### Foundations of Computer Graphics

**Online Lecture 6: OpenGL 1**

**Overview and Motivation**

Ravi Ramamoorthi

---

**This Lecture**

- Introduction to OpenGL and simple demo code
  - mytest1.cpp; you compiled mytest3.cpp for HW 0
- I am going to show (and write) actual code
  - Code helps you understand HW 2 better
- Simple demo of mytest1
- This lecture deals with very basic OpenGL setup. Next 2 lectures will likely be more interesting

---

**Outline**

- **Basic idea about OpenGL**
  - Basic setup and buffers
  - Matrix modes
  - Window system interaction and callbacks
  - Drawing basic OpenGL primitives
  - Initializing Shaders

---

**Introduction to OpenGL**

- OpenGL is a graphics **API**
  - Portable software library (platform-independent)
  - Layer between programmer and graphics hardware
  - Uniform instruction set (hides different capabilities)
- OpenGL can fit in many places
  - Between application and graphics system
  - Between higher level API and graphics system

---

**Why OpenGL?**

- **Why do we need OpenGL or an API?**
  - Encapsulates many basic functions of 2D/3D graphics
  - Think of it as high-level language (C++) for graphics
  - History: Introduced SGI in 92, maintained by Khronos
  - Precursor for DirectX, WebGL, Java3D etc.

---

**Programmer’s View**

Slide inspired by Greg Humphreys
OpenGL Rendering Pipeline

- **Vertices** → Geometry
- **Scan Conversion (Rasterize)** → Pixel Operations
- **Texture Memory** → Fragment Operations
- **Images** → Programmable in Modern GPUs (Fragment Shader)
- **Programmable in Modern GPUs (Vertex Shader)**

Traditional Approach: Fixed function pipeline (state machine)
New Development (2003-): Programmable pipeline

GPUs and Programmability

- Since 2003, can write vertex/pixel shaders
- Fixed function pipeline special type of shader
- Like writing C programs (see GLSL book)
- Performance >> CPU (even used for non-graphics)
- Operate in parallel on all vertices or fragments

Outline

- Basic idea about OpenGL
- Basic setup and buffers
- Matrix modes
- Window system interaction and callbacks
- Drawing basic OpenGL primitives
- Initializing Shaders

Foundations of Computer Graphics

Online Lecture 6: OpenGL 1

*Basic Setup and Buffers, Matrix Modes*

Ravi Ramamoorthi

Buffers and Window Interactions

- Buffers: Color (front, back, left, right), depth (z), accumulation, stencil. When you draw, you write to some buffer (most simply, front and depth)
- Buffers also used for vertices etc. Buffer data and buffer arrays (will see in creating objects)
- No window system interactions (for portability)
  - But can use GLUT (or Motif, GLX, Tcl/Tk)
  - Callbacks to implement mouse, keyboard interaction
Basic setup (can copy; slight OS diffs)

```c
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    // Requests the type of buffers (Single, RGB).
    // Think about what buffers you would need...
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    // Need to add GLUT_3_2_CORE_PROFILE for Apple/Mac OS
    glutInitWindowSize (500, 500);
    glutInitWindowPosition (100, 100);
    glutCreateWindow ("Simple Demo with Shaders");
    // glewInit(); // GLEW related stuff for non-Apple systems
    init (); // Always initialize first
    // Now, we define callbacks and functions for various tasks.
    ...
}
```

Outline

- Basic idea about OpenGL
- Basic setup and buffers
- **Matrix modes**
- Window system interaction and callbacks
- Drawing basic OpenGL primitives
- Initializing Shaders

Viewing in OpenGL

- Inspired by old OpenGL. Now, only best practice, not requirement
  - You could do your own thing, but this is still the best way to develop viewing
- Viewing consists of two parts
  - Object positioning: model view transformation matrix
  - View projection: projection transformation matrix
- Old OpenGL (no longer supported/taught in 167x), two matrix stacks
  - GL_MODELVIEW_MATRIX, GL_PROJECTION_MATRIX
  - Can push and pop matrices onto stacks
- New OpenGL: Use C++ STL templates to make stacks as needed
  - e.g. stack <mat4> modelview ; modelview.push(mat4(1.0)) ;
  - GLM libraries replace many deprecated commands. Include mat4

Basic initialization code for viewing

```c
#include <GL/glut.h> //also GL/glew.h; :GLUT/glut.h for Mac OS
#include <cmath> //also sin, acos, glm, others
int mouseoldx, mouseoldy ; // For mouse motion
GLfloat eyeloc = 2.0 ; // Where to look from; initially 0 -2, 2
glm::mat4 projection, modelview; // The mvp matrices themselves
void init (void) {
    /* select clearing color */
    glClearColor (0.0, 0.0, 0.0, 0.0);
    /* initialize viewing values */
    projection = glm::mat4(1.0f); // The identity matrix
    // Think about this. Why is the up vector not normalized?
    modelview = glm::lookAt(glm::vec3(0), glm::vec3(0, 1, 1), glm::vec3(0, 0, 0));
    // (To be cont’d) Geometry and shader set up later ...
}
```
Outline

- Basic idea about OpenGL
- Basic setup and buffers
- Matrix modes
- Window system interaction and callbacks
- Drawing basic OpenGL primitives
- Initializing Shaders

Window System Interaction

- Not part of OpenGL
  - Toolkits (GLUT) available (also freeglut)
  - Callback functions for events (similar to X, Java,)
    - Keyboard, Mouse, etc.
    - Open, initialize, resize window
- Our main func included
  - glutDisplayFunc(display);
  - glutReshapeFunc(reshape);
  - glutKeyboardFunc(keyboard);
  - glutMouseFunc(mouse);
  - glutMotionFunc(mousedrag);

Basic window interaction code

/* Defines what to do when various keys are pressed */
void keyboard(unsigned char key, int x, int y)
{
    switch (key) {
    case 27:  // Escape to quit
        exit(0);
        break;
    default:
        break;
    }
}

Basic window interaction code

/* Reshapes the window appropriately */
void reshape(int w, int h)
{
    glViewport(0, 0, (GLsizei) w, (GLsizei) h);
    // Note that the field of view takes in a radian angle
    projection = glm::perspective(30.0f / 180.0f * glm::pi<float>(),
        (GLfloat)w / (GLfloat)h, 1.0f, 10.0f);
    glUniformMatrix4fv(projectionPos, 1, GL_FALSE, &projection[0][0]);
    // To send the projection matrix to the shader
}

Mouse motion (demo)

void mouse(int button, int state, int x, int y) {
    if (button == GLUT_LEFT_BUTTON) {
        if (state == GLUT_UP) {
            // Do Nothing
        } else if (state == GLUT_DOWN) {
            mouseoldx = x; mouseoldy = y; // so we can move wrt x, y
        }
    } else if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {
        // Reset gluLookAt
        eyeloc = 2.0;
        modelview = glm::lookAt(glm::vec3(0, -eyeloc, eyeloc),
            glm::vec3(0, 0, 0), glm::vec3(0, 1, 1));
        // Send the updated matrix to the shader
        glUniformMatrix4fv(modelviewPos, 1, GL_FALSE, &modelview[0][0]);
        glutPostRedisplay(); // Redraw scene
    }
}
Mouse drag (demo)

```c
void mousedrag(int x, int y) {
    int yloc = y - mouseoldy;    // We will use the y coord to zoom in/out
    eyeloc += 0.005*yloc;        // Where do we look from
    if (eyeloc < 0) eyeloc = 0.0;
    mouseoldy = y;
    /* Set the eye location */
    modelview = glm::lookAt(glm::vec3(0, -eyeloc, eyeloc),
                            glm::vec3(0, 0, 0), glm::vec3(0, 1, 1));
    // Send the updated matrix over to the shader
    glUniformMatrix4fv(modelviewPos,1,GL_FALSE,&modelview[0][0]);
    glutPostRedisplay();
}
```

Outline

- Basic idea about OpenGL
- Basic setup and buffers
- Matrix modes
- Window system interaction and callbacks
- Drawing basic OpenGL primitives
- Initializing Shaders

Geometry

- Points (GL_POINTS)
  - Stored in Homogeneous coordinates
- Line segments (GL_LINES)
  - Also (GL_LINE_STRIP, GL_LINE_LOOP)
- Triangles (GL_TRIANGLES)
  - Also strips, fans (GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN)
- More complex primitives (GLUT): Sphere, teapot, cube....
  - Must now be converted into triangles (which is what skeleton does)

New OpenGL Primitives (fewer)

- Points
- Lines (also strips, loops)
- Polygon
- Triangle
- Quad
- Quad Strip
- Triangle Strip
- Triangle Fan

Old OpenGL: Drawing

- Enclose vertices between glBegin() ... glEnd() pair
- Can include normal C code and attributes like the colors
- Inside are commands like glVertex3f, glColor3f
- Attributes must be set before the vertex
- Assembly line (pass vertices, transform, shade)
  - These are vertex, fragment shaders on current GPUs
  - Immediate Mode: Sent to server and drawn
Old OpenGL: Drawing (not used)

```c
void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    // draw polygon (square) of unit length centered at the origin
    // This code draws each vertex in a different color.
    glBegin(GL_POLYGON);
    glColor3f (1.0, 0.0, 0.0);  // RED
    glVertex3f (0.5, 0.5, 0.0);
    glColor3f (0.0, 1.0, 0.0);  // BLUE
    glVertex3f (-0.5, 0.5, 0.0);
    glColor3f (0.0, 0.0, 1.0);  // GREEN
    glVertex3f (-0.5, -0.5, 0.0);
    glColor3f (1.0, 1.0, 1.0);  // WHITE
    glVertex3f (0.5, -0.5, 0.0);
    glEnd();
    glFlush();
}
```

Old OpenGL: Drawing

- Client-Server model (client generates vertices, server draws) even if on same machine
- `glFlush()` forces client to send network packet
- `glFinish()` waits for ack, sparingly use synchronization

New OpenGL: Vertex Array Objects (next)

Modern OpenGL: Floor Specification

```c
const GLfloat floorverts[4][3] = {
    {0.5, 0.5, 0.0},
    {-0.5, 0.5, 0.0},
    {-0.5, -0.5, 0.0},
    {0.5, -0.5, 0.0}
};
const GLfloat floorcol[4][3] = {
    {1.0, 0.0, 0.0},
    {0.0, 1.0, 0.0},
    {0.0, 0.0, 1.0},
    {1.0, 1.0, 1.0}
};
const GLubyte floorinds[1][6] = { 0, 1, 2, 0, 2, 3 };  // triangles
```

Modern OpenGL: Vertex Array Objects

```c
const int numobjects = 2;  // number of objects for buffer
const int numperobj = 3;  // Vertices, colors, indices
GLuint VAOs[numobjects];    // A Vertex Array Object per object
GLuint buffers[3*numperobj*numobjects]; // List of buffers geometric data
GLuint objects[numobjects]; // For each object
GLenum PrimType[numobjects]; // Primitive Type (triangles, strips)
GLsizei NumElems[numobjects]; // Number of geometric elements

// Floor Geometry is specified with a vertex array
enum {Vertices, Colors, Elements} ; // For arrays for object
enum {FLOOR, FLOOR2} ; // For objects, for the floor

void initobject (GLuint object, GLfloat * vert, GLint sizevert, GLfloat * col, GLint sizecol, GLubyte * inds, GLint sizeind, GLenum type) {
    int offset = object * numperobj;
    glBindVertexArray(VAOs[object]);
    glBindBuffer(GL_ARRAY_BUFFER, buffers[Vertices+offset]) ;
    glBufferData(GL_ARRAY_BUFFER, sizevert, vert,GL_STATIC_DRAW);
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), 0);
    glBindBuffer(GL_ARRAY_BUFFER, buffers[Colors+offset]) ;
    glBufferData(GL_ARRAY_BUFFER, sizecol, col,GL_STATIC_DRAW);
    // Use layout location 1 for the colors
}
```

Modern OpenGL: Initialize Buffers

```c
void initbuffers(int object, GLuint * VAOs, GLuint * VBOs, GLuint * floorverts, GLuint * floorcol) {
    // Create unique identifiers
    GLuint bufferIDs[3*numperobj*numobjects];
    glGenBuffers(numperobj*numobjects, bufferIDs); // and for buffers

    void deleteBuffers() { // Like a destructor
        // glDeleteVertexArrays(numobjects, VAOs);
        GLuint bufferIDs[3*numperobj*numobjects]; // create unique identifiers
        glGenBuffers(numperobj*numobjects, bufferIDs); // and for buffers

    void deleteBuffers() { // Like a destructor
        glDeleteVertexArrays(numobjects, VAOs);
        glDeleteBuffers(numperobj*numobjects, buffers);
```
Modern OpenGL: Initialize Buffers

```c
void initobject (GLuint object, GLfloat * vert, GLint sizevert, GLfloat * col, GLint sizecol, GLubyte * inds, GLint sizeind, GLenum type) {
    // ...
    // Use layout location 1 for the colors
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), 0);
    // Use layout location 0 for the indices
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, buffers[Elements+offset]) ;
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeind, inds,GL_STATIC_DRAW);
    PrimType[object] = type;
    NumElems[object] = sizeind;
    // Prevent further modification of this VAO by unbinding it
    glBindVertexArray(0);
}
```

Modern OpenGL: Draw Vertex Object

```c
void drawobject(GLuint object) {
    glBindVertexArray(VAOs[object]);
    glDrawElements(PrimType[object], NumElems[object], GL_UNSIGNED_BYTE, 0);
    glBindVertexArray(0); //unbind
}
```

Initialization for Drawing, Shading

```c
#include "shaders.h"
GLuint vertexshader, fragmentshader, shaderprogram ; // shaders
// Initialization in init() for Drawing
glGenVertexArrays(numobjects, VAOs) ;
glGenBuffers(numperobj*numobjects, buffers) ;
initobject(FLOOR, (GLfloat *) floorverts, sizeof(floorverts), (GLfloat *) floorcol, sizeof (floorcol), (GLubyte *) floorinds, sizeof (floorinds), GL_TRIANGLES) ;
initobject(FLOOR2, (GLfloat *) floorverts2, sizeof(floorverts2), (GLfloat *) floorcol2, sizeof (floorcol2), (GLubyte *) floorinds2, sizeof (floorinds2), GL_TRIANGLES) ;
// In init() for Shaders, discussed next
vertexshader = initshaders(GL_VERTEX_SHADER, "shaders/nop.vert") ;
fragmentshader = initshaders(GL_FRAGMENT_SHADER, "shaders/nop.frag") ;
shaderprogram = initprogram(vertexshader, fragmentshader) ;
```

Demo (change colors)

```c
#include "shaders.h"
GLuint vertexshader, fragmentshader, shaderprogram ; // shaders
// Initialization in init() for Drawing
glGenVertexArrays(numobjects, VAOs) ;
glGenBuffers(numperobj*numobjects, buffers) ;
initobject(FLOOR, (GLfloat *) floorverts, sizeof(floorverts), (GLfloat *) floorcol, sizeof (floorcol), (GLubyte *) floorinds, sizeof (floorinds), GL_TRIANGLES) ;
```

Outline

- Basic idea about OpenGL
- Basic setup and buffers
- Matrix modes
- Window system interaction and callbacks
- Drawing basic OpenGL primitives
- Initializing Shaders
Full OpenGL Pipeline

- Vertex Shader
- Tessellation Control Shader
- Tessellation Evaluation Shader
- Geometry Shader
- Fragment Shader
- Text/Image Data (Program)
- Final Pixel Color (Image)

User/program generates original vertices, textures

We cover programmable vertex and fragment shaders in course
OpenGL primitive setup, clipping, rasterization not programmable

Simplified OpenGL Pipeline

- User specifies vertices (via vertex arrays)
- For each vertex in parallel
  - OpenGL calls user-specified vertex shader:
    Transform vertex (ModelView, Projection), other ops
- For each primitive, OpenGL rasterizes
  - Generates a fragment for each pixel the primitive covers
- For each fragment in parallel
  - OpenGL calls user-specified fragment shader:
    Shading and lighting calculations
  - OpenGL handles z-buffer depth test unless overwritten

Shader Setup

- Initializing (shader itself discussed later)
  1. Create shader (Vertex and Fragment)
  2. Compile shader
  3. Attach shader to program
  4. Link program
  5. Use program

Shader Initialization Code

```c
GLuint initshaders (GLenum type, const char *filename) {
  GLuint shader = glCreateShader(type) ;
  GLint compiled ;
  string str = textFileRead (filename) ;
  const GLchar * cstr = str.c_str() ;
  glShaderSource (shader, 1, &cstr, NULL) ;
  glCompileShader (shader) ;
  glGetShaderiv (shader, GL_COMPILE_STATUS, &compiled) ;
  if (!compiled) {
    shadererrors (shader) ;
    throw 3 ;
  }
  return shader ;
}
```
Linking Shader Program

```c
GLuint initprogram (GLuint vertexshader, GLuint fragmentshader) {
    GLuint program = glCreateProgram() ;
    GLint linked ;
    glAttachShader(program, vertexshader) ;
    glAttachShader(program, fragmentshader) ;
    glLinkProgram(program) ;
    glGetProgramiv(program, GL_LINK_STATUS, &linked) ;
    if (linked) glUseProgram(program) ;
    else {
        programerrors(program) ;
        throw 4 ; }
    cout << "Shader program successfully attached and linked." << endl;
    return program ; }
```

Basic (nop) vertex shader

```c
# version 330 core
layout (location = 0) in vec3 position;
layout (location = 1) in vec3 color;

// Shader outputs, if any
out vec3 Color;

// Uniform variables
uniform mat4 modelview;
uniform mat4 projection;

void main() {
    gl_Position = projection * modelview * vec4(position, 1.0f);
    Color = color; // Just forward this color to the fragment shader
}
```

Basic (nop) fragment shader

```c
# version 330 core
in vec3 Color;

// Output
out vec4 fragColor;
void main (void) {
    fragColor = vec4(Color, 1.0f);
}
```