OpenGL tutorial

09/28/2015
Ji Hwan Park
OpenGL?

• Is a computer graphics rendering Application Programming Interface (API)
• Can create 2D and 3D geometric objects and images
• Is operating systems and window system independent
History of OpenGL

• Ver 1.0: July 1st, 1994, fixed-function pipeline
• Ver 2.0: programmable vertex and fragment shaders
• Ver 3.1: removed the fixed-function pipeline
• Ver 3.2: Geometry shaders
• Ver 4.1: tessellation-control and tessellation-evaluation
OpenGL pipeline
GLUT

- OpenGL need a place to render into
- OpenGL Utility Toolkit (GLUT), Qt, so on.
- http://freeglut.sourceforge.net/
- http://www.qt.io/developers/
- In this section, we will use GLUT
GLUT

• Multiple windows for OpenGL rendering.
• Callback driven event processing.
• An `idle' routine and timers.
• Utility routines to generate various solid and wire frame objects.
• Support for bitmap and stroke fonts.
• Miscellaneous window management functions.
Examples

```c
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA);
    glutInitWindowSize(512, 512);
    glutInitContextVersion(4, 3);
    glutInitContextProfile(GLUT_CORE_PROFILE);
    glutInitContextProfile(GLUT_Core_PROFILE);
    glutCreateWindow(argv[0]);
    if (glewInit()) {
        cerr << "Unable to initialize GLEW ... exiting" << endl;
        exit(EXIT_FAILURE);
    }
    init();
    glutDisplayFunc(display);
    glutMouseFunc(mouseEvent);
    glutMainLoop();
}
```
GLEW: The OpenGL Extension Wrangler

- To simplify both verifying extension support and dealing with the associated function pointers
- calling `glewInit()` after you’ve created your window
- To verify the OpenGL version,

```c
glewInit(); // Initialize the GLEW library
if (GL_version_3_0) {
    // use functions from OpenGL 3.0
}
```
Geometric Objects

• Geometric objects are represented using vertices
• A vertex is a collection of generic attributes
  • Positional coordinates (x,y,z,w)
  • Colors (r,g,b,a)
  • Texture coordinates (u,v,w)
  • Any other associated with that point in space
• Vertex data must be stored in Vertex Buffer Objects (VBOs)
• VBOs must be stored in Vertex Array Objects (VAOs)
Example

```c
void
init(void)
{
    glGenVertexArrays(NumVAOs, VAOs);
    glBindVertexArray(VAOs[Triangles]);
    GLfloat vertices[NumVertices][2] = {
        { -0.90, -0.90 }, // Triangle 1
        { 0.85, -0.90 },
        { -0.90, 0.85 },
        { 0.90, -0.85 }, // Triangle 2
        { 0.90, 0.90 },
        { -0.85, 0.90 }
    };
    glGenBuffers(NumBuffers, Buffers);
    glBindBuffer(GL_ARRAY_BUFFER, Buffers[ArrayBuffer]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices),
                 vertices, GL_STATIC_DRAW);
}
Shaders

• Vertex shaders: process individual vertices

• Fragment shaders: process a fragment from the rasterization process
Shaders (1)

ShaderInfo shaders[] = {
    { GL_VERTEX_SHADER, "triangles.vert" },
    { GL_FRAGMENT_SHADER, "triangles.frag" },
    { GL_NONE, NULL }
};
GLuint program = LoadShaders(shaders);
glUseProgram(program);
glVertexAttribPointer(vPosition, 2, GL_FLOAT, GL_FALSE, 0, BUFFER_OFFSET(0));
glEnableVertexAttribArray(vPosition);
Shaders (2)

```glsl
#version 430 core
layout(location = 0) in vec4 vPosition;
void main()
{
    gl_Position = vPosition;
}

#version 430 core
out vec4 fColor;
void main()
{
    fColor = vec4(0.0, 0.0, 1.0, 1.0);
}
```
Example (rendering)

```c
void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    glBindVertexArray(VAOs[Triangles]);
    glDrawArrays(GL_TRIANGLES, 0, NumVertices);
    glFlush();
}
```
Buffers

• Color buffer: RGB or sRGB color data, and may also contain alpha values for each pixel in the framebuffer.

• Depth buffer: a depth value for each pixel

• Stencil buffer: to restrict drawing to certain portions of the screen.

• `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT)`
Drawing functions

• Non-indexed drawing function:
  • void glDrawArrays(GLenum mode, GLint first, GLsizei count);

• Indexed drawing function:
  • void glDrawElements(GLenum mode, GLsizei count, GLenum type, const GLvoid *indices);
Geometric primitives
Buffering

• Single buffering: use a single buffer.
  • glFlush()

• Double buffering: use two buffers – a front buffer for displaying, a back buffer for the next image
  • glutSwapBuffers()
Interaction (1)

• void mouseEvent() – a function is called when a user presses and releases mouse buttons in the window
  • Button: GLUT_LEFT_BUTTON, GLUT_RIGHT_BUTTON, GLUT_MIDDLE_BUTTON
  • State: GLUT_DOWN, GLUT_UP
Interaction (2)

• `void glutMotionFunc()` – a function is called when a key is pressed
• `void glutKeyboardFunc()` – a function is called when a mouse is moved
Any Questions?