OPENGL AND GLSL

Computer Graphics

OUTLINE

I. Detecting GLSL Errors
II. Drawing a (gasp) Triangle!
III. (Simple) Animation

Interactive Computer Graphics, http://www.mechapen.com/projects.html

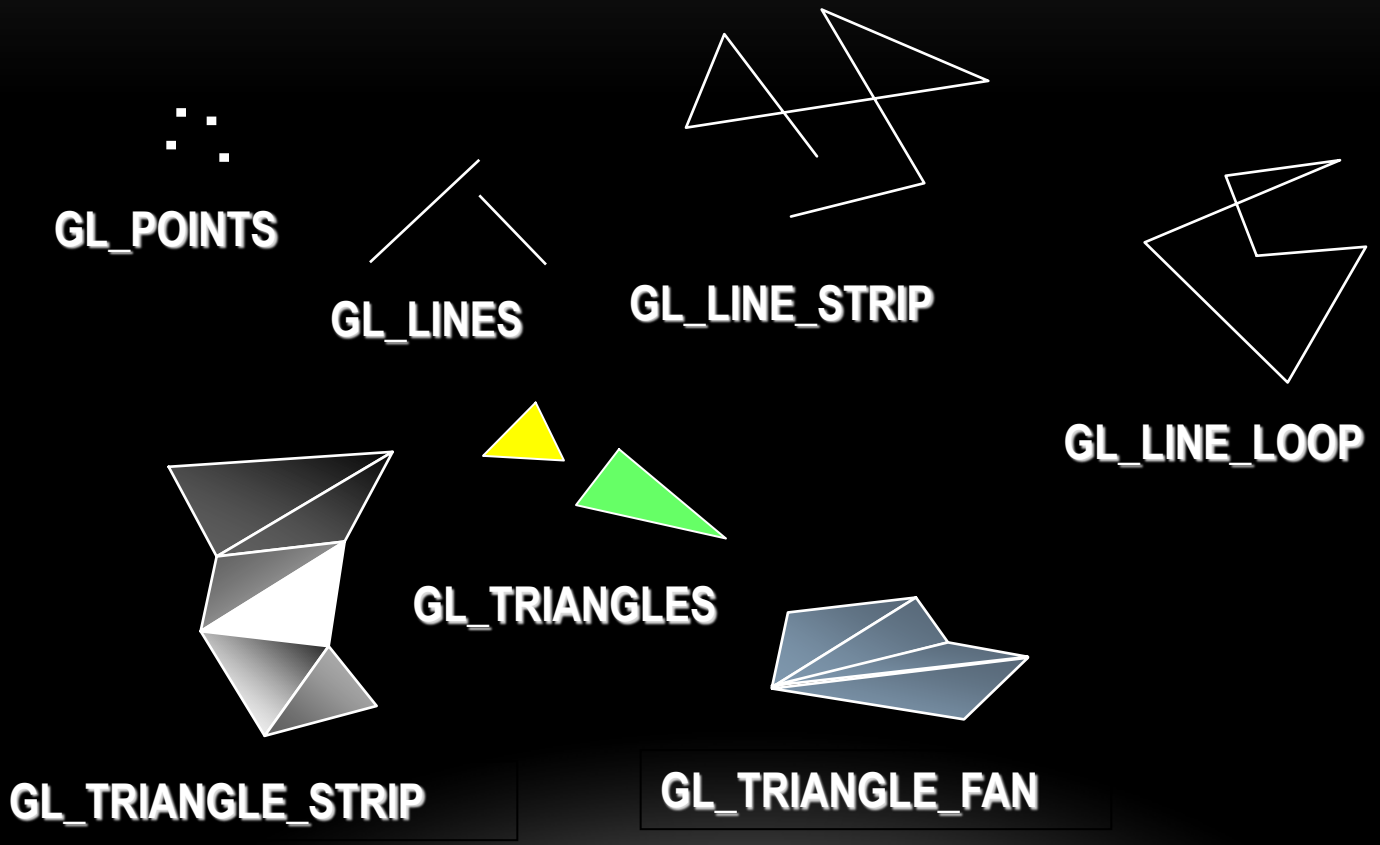
WHAT IS OPENGL?

- Software Interface to Graphics Hardware
  - V4.3 has over 500 commands
- Hardware Independent
  - No Commands for:
    - Windowing Tasks
    - User Input
    - High-Level Geometric Shapes (must use primitives)
- Rendering
  - Construct Shapes from Primitives
  - Arrange Shapes in 3D Space (can change viewpoint)
  - Calculate Object Color (from color, light, texture)
  - Convert to Image on Screen (Rasterization)

OPENGL COMMAND SYNTAX

- Start with gl, use initial capital letters for each word
  - glClearColor()
  - Note: since we are using JOGL:
    - We declare a GL4 object (named gl in the sample code, you can name it what you want)
    - All openGL calls are preceded by the object name
      - e.g. gl.glDrawArrays(…)
- Constants start with GL_ and use all caps and underscores

OPENGL PRIMITIVES



EVENT LOOP

- Every program that implements the GLEventListener:
  - Must have an init callback
    - The init callback is executed once when the scene/graphics are made visible
    - It generally will deal with the shader code and loading 3D models
  - Must have a display callback
    - The display callback is executed whenever OpenGL decides the display must be refreshed, for example when the window is opened
  - Must define a resize() and dispose() callback
    - resize() is called when the window size is changed
    - dispose() is called when the window is … well, disposed of

OPENGL ERRORS

- When we run OpenGL code, we are several levels away from pieces of the actual code
  - OpenGL functions are in C, and we're using an interface to those calls
  - GLSL code is compiled and linked while our Java program is running
    - And when the code runs, it is on the GPU, not CPU, so OS can't catch errors
    - If the GLSL code fails at any point, it doesn't cause the Java/JOGL code to stop
- This makes for difficult debugging
  - Often, the only thing that happens is… nothing
    - Nothing displays, and no errors are reported

OPENGL ERRORS

- Book provides code for three methods to check for errors:
  - checkOpenGLError
    - checks the OpenGL error flag for errors
  - printShaderLog
    - displays OpenGL log on GLSL compilation failure
  - printProgramLog
    - displays OpenGL log on GLSL linking failure
- Debug and trace capability available also
  - Use runtime command line options:
    - -Djogl.debug.DebugGL
    - -Djogl.debug.TraceGL
  - These provide standard output of errors and a display of all OpenGL calls made
  - Help to isolate the occurrence of bugs

DRAWING A (GASP) TRIANGLE

- For now, we will do this in the vertex shader
  - (Later we will do this in the Java/JOGL code)

```
#version 430
void main(void)
{ if (gl_VertexID == 0) gl_Position = vec4( 0.25,-0.25, 0.0, 1.0);
  else if (gl_VertexID == 1) gl_Position = vec4(-0.25,-0.25, 0.0, 1.0);
  else gl_Position = vec4( 0.25, 0.25, 0.0, 1.0);
}
```

  - And in Java – the call to drawArrays – becomes:

```
gl.glDrawArrays(GL_TRIANGLES, 0, 3);
```

(SIMPLE) ANIMATION

- **Motion = Redraw + Swap**
- **Flipbook**
- In a movie theater, motion is achieved by taking a sequence of pictures and projecting them at 24 per second on the screen.
- Each frame is moved into position behind the lens, the shutter is opened, and the frame is displayed. The shutter is momentarily closed while the film is advanced to the next frame, then that frame is displayed, and so on.
- Although you're watching 24 different frames each second, your brain blends them all into a smooth animation.
- Computer-graphics screens typically refresh (redraw the picture) approximately 60 to 76 times per second, and some even run at about 120 refreshes per second.
- Clearly, 60 per second is smoother than 30, and 120 is marginally better than 60. Refresh rates faster than 120, however, are beyond the point of diminishing returns, since the human eye is only so good.

ANIMATION SPEED MISMATCH

Porsche Dyno Test

(SIMPLE) ANIMATION

- The key reason that motion picture projection works is that each frame is complete when it is displayed.
- Suppose you try to do computer animation of your million-frame movie with a program like this:

```
open_window_in_double_buffer_mode();
for (i = 0; i < 1000000; i++) {
        clear_the_window();
        draw_frame(i);
        swap_the_buffers();
}
```

- Most OpenGL implementations provide double-buffering - hardware or software that supplies two complete color buffers. One is displayed while the other is being drawn

(SIMPLE) ANIMATION

- Assuming that your system refreshes the display 60 times per second, the fastest frame rate you can achieve is 60 frames per second (*fps*)
  - If all your frames can be cleared and drawn in under 1/60 second, your animation will run smoothly
  - If too complicated to draw in 1/60 second, each frame is displayed more than once.
    - If it takes 1/45 second to draw a frame, you get 30 fps, and the graphics are idle for 1/30-1/45=1/90 second per frame, one-third of the time.
- If the scene's complexity is close to any of the magic times (1/60 second, 2/60 second, 3/60 second, and so on in this example), then because of random variation, some frames go slightly over the time and some slightly under. Then the frame rate is irregular, which can be visually disturbing.
- Swap functionality is in the windowing library, not in OpenGL.

ANIMATING THE (GASP) TRIANGLE

- Add a line to imports:

```
import com.jogamp.opengl.util.*;
```

- Add private instance variables for location and offset on each move:

```
private float x = 0.0f;
private float offset = 0.01f;
```

- Create a new FPSAnimator and start it:
  - Note: FPS is frames per second, not first person shooter
  - FPSAnimator will attempt to redisplay n times per second

```
FPSAnimator animtr = new FPSAnimator(myCanvas, 50);
animtr.start();
```

ANIMATING THE (GASP) TRIANGLE

- That was easy! Is that all?
  - No…
  - Something (location) needs to change in our display() method, and we need to pass that to the vertex shader

```
x += offset;

if (x>1.0f) offset = -0.01f;

if (x<-0.01f) offset = 0.01f;
```

  - Now for the fun part:

```
int offset_loc =
gl.glgetUniformLocation(rendering_program, “offset”;

gl.glProgramUniform1f(rendering_program, offset_loc, x);
```

ANIMATING THE (GASP) TRIANGLE

- And the new vertex shader:

```
#version 430
uniform float offset;
void main(void)
{ if (gl_VertexID == 0) gl_Position = vec4( 0.25 +
offset,-0.25, 0.0, 1.0);
  else if (gl_VertexID == 1) gl_Position = vec4(-0.25 +
offset,-0.25, 0.0, 1.0);
  else gl_Position = vec4( 0.25 + offset, 0.25, 0.0,
1.0);
}
```

SUMMARY

I. Detecting GLSL Errors
II. Drawing a (gasp) Triangle!
III. (Simple) Animation