Foundations of Computer Graphics (Fall 2012)
CS 184, Lecture 6: OpenGL 1
http://inst.eecs.berkeley.edu/~cs184

To Do
- Start thinking (now) about HW 2 and HW 4.
- HW 2 (much) more difficult than HW 1
  - Much of it based on Thu lecture
  - START EARLY

Demo: Surreal (HW 3)

This Lecture
- Introduction to OpenGL and simple demo code
  - mytest1.cpp; you compiled mytest3.cpp for HW 0
  - Include source (compiles Mac OS only) for programs
- I am going to show (maybe write) actual code
  - Online code helps you understand HW 2, HW 3, 4 better
  - ASK QUESTIONS if confused!
- 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 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.

Best source for OpenGL is the redbook. Of course, this is more a reference manual than a textbook, and you are better off implementing rather than reading end to end.
**Programmer’s View**

Application

Application

OpenGL Application Programming Interface

Hardware and software (graphics card)

Output Device

Input Device

Input Device

---

**OpenGL Rendering Pipeline**

- Programmable in Modern GPUs (Vertex Shader)
- Programmable in Modern GPUs (Fragment Shader)
- Geometry
- Primitive Operations
- Pixel Operations
- Scan Conversion (Rasterize)
- Texture Memory
- Framebuffer

Slide inspired by Greg Humphreys

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

- Are teaching CS 184 with programmable shaders

---

**Outline**

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

---

**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)
- No window system interactions (for portability)
  - But can use GLUT (or Motif, GLX, Tcl/Tk)
  - Callbacks to implement mouse, keyboard interaction

---

**Basic setup code (you will likely copy)**

```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);
    glutInitWindowSize (500, 500);
    glutInitWindowPosition (100, 100);
    glutCreateWindow ("Simple Demo with Shaders");
    glewInit();
    // Always initialize first
    init ();
    // Now, we define callbacks and functions for various tasks.
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutMouseFunc(mouse);
    glutMotionFunc(mousedrag);
    glutMainLoop();
    return 0;   /* ANSI C requires main to return int. */
}
```
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

- Viewing consists of two parts
  - Object positioning: model view transformation matrix
    - View projection: projection transformation matrix
  - Old OpenGL (still supported), 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
  - OpenGL’s camera is always at the origin, pointing in the –z direction
  - Transformations move objects relative to the camera
  - In old OpenGL, Matrices are column-major and right-multiply top of stack. (Last transform in code is first actually applied). In new GLM, it’s confusing since matrices are row-order but still right-multiply (read the assignment notes and documentation).

Basic initialization code for viewing

```c
#include <GL/glut.h>
#include <stdlib.h>
int mouseoldx, mouseoldy ; // For mouse motion
GLfloat eyeloc = 2.0 ; // Where to look from; initially 0 -2, 2

void init (void)
{
  /* select clearing color */
  glClearColor (0.0, 0.0, 0.0, 0.0);
  /* initialize viewing values */
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  // Think about this. Why is the up vector not normalized?
  glMatrixMode(GL_MODELVIEW) ;
  glLoadIdentity();
  glLoadIdentity(0,-eyeloc,eyeloc,0,0,0,0,1,1) ;
  // (To be cont'd). Geometry and shader set up later ...
}
```

Basic window interaction code

```c
#include <GL/glut.h>
#include <stdlib.h>
int mouseoldx, mouseoldy ; // For mouse motion
GLfloat eyeloc = 2.0 ; // Where to look from; initially 0 -2, 2

void init (void)
{
  /* select clearing color */
  glClearColor (0.0, 0.0, 0.0, 0.0);
  /* initialize viewing values */
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  // Think about this. Why is the up vector not normalized?
  glMatrixMode(GL_MODELVIEW) ;
  glLoadIdentity();
  glLoadIdentity(0,-eyeloc,eyeloc,0,0,0,0,1,1) ;
  // (To be cont'd). Geometry and shader set up later ...
}
```

Window System Interaction

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

Basic window interaction code

```c
#include <GL/glut.h>
#include <stdlib.h>
int mouseoldx, mouseoldy ; // For mouse motion
GLfloat eyeloc = 2.0 ; // Where to look from; initially 0 -2, 2

void init (void)
{
  /* select clearing color */
  glClearColor (0.0, 0.0, 0.0, 0.0);
  /* initialize viewing values */
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  // Think about this. Why is the up vector not normalized?
  glMatrixMode(GL_MODELVIEW) ;
  glLoadIdentity();
  glLoadIdentity(0,-eyeloc,eyeloc,0,0,0,0,1,1) ;
  // (To be cont'd). Geometry and shader set up later ...
}
```

/* 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 ;
    }
}
```

/* Reshapes the window appropriately */
void reshape(int w, int h)
{
    glViewport (0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(30.0, (GLfloat)w/(GLfloat)h, 1.0, 10.0) ;
}```
**Mouse motion (demo)**

```c
/* Defines a Mouse callback to zoom in and out */
/* This is done by modifying gluLookAt */
/* The actual motion is to mousedrag */
/* Mouse simply sets state for mousedrag */
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 ;
        glMatrixMode(GL_MODELVIEW) ;
        glLoadIdentity() ;
        gluLookAt(0,-eyeloc,eyeloc,0,0,0,0,1,1) ;
        glutPostRedisplay() ;
    }
}
```

**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 */
glMatrixMode(GL_MODELVIEW) ;
gluLookAt(0,-eyeloc,eyeloc,0,0,0,0,1,1) ;
glutPostRedisplay() ;
}
```

**Outline**

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

**OpenGL Primitives**

- Points
- Lines
- Polygon
- Triangle
- Quad
- Quad Strip
- Triangle Strip
- Triangle Fan

**Geometry**

- Points (GL_POINTS)
  - Stored in Homogeneous coordinates
- Line segments (GL_LINES)
- Polygons
  - Simple, convex (take your chances with concave)
  - Tessellate, GLU for complex shapes
  - Rectangles: glRect
- Special cases (strips, loops, triangles, fans, quads)
- More complex primitives (GLUT): Sphere, teapot, cube,...

**GLUT 3D Primitives**

- Cube
- Sphere
- Teapot
Old OpenGL: Drawing

- Enclose vertices between glBegin() ... glEnd() pair
- Can include normal C code and attributes like the colors
- 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
- 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 Buffer Objects (next)

Old OpenGL: Specifying Geometry

- glBegin(GL_POLYGON) ; // Chapter 2 but I do Counter Clock W
- glVertex2f (4.0, 0.0) ;
- glVertex2f (6.0, 1.5) ;
- glVertex2f (4.0, 3.0) ;
- glVertex2f (0.0, 3.0) ;
- glVertex2f (0.0, 0.0) ;
- glEnd() ;

Modern OpenGL: Floor Specification

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][4] = { {0, 1, 2, 3} } ;
const GLfloat floorverts2[4][3] = {
{0.5, 0.5, 1.0}, {-0.5, 0.5, 1.0}, {-0.5, -0.5, 1.0}, {0.5, -0.5, 1.0}
} ;
const GLfloat floorcol2[4][3] = {
{1.0, 0.0, 0.0}, {1.0, 0.0, 0.0}, {1.0, 0.0, 0.0}, {1.0, 0.0, 0.0}
} ;
const GLubyte floorinds2[1][4] = { {0, 1, 2, 3} } ;
Modern OpenGL: Draw Vertex Object

```c
void drawobject(GLuint object) {
    int offset = object * numperobj;
    glBindBuffer(GL_ARRAY_BUFFER, buffers[Vertices+offset]);
    glVertexPointer(3, GL_FLOAT, 0, BUFFER_OFFSET(0));
    glEnableClientState(GL_VERTEX_ARRAY);
    glBindBuffer(GL_ARRAY_BUFFER, buffers[Colors+offset]);
    glColorPointer(3, GL_FLOAT, 0, BUFFER_OFFSET(0));
    glEnableClientState(GL_COLOR_ARRAY);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, buffers[Elements+offset]);
    glDrawElements(PrimType[object], NumElems[object], GL_UNSIGNED_BYTE, BUFFER_OFFSET(0));
}
```

```c
void display(void) {
    glClear (GL_COLOR_BUFFER_BIT);
    drawobject(FLOOR) ;
    drawobject(FLOOR2);
    glFlush ();
}
```

Initialization for Drawing, Shading

```c
#include "shaders.h"
GLuint vertexshader, fragmentshader, shaderprogram ; // shaders
// Initialization in init() for Drawing
glGenBuffers(numperobj*numobjects, buffers) ;
initobject(FLOOR, (GLfloat *) floorverts, sizeof(floorverts),
            (GLfloat *) floorcol, sizeof (floorcol), (GLubyte *) floorinds,
            sizeof (floorinds), GL_POLYGON) ;
initobject(FLOOR2, (GLfloat *) floorverts2, sizeof(floorverts2),
            (GLfloat *) floorcol2, sizeof (floorcol2), (GLubyte *) floorinds2,
            sizeof (floorinds2), GL_POLYGON) ;
// 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)

Outline

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

OpenGL Rendering Pipeline

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

Simplified OpenGL Pipeline

- User specifies vertices (vertex buffer object)
- 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 fragment covers
- For each fragment in parallel
  - OpenGL calls user-specified fragment shader: Shading and lighting calculations
  - OpenGL handles z-buffer depth test unless overwritten
- Modern OpenGL is "lite" basically just a rasterizer
  - "Real" action in user-defined vertex, fragment shaders
Shader Setup

- Initializing (shader itself discussed later)
- Create shader (Vertex and Fragment)
- Compile shader
- Attach shader to program
- Link program
- Use program

- Shader source is just sequence of strings
- Similar steps to compile a normal program

Shader Initialization Code

```cpp
GLuint initshaders (GLenum type, const char *filename) {
    // Using GLSL shaders, OpenGL book, page 679
    GLuint shader = glCreateShader(type) ;
    GLint compiled ;
    string str = textFileRead(filename) ;
    GLchar * cstr = new GLchar[str.size()+1] ;
    const GLchar * cstr2 = cstr ; // Weirdness to get a const char
    strcpy(cstr,str.c_str()) ;
    glShaderSource(shader, 1, &cstr2, NULL) ;
    glCompileShader(shader) ;
    glGetShaderiv(shader, GL_COMPILE_STATUS, &compiled) ;
    if (!compiled) {
        shadererrors(shader) ;
        throw 3 ;
    }
    return shader ;
}
```

Linking Shader Program

```cpp
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 ;
    }
    return program ;
}
```

Basic (nop) vertex shader

```glsl
# version 120
// Mine is an old machine. For version 130 or higher, do
// out vec4 color ;
// That is certainly more modern
varying vec4 color ;
void main() {
    gl_Position = gl_ProjectionMatrix * gl_ModelViewMatrix * gl_Vertex ;
    color = gl_Color ;
}
```

Basic (nop) fragment shader

```glsl
# version 120
// Mine is an old machine. For version 130 or higher, do
// in vec4 color ;
// That is certainly more modern
attribute vec4 color ;
void main (void) {
    gl_FragColor = color ;
}
```