6.S096 Lecture 9 – Visualization
OpenGL, Makefiles, Large Projects

Andre Kessler
What is OpenGL?

The standard for most 2D/3D graphics rendering today.

http://www.opengl.org/

- Highly cross-platform (between OS, architecture, etc)
- Everything from decade-old computers to mobile devices today.
- An abstract API for drawing; bindings based in C
- Interface with the GPU graphics pipeline.
How do we get it?

There are a lot of really old (harmful!) tutorials out there.

These are good ones:

- WikiBooks
How will we use it?

I’ve written some wrappers for the initialization (both general GL and glut).

Let’s look at the code (GlutWrapper.h)
OpenGL Display Function (jumping ahead)

```c
glBindBuffer( GL_ARRAY_BUFFER, _positionBufferObject );  
glBufferSubData( GL_ARRAY_BUFFER, 0,  
    sizeof( float ) * _bufSize, _buf );  
glBindBuffer( GL_ARRAY_BUFFER, 0 );  

glClearColor( 0.0f, 0.0f, 0.0f, 0.0f );  
glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
```
glUseProgram(_program);
glBindBuffer(GL_ARRAY_BUFFER, _positionBufferObject);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, 0);

glDrawArrays(GL_TRIANGLE_STRIP, 0, (GLsizei)_bufSize);

glDisableVertexAttribArray(0);
glUseProgram(0);

glutSwapBuffers();

 glutPostRedisplay();
OpenGL Initialization

```c
glutInit( &argc, argv );

uint32_t displayMode = GLUT_DOUBLE
| GLUT_ALPHA
| GLUT_DEPTH
| GLUT_STENCIL;

glutInitDisplayMode( displayMode );

// We’ll be using OpenGL 3.0
glutInitContextVersion( 3, 0 );

// We’ll be using OpenGL 3.0
```
OpenGL: Buffer Objects

```c
glGenBuffers( 1, &_positionBufferObject );
glBindBuffer( GL_ARRAY_BUFFER, _positionBufferObject );
//..etc
glBufferData( GL_ARRAY_BUFFER, 4 * bufSize, _buf, GL_STATIC_DRAW );
glBindBuffer( GL_ARRAY_BUFFER, 0 );
```
More in the code...

Let’s look into the code...
Components

Requirements

25% **Physics Engine** - quality and extensibility of simulation code
25% **Visualization** - OpenGL; getting a good visualization working
15% **Unit testing** - gtest, quality and coverage of tests
15% **Software Process** - code reviews, overall integration of project
10% **Interactive** - user interactivity with simulation (keyboard, mouse, etc)
10% **Do something cool** - make it look cool, add a useful feature, do something interesting!

Extra 5% available in all areas for exceptional effort.
Physics Engine Inaccuracies

Your integrator should be improving on the basic; this is what the basic one does:
Binary Star System Example 1
Binary Star System Example 2
Binary Star System Example 3
Binary Star System Example 4
Binary Star System Example 5

The image shows a graph using Gnuplot with two datasets from a file named "logfile.txt". The graph plots two sets of data points, one in red and one in green, which represent the positions of stars in a binary system. The data points are plotted on a Cartesian coordinate system with axes ranging from -0.6 to 0.6 on both the x and y axes. The data points are marked with crosses, indicating their positions on the graph.
OpenGL

Binary Star System Example 6

"logfile.txt" using 1:2
"logfile.txt" using 4:5

-0.531909, 0.657656
Binary Star System Example 7
Binary Star System Example 8
Binary Star System Example 9
Binary Star System Example 10
OpenGL Visualization

OpenGL!

Courtesy of Aaron M. Geller. Used with permission.
Content Provided: Reminder

Vector3.h

So that you don’t have to write (all) of your own vector math, feel free to use the header available.

It’s a templated 3-d vector class that can be widely useful and is guaranteed fast ("plain old data type")
template<typename T>
class Vector3 {
    T _x, _y, _z;

public:
    Vector3() : _x{}, _y{}, _z{} {}  
    Vector3( T x_, T y_, T z_ ) :  
        _x{x_}, _y{y_}, _z{z_} {}  
    inline T x() const { return _x; }  
    inline T y() const { return _y; }  
    inline T z() const { return _z; }  
    T norm() const;  
    T normsq() const;  
};
Reminder: the compilation process

1. Preprocess
2. Compile
3. Link
Code Reviews: what you send to me

- Your name and the name of the person whose code you are reviewing.
- The snippet of code you are reviewing: more than 30 lines, less than 100.
- Your comments interspersed in their code.
- A summary of main points relating to the review (what they did well, major areas for improvement, common issues, general observations).
Code Reviews: what you send to me

- Your name and the name of the person whose code you are reviewing.
- The snippet of code you are reviewing: more than 30 lines, less than 100.
- Your comments interspersed in their code.
- A summary of main points relating to the review (what they did well, major areas for improvement, common issues, general observations).

You should choose a bite-sized chunk that will take you 45 mins to 1 hour to fully review.
Let’s see some examples...
Wrap-up & Friday

Final project due Saturday.

Send me your code reviews *tonight* please!

Class on Fri.

- Grab-bag: coding interviews, general perspective
- Bring all your C++ questions!

Questions?

- Lab today
- We’ll be covering more OpenGL and helping out with projects.