# CS 432 – Interactive Computer Graphics

Lecture 1 – Part 3

Anatomy of OpenGL Programs

# Reading

- Angel Chapter 2
- Red Book Chapter 4

# Include Directories

- Basic functionality of OpenGL is in the GL library
  - `gl.h`
- Additional functionalities (including matrix operations) are in the OpenGL Utility Library *GLU,* both of which are distributed with every OpenGL implementation
  - `glu.h`
- GLUT includes both `gl.h` and `glu.h` so it is suffice to just include GLUT
  - `#include <GL/glut.h>`

# Include Files

- For our purposes copy the following files into all your projects:
  - `InitShader.cpp`
    - Provides code for loading and linking shaders (more on this later)
    - Add this to each project as a source file since it needs to be compiled
  - `mat.h` and `vec.h`
    - Classes for vector and matrix objects (we'll use these a lot in graphics!)
    - Mimic the shader programming language objects
  - `CheckError.h`
    - Just some error checking stuff…
- The author of the book provided a header file that does all the system independent includes for you!
  - So you just need to do `#include "Angel.h"`
  - This includes all the other stuff you need, i.e `gl.h`, `glut.h`, `freeglut.h` etc..

# Program Structure

- Most OpenGL programs have a similar structure consisting of:
  - `main()`
    - Initialize and open one or more windows
    - Initialize glew/glut systems
    - Specify callback functions
    - Enter event loop
  - `init()`
    - Set state variables
    - Create buffers on GPU for the data
      - Put initial data on the GPU
    - Set up vertex array objects (VAOs)
    - Initialize shaders
  - Callbacks
    - Display Function
    - Input functions
    - Etc..

# GLUT and OpenGL: Main function

Initialize glut

Initialize window

Initialize glew
(don't need if on Mac)

Register
Callback
functions

Enter even loop (wait forever for
events or for program to be
killed via Cntrl-C)

```
int main(int argc, char** argv)

{
        glutInit(&argc, argv);

#ifdef __APPLE__
        glutInitDisplayMode(GLUT_3_2_CORE_PROFILE|GLUT_RGB|GLUT_SINGLE);
#else
        glutInitDisplayMode(GLUT_RGB|GLUT_SINGLE);
#endif
        glutInitWindowSize(700,700);
        glutInitWindowPosition(0,0);

        glutCreateWindow("Hello World");


#ifndef __APPLE__

        glewInit();
#endif


        init();  //additional initializations (data?)



        //at least need this callback
        glutDisplayFunc(display);

        glutMainLoop();

        return 0;
}
```

# Compatibility

- Although we'll be doing "modern shader-based" interactive graphics, Apple recognizes that a lot of the GLUT stuff is depreciated, and therefore will marks them as "depreciated".
- In the interest of providing a single cross-compatible course, that's ok.

# Compatibility

- For example, we'll force Apple to use GLUT version 3.2

```
#ifdef __APPLE__
     glutInitDisplayMode(GLUT_3_2_CORE_PROFILE|GLUT_RGB|GLUT_SINGLE);
#else
     glutInitDisplayMode(GLUT_RGB|GLUT_SINGLE);
#endif
```

- And note that new don't need GLEW if we're on a Mac

```
#ifndef __APPLE__
    glewInit();
#endif
```

# GLUT Functions

- As a summary, commonly used glut functions include:
- `glutInit` – Allow the application to get command line arguments and initialize the system.
- `glutInitDisplayMode` – Requests/sets properties for the window
  - RGB color mode
  - Buffering mode
- `glutWindowSize` – Set the window size in pixels
- `glutWindowPosition` – Where the top-left corner of the window should be.
- `glutCreateWindow` – Create a window with a title.
- `glutDisplayFunc` – Set the display callback function.
- `glutMainLoop` – Enter infinite event loop.

# Init Function

Set the clear (background) color

Load the shader programs (vertex and fragment)

Make this shader program active (on the GPU)

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

    Gluint program = InitShader("vshader.glsl", "fshader.glsl");
    glUseProgram(program);

}
```

# Display Callback Function

Erase the screen (replace with active background color)

Tell GPU to render using active requests

```
void display()
{
        glClear(GL_COLOR_BUFFER_BIT);

        glFlush();
}
```

# Event-Drive Programming

- In general, windows based and interactive programs are *event driven*
- This means they *wait* for something to happen then *do* something
- Typically events include:
  - Mouse click
  - Keyboard pressed
  - Window resized
  - Timeout occurred
- For each event we want to react to we must **create** and **register** a callback function

# Common Callback Functions

| Event | Callback |
|---|---|
| Rendering | void glutDisplayFunc(void (*func) (void)); |
| Reshape | void  glutReshapeFunc(void (*func)(int w, int h)); |
| Mouse | void glutMouseFunc(void (*func)(int b, int state, int x, int y)); |
| Keyboard | void glutKeyboardFunc(void (*func)(unsigned char key, int x, int y)); |
| Idle | void glutIdleFunc(void (*func) (void)); |
| Motion | void glutMotionFunc(void (*func)(int x, int y)) |
| Passive Motion | void glutMotionFunc(void (*func)(int x, int y)) |
| Timer | void glutTimerFunc(int usec, void (*func)(int val, val)); |

# More Callbacks

- https://www.opengl.org/resources/libraries/glut/spec3/node45.html