VR Graphics Programming for, well, you know

Some (relatively) easy steps to virtual reality, Part 2

Tom Sgouros

Center for Computation and Visualization
Brown University
thomas_sgouros@brown.edu

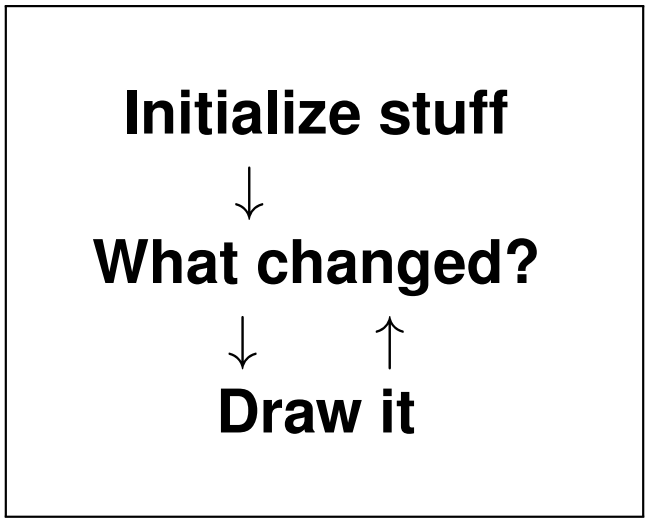
Spring 2018

Program structure

- Event loop
  - Event handler
  - Animation
  - Rendering
- Shaders

Event loop

Basic real-time programming structure:




```
...
scene.addObject(axesSet);

scene.setLookAtPosition(glm::vec3(0.0f, 0.0f, 0.0f));
scene.setCameraPosition(glm::vec3(1.0f, 2.0f, 7.5f));
scene.prepare();

glutMainLoop();
return(0);
```

Different kinds of input

Synchronous: On my schedule.

Prompt for input and, um, prompt for input. Also sometimes polled input.

Asynchronous: On your schedule.

Mouse movements, head movements, wand movements, button presses, typing, gesture recognition. . .

a/k/a *events*, *interrupts*.

Event handler

Handles asynchronous input.

```
void processNormalKeys(unsigned char key, int x, int y) {

  float step = 0.5f;

  switch (key) {
  case 27:    exit(0);
  case 'a':
    scene.addToCameraPosition(glm::vec3(-step, 0.0f, 0.0f));
    break;
  case 'q':
    scene.addToCameraPosition(glm::vec3( step, 0.0f, 0.0f));
    break;
  case 's':
    ...
```

Animation

Change what is seen

Motion in space, motion within scene component, model changing shape

Change how it is seen

Camera motion, lighting change

Render function

Combines animation and drawing stuff.

```
void renderScene() {
  ...
  glm::vec3 pos = tetrahedron->getPosition();
  oscillator += oscillationStep;
  pos.x = sin(oscillator);
  pos.y = 1.0f - cos(oscillator);
  tetrahedron->setPosition(pos);

  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  scene.load();
  scene.draw(scene.getViewMatrix(), scene.getProjMatrix());

  glutSwapBuffers();
```

Shaders

OpenGL innovation, historically related to the transformation of graphics cards to GPUs. Looks C-ish. Biggest problem is the lack of clarity about inputs and outputs.

```
#version 120

uniform mat4 projMatrix, viewMatrix, modelMatrix;
attribute vec4 position, color;
varying vec4 colorFrag;

void main()
{
  colorFrag = color;
  gl_Position = projMatrix * viewMatrix * modelMatrix * positio
}
```

Shaders

OpenGL innovation, historically related to the transformation of graphics cards to GPUs. Looks C-ish. Biggest problem is the lack of clarity about inputs and outputs. Slightly improved in OpenGL 3.

```
#version 330

uniform mat4 projMatrix, viewMatrix, modelMatrix;
layout(location=0) in vec4 position;
layout(location=1) in vec4 color;
out vec4 colorFrag;

void main()
{
  colorFrag = color;
  gl_Position = projMatrix * viewMatrix * modelMatrix * positio
}
```

Prepare a shader

A shader has to be given an ID, compiled and then a “program” is created and the shader(s) are linked to it. Here’s the vertex shader. The other shaders would be virtually the same.

```
shaderMgr::compileShaders() {...
  shaderID = glCreateShader(GL_VERTEX_SHADER);
  const char* vs = shaderText.c_str();
  glShaderSource(shaderID, 1, &vs, NULL);
  glCompileShader(shaderID);
  errorLog = _getShaderInfoLog(shaderID);
  if (errorLog.size() > 1)
    std::cerr << "** Vertex compile error in " ...;
  programID = glCreateProgram();
  glAttachShader(programID, shaderID);
  glLinkProgram(programID);
  ...
```

Shader input

Then you have to make a buffer in your program correspond to an attribute name in the shader.

```
glUseProgram(programID);
attribID = glGetAttribLocation(programID, attribName.c_str())

glGenBuffers(1, &attribBufferID);

glEnableVertexAttribArray(attribID);
glBindBuffer(GL_ARRAY_BUFFER, attribBufferID);
glBufferData(GL_ARRAY_BUFFER, attribLengthInBytes, attrib[0],
             GL_STATIC_DRAW);
```

More shader input

Then you have to make a buffer in your program correspond to a uniform name in the shader.

```
glUseProgram(programID);
unifID = glGetUniformLocation(programID, unifName.c_str());

glUniformMatrix4fv(unifID, 1, false, &unifMatrix[0][0]);
```

Draw it!

Bind the buffer, say what's in it, draw it.

```
glBindBuffer(GL_ARRAY_BUFFER, verticesBufferID);
glVertexAttribPointer(verticesID, verticesComponentsPerVertex
                      GL_FLOAT, 0, 0, 0);

glBindBuffer(GL_ARRAY_BUFFER, colorsBufferID);
glVertexAttribPointer(colorsID, colorsComponentsPerVertex,
                      GL_FLOAT, 0, 0, 0);

glDrawArrays(drawType, 0, verticesCount);
```

Draw with more complicated buffers

Buffers can also have interleaved data.

```
glBindBuffer(GL_ARRAY_BUFFER, interleavedBufferID);

glVertexAttribPointer(verticesID, 3,
                      GL_FLOAT, GL_FALSE, stride,
                      BUFFER_OFFSET(0));
glVertexAttribPointer(colorsID, 3,
                      GL_FLOAT, GL_FALSE, stride,
                      BUFFER_OFFSET(3));
glDrawArrays(drawType, 0, verticesCount);
```

Using BSG, step 1

Deal with shaders

```
bsg::bsgPtr<bsg::lightList> lights = new bsg::lightList();
lights->addLight(glm::vec4(0.0f, 0.0f, 3.0f, 1.0f));

bsg::bsgPtr<bsg::shaderMgr> shader = new bsg::shaderMgr();
shader->addLights(lights);
shader->addShader(bsg::GLSHADER_VERTEX, vertexFile);
shader->addShader(bsg::GLSHADER_FRAGMENT, fragmentFile);
shader->compileShaders();

bsg::bsgPtr<bsg::textureMgr> texture = new bsg::textureMgr();
texture->readFile(bsg::texturePNG, "../data/gladiolas-sq.png"
shader->addTexture(texture);
```

Using BSG, step 2

Deal with objects, build the scene.

```
bsg::drawableRectangle* bigRectangle, smallRectangle;
bsg::drawableCollection* rectGroup;

bigRectangle = new bsg::drawableRectangle(shader, 9.0f, 9.0f,
smallRectangle = new bsg::drawableRectangle(shader, 3.0f, 5.0

smallRectangle->setPosition(1.0f, 1.0f, 0.5f);

rectGroup = new bsg::drawableCollection("rectangles");

rectGroup->addObject("big", bigRectangle);
rectGroup->addObject("small", smallRectangle);

scene.addObject(rectGroup);
```

Using BSG, step 3

Finish up.

```
axes = new bsg::drawableAxes(axesShader, 100.0f);
scene.addObject(axes);

scene.setLookAtPosition(glm::vec3(0.0f, 0.0f, 0.0f));
scene.setCameraPosition(glm::vec3(1.0f, 2.0f, 7.5f));

scene.prepare();

glutMainLoop();
```

Admire the result

Sigh with relief.