CSE 167:
Introduction to Computer Graphics
Lecture #2: OpenGL Overview

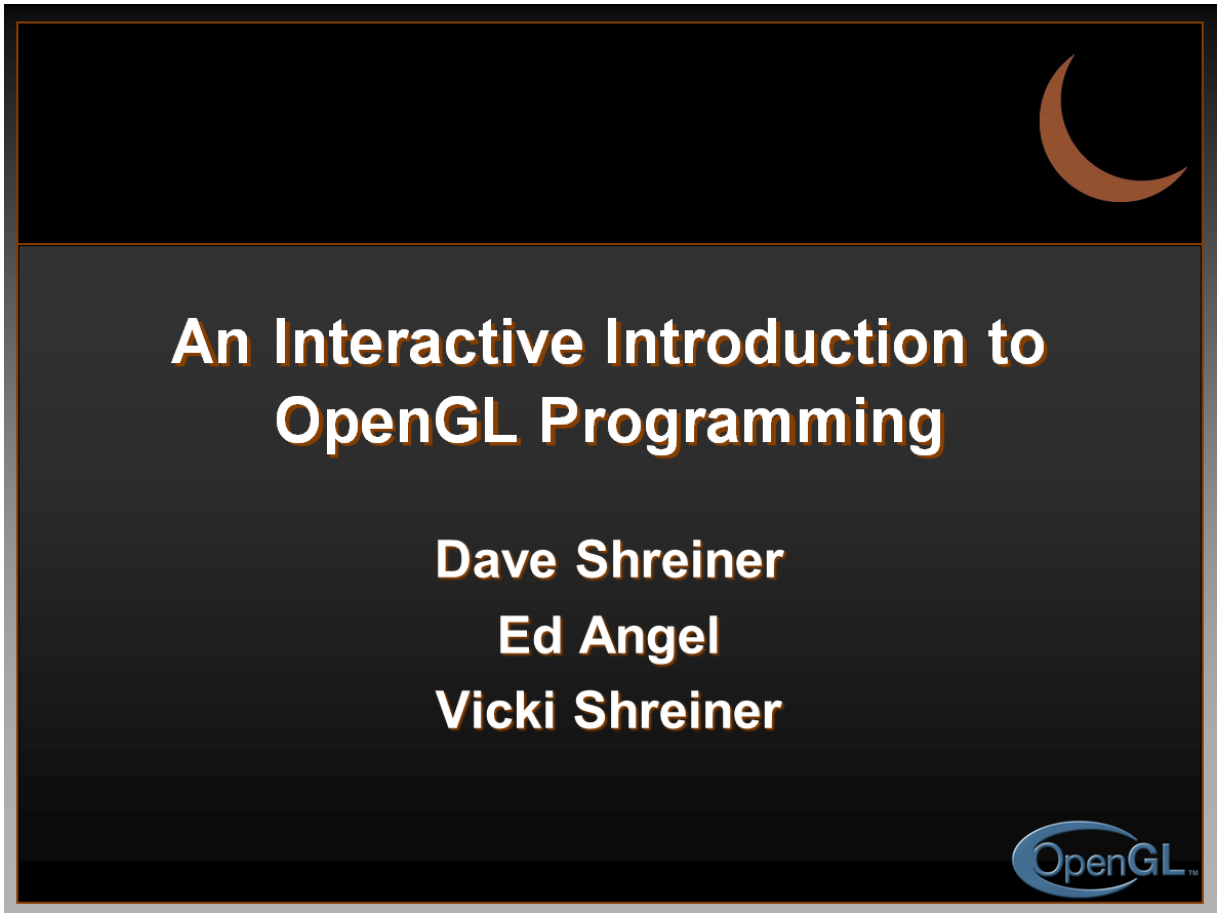
Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2020

Announcements

- Homework Project 1 due October 25
  - Start now to avoid panic mode

Introduction to OpenGL

- Some of the slides are from SIGGRAPH course:



UCSD

OpenGL and GLFW Overview

- What is OpenGL
- OpenGL in windowing systems
- Why GLFW
- A GLFW program template

What Is OpenGL?

- Graphics rendering API
  - high-quality color images composed of geometric and image primitives
  - window system independent
  - operating system independent

UCSD

OpenGL as a Renderer

- Geometric primitives
  - points, lines and polygons
- Image Primitives
  - images and bitmaps
  - separate pipeline for images and geometry
    - linked through texture mapping
- Rendering depends on state
  - colors, materials, light sources, etc.

Related APIs

- GLU (OpenGL Utility Library)
  - part of OpenGL
  - supports more complex shapes like NURBS, tessellators, quadric shapes, etc.
- GLFW (OpenGL Utility Toolkit)
  - portable windowing API
  - not part of OpenGL
  - one of many ways to create an OpenGL window

Preliminaries

- Headers Files
  - #include <GL/gl.h>
  - #include <GL/glu.h>
  - #include <GLFW/glfw3.h>
- Libraries
- Enumerated Types
  - OpenGL defines numerous types for compatibility
    - GLfloat, GLint, GLenum, etc.

GLFW Basics

- Application Structure
  - Configure and open window
  - Initialize OpenGL state
  - Enter event processing loop

Sample Program

```
#include <GLFW/glfw3.h>

int main(void)
{
    GLFWwindow* window;

    /* Initialize the library */
    if (!glfwInit()) return -1;

    /* Create a windowed mode window and its OpenGL context */
    window = glfwCreateWindow(640, 480, "Hello CSE 167", NULL, NULL);
    if (!window)
    {
        glfwTerminate();
        return -1;
    }
    /* Make the window's context current */
    glfwMakeContextCurrent(window);

    /* Initialize OpenGL here */

    /* Loop until the user closes the window */
    while (!glfwWindowShouldClose(window))
    {
        /* Render here with OpenGL */

        /* Swap front and back buffers */
        glfwSwapBuffers(window);

        /* Poll for and process events */
        glfwPollEvents();
    }
    glfwTerminate();
    return 0;
}
```

OpenGL Initialization

- Set up OpenGL states you are going to use

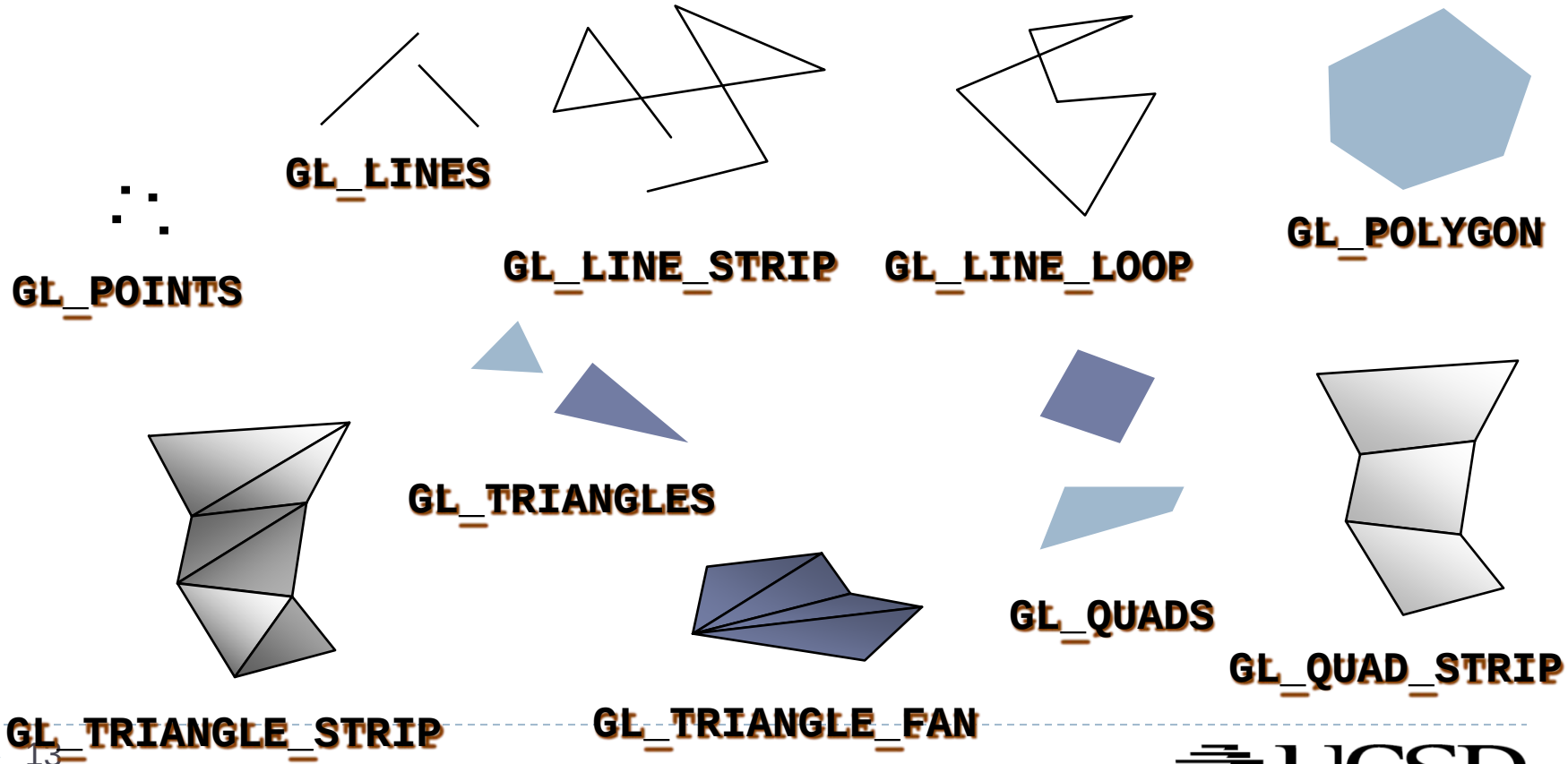
```
void init( void )
{
  glClearColor( 0.0, 0.0, 0.0, 1.0 );
  glClearDepth( 1.0 );

  glEnable( GL_DEPTH_TEST );
  glEnable( GL_CULL_FACE );
}
```

Elementary Rendering

- Geometric Primitives
- Managing OpenGL State
- OpenGL Buffers

OpenGL Geometric Primitives

- All geometric primitives are specified by vertices



UCSD

OpenGL's State Machine

- All rendering attributes are encapsulated in the OpenGL State
  - rendering styles
  - texture mapping
  - control of programmable shaders

UCSD

Manipulating the OpenGL State

- Appearance is controlled by current state

```
for each ( primitive type to render )
{
    update OpenGL state
    render primitives
}
```


UCSD

Manipulating the OpenGL State

- Setting the State

```
glPointSize( size );
glLineStipple( repeat, pattern );
```

- Enabling Features

```
glEnable( GL_DEPTH_TEST );
glDisable( GL_TEXTURE_2D );
```


UCSD

OpenGL Memory Management: Buffer Usage Hints

void glBufferData(GLenum target, GLsizeiptr size, const GLvoid * data, GLenum **usage**);

usage is a hint to the GL implementation as to how a buffer object's data store will be accessed. This enables the GL implementation to make more intelligent decisions that may significantly impact buffer object performance. It does not, however, constrain the actual usage of the data store. usage may be one of these:

- **GL_STATIC_DRAW**: The data store contents will be modified once and used many times as the source for GL drawing commands.
- **GL_DYNAMIC_DRAW**: The data store contents will be modified repeatedly and used many times as the source for GL drawing commands.

```
static void LoadTriangle()
{
    // make and bind the VAO
    glGenVertexArrays(1, &gVAO);
    glBindVertexArray(gVAO);

    // make and bind the VBO
    glGenBuffers(1, &gVBO);
    glBindBuffer(GL_ARRAY_BUFFER, gVBO);

    // Put the three triangle vertices into the VBO
    GLfloat vertexData[] = {0.0f, 0.8f, 0.0f, -0.8f,-0.8f, 0.0f, 0.8f,-0.8f, 0.0f,
    };
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertexData), vertexData, GL_STATIC_DRAW);
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, NULL);

    // unbind the VBO and VAO
    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glBindVertexArray(0);
}
```

Debugging OpenGL Code

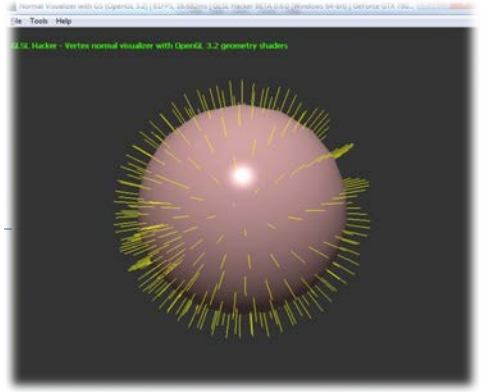
UCSD

OpenGL error state: glGetError()

- OpenGL has an error state
- Use glGetError() to find location of error. It will clear the error flag.
- Then gluErrorString() to parse the error message

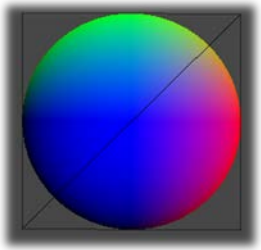
```
void printGLError(const char* msg)
{
  const GLenum err = glGetError();
  if(err != GL_NO_ERROR)
  {
    const char* str = (const char*)gluErrorString(err);
    cerr << "OpenGL error: " << msg << ", " << str << endl;
  }
}
```

Tips for Visual Debugging



Normal Visualizer

- Collisions, view frustum culling:
  - Show bounding boxes/spheres for all objects
- Problems with shading:
  - Display normal vectors on vertices as line segments pointing in the direction of the vector. Example: Normal Visualizer (pictured above).
  - Or interpret surface normals as RGB colors by shifting x/y/z range from -1..1 to 0..1.
- Display direction and other vectors:
  - Display normal vectors as described above.
- Objects don't get rendered:
  - Find out if they won't render or are just off screen by temporarily overwriting GL_MODELVIEW and GL_PROJECTION matrices with simpler ones, and/or zooming out by increasing the field of view angle.



Normal shading

OpenGL Debugging Tools

- Overview with many links:
  - https://www.opengl.org/wiki/Debugging_Tools
- Nvidia tools (Nsight Graphics and others):
  - https://developer.nvidia.com/gameworks-tools-overview

Tutorials and Documentation

- OpenGL Tutorials
  - http://www.lighthouse3d.com/tutorials/
  - http://www.tomdalling.com/blog/category/modern-opengl/
  - http://www.swiftless.com/opengl4tuts.html
- OpenGL and GLSL Specifications
  - https://www.opengl.org/registry/
- OpenGL 3.2 API Reference Card
  - http://www.opengl.org/sdk/docs/reference_card/opengl32-quick-reference-card.pdf