-based API.
  - Object oriented.
  - *Sits on top of OpenGL*.
  - Learning OpenGL will assist your understanding.

Other API’s

- PHIGS / PHIGS-Plus
  - **THE** official standard (ANSI, ISO).
  - National and international standards bodies could not keep pace with the rapid growth in graphics hardware functionality.
  - Not necessarily interested in advancing the field.
  - I was on the ANSI PHIGS-Plus committee in the late 1980’s.

Older API’s

- Display device dependent (different units / res)
- Window system dependent
- Operating system dependent

Without a standard API (such as OpenGL) - difficult to port

- Line(100,50,150,80) - device/lib 1
- Moveto(100,50) - device/lib 2
- Lineto(150,100)
OpenGL Basics

- OpenGL’s primary functions
  - Geometric description of objects.
  - Composition or lay-out of objects.
  - Color specification and lighting calculations
  - Rasterization or sampling – calculating the pixel color and depth values from the above mathematical descriptions.

- OpenGL can render:
  - Geometric primitives
  - Bitmaps and Images (Raster primitives)

Computer Graphics v. OpenGL

- Computer Graphics
  - Object or model creation
  - Data management / optimization
  - Mapping from abstract of mathematical entities to low-level geometric primitives
  - Specifying and controlling the environment (lighting, appearance, etc.)
  - Dynamic or time-varying behavior.
  - User-interaction / user interfaces for the above.

- Bottom-line: OpenGL is usually a small part of your application => porting not that hard.

Code Example

```c
void Display()
{
  glColor3f(1.0f, 1.0f, 0.0f);
  glBegin(GL_POLYGON);
  glVertex2f(-0.5f, -0.5f);
  glVertex2f(-0.5f, 0.5f);
  glVertex2f( 0.5f,  0.5f);
  glVertex2f( 0.5f, -0.5f);
  glEnd();
  glFlush();
}
```

- A possible result

Advise: Never use GL_POLYGON

Specifying Geometric primitives

- Primitives are specified using
  - glBegin(primType);
  - // define your vertices here
  - ...
  - glEnd();

- primType: GL_POINTS, GL_LINES, GL_TRIANGLES, GL_QUADS, ...
OpenGL: Front/Back Rendering

- Each polygon has two sides, front and back
- OpenGL can render the two differently
- The ordering of vertices in the list determines which is the front side:
  - When looking at the front side, the vertices go counterclockwise
    - This is basically the right-hand rule
    - Note that this still holds after perspective projection

OpenGL: Drawing Triangles

- You can draw multiple triangles between glBegin(GL_TRIANGLES) and glEnd():
  - float v1[3], v2[3], v3[3], v4[3];
  - ...
  - glBegin(GL_TRIANGLES);
  - glVertex3fv(v1); glVertex3fv(v2); glVertex3fv(v3);
  - glVertex3fv(v1); glVertex3fv(v3); glVertex3fv(v4);
  - glEnd();
  - The same vertex is used (sent, transformed, colored) many times (6 on average)

OpenGL: Triangle Strips

- An OpenGL triangle strip primitive reduces this redundancy by sharing vertices:
  - glBegin(GL_TRIANGLE_STRIP);
  - glVertex3fv(v0);
  - glVertex3fv(v1);
  - glVertex3fv(v2);
  - glVertex3fv(v3);
  - glVertex3fv(v4);
  - glVertex3fv(v5);
  - glEnd();
  - triangle 0 is v0, v1, v2
  - triangle 1 is v2, v1, v3 (why not v1, v2, v3?)
  - triangle 2 is v2, v3, v4
  - triangle 3 is v4, v3, v5 (again, not v3, v4, v5)

OpenGL: Triangle Fan

- The GL_TRIANGLE_FAN primitive is another way to reduce vertex redundancy:
OpenGL: Other Primitives

- You can draw other primitives using:
  - GL_POINTS
  - GL_LINES
  - GL_LINE_STRIP
  - GL_LINE_LOOP
  - GL_QUADS
  - …

Primitive Types

- All primitives are specified by vertices:

Points in OpenGL

```c
glBegin(GL_POINTS);
glVertex2fv(p0);
glVertex2fv(p1);
glVertex2fv(p2);
glVertex2fv(p3);
glVertex2fv(p4);
glVertex2fv(p5);
glVertex2fv(p6);
glVertex2fv(p7);
glEnd();
```

Lines in OpenGL (1/3)

- Line Segments

```c
glBegin(GL_LINES);
glVertex2fv(p0);
glVertex2fv(p1);
glVertex2fv(p2);
glVertex2fv(p3);
glVertex2fv(p4);
glVertex2fv(p5);
glVertex2fv(p6);
glVertex2fv(p7);
glEnd();
```
### Lines in OpenGL (2/3)

- **Polylines – Line Strip**

```gl
// Beginning line strip with vertices
glBegin(GL_LINE_STRIP);
glVertex2fv(p0);
glVertex2fv(p1);
glVertex2fv(p2);
glVertex2fv(p3);
glVertex2fv(p4);
glVertex2fv(p5);
glVertex2fv(p6);
glVertex2fv(p7);
glEnd();
```

### Lines in OpenGL (3/3)

- **Polylines – Line Loop**

```gl
// Beginning line loop with vertices
glBegin(GL_LINE_LOOP);
glVertex2fv(p0);
glVertex2fv(p1);
glVertex2fv(p2);
glVertex2fv(p3);
glVertex2fv(p4);
glVertex2fv(p5);
glVertex2fv(p6);
glVertex2fv(p7);
glEnd();
```

### Polygons (1/2)

- **Definition**

  - Object that is closed as in a line loop, but that has an interior

- **Simple Polygon**

  - No pair of edges of a polygon cross each other

### Polygons (2/2)

- **Convexity**

  - If all points on the line segment between any two points inside the object, or on its boundary, are inside the object
Polygons in OpenGL (1/6)

- Polygon

```c
glBegin(GL_POLYGON);
    glVertex2fv(p0);
    glVertex2fv(p1);
    glVertex2fv(p2);
    glVertex2fv(p3);
    glVertex2fv(p4);
    glVertex2fv(p5);
    glVertex2fv(p6);
    glVertex2fv(p7);
glEnd();
```

Polygons in OpenGL (2/6)

- Quadrilaterals

```c
glBegin(GL_QUADS);
    glVertex2fv(p0);
    glVertex2fv(p1);
    glVertex2fv(p2);
    glVertex2fv(p3);
    glVertex2fv(p4);
    glVertex2fv(p5);
    glVertex2fv(p6);
    glVertex2fv(p7);
glEnd();
```

Polygons in OpenGL (3/6)

- Quadstrip

```c
glBegin(GL_QUAD_STRIP);
    glVertex2fv(p1);
    glVertex2fv(p2);
    glVertex2fv(p3);
    glVertex2fv(p0);
    glVertex2fv(p4);
    glVertex2fv(p7);
    glVertex2fv(p5);
    glVertex2fv(p6);
glEnd();
```

Polygons in OpenGL (4/6)

- Triangles

```c
glBegin(GL_TRIANGLES);
    glVertex2fv(p0);
    glVertex2fv(p1);
    glVertex2fv(p2);
    glVertex2fv(p3);
    glVertex2fv(p4);
    glVertex2fv(p5);
    glVertex2fv(p6);
    glVertex2fv(p7);
glEnd();
```
• Triangle Strip

```c
glBegin(GL_TRIANGLE_STRIP);
glVertex2fv(p0);
glVertex2fv(p7);
glVertex2fv(p1);
glVertex2fv(p6);
glVertex2fv(p2);
glVertex2fv(p5);
glVertex2fv(p3);
glVertex2fv(p4);
glEnd();
```

• Triangle Fan

```c
glBegin(GL_TRIANGLE_FAN);
glVertex2fv(p0);
glVertex2fv(p1);
glVertex2fv(p2);
glVertex2fv(p3);
glVertex2fv(p4);
glVertex2fv(p5);
glVertex2fv(p6);
glVertex2fv(p7);
glEnd();
```

Attributes

• Properties that determines How to render a geometric primitive
  – Color, thickness, pattern of filling, etc.

• Color
  – Three color theory

OpenGL’s State Machine

• All rendering attributes are encapsulated in the OpenGL State
  – rendering styles
  – shading
  – lighting
  – texture mapping
Manipulating OpenGL State

- Appearance is controlled by current state
  - for each (primitive to render) {
    - update OpenGL state
    - render primitive
  - }
- Manipulating vertex attributes is the most common way to manipulate state
  - glColor*() / glIndex*()
  - glNormal*()
  - glTexCoord*()

Controlling current state

- Setting State
  - glPointSize(size);
  - glLineStipple(repeat, pattern);
  - glShadeModel(GL_SMOOTH);
- Enabling Features
  - glEnable(GL_LIGHTING);
  - glDisable(GL_TEXTURE_2D);

Simple Example

```c
void DrawBlueQuad()
{
  glColor3f(0.0f, 0.0f, 1.0f);
  glBegin(GL_QUADS);
  glVertex2f(0.0f, 0.0f);
  glVertex2f(1.0f, 0.0f);
  glVertex2f(1.0f, 1.0f);
  glVertex2f(0.0f, 1.0f);
  glEnd();
}
```

- This type of operation is called immediate-mode rendering;
- Each command happens immediately
- Although you may not see the result if you use double buffering
  - Things get drawn into the back buffer
  - Then buffers are swapped

OpenGL Command Formats

- `glVertex2f(x, y)`
- `glVertex2fv(v)`

- No method overloading in C or FORTRAN
- Internally everything is usually a float - I think.
OpenGL: Specifying Color

- Can specify other properties such as color
  - To produce a single aqua-colored triangle:
    ```
    glColor3f(0.1, 0.5, 1.0);
    glVertex3fv(v0); glVertex3fv(v1);
    glVertex3fv(v2);
    ```
  - To produce a smoothly shaded triangle:
    ```
    glColor3f(1, 0, 0); glVertex3fv(v0);
    glColor3f(0, 1, 0); glVertex3fv(v1);
    glColor3f(0, 0, 1); glVertex3fv(v2);
    ```
  - In OpenGL, colors can also have a fourth component $\alpha$ (opacity or 1-transparency)
    - Generally want $\alpha = 1.0$ (opaque);

Window system independent

- OpenGL is window system independent
  - No window management functions – create windows, resize windows, event handling, etc
  - This is to ensure the application’s portability
  - Creates some headaches though – just a pure OpenGL program won’t work anywhere.

More APIs are needed

- X window system: GLX
- Apple Macintosh: AGL
- Microsoft Windows: WGL
- Additional libraries are needed to create Graphical User Interface (GUI) elements, such as sliders, buttons, menus, etc.

- Problem – you need to learn and implement them all to write truly portable software