2D Triangle Translation Without GLM

Table of Contents

Chapter 1. 2D Triangle Translation Without GLM

1.1. Summary

This example introduces how to move geometry (triangles) in OpenGL. It renders a single triangle with the position of the triangle changing over time.

1.1.1. translationVector (Vertex Shader)

Changing where geometry (triangles) appears on the screen is done by transforming the vertices for the triangle. This should be done in the Vertex Shader - it is the main job, or use, of the Vertex Shader. In order to transform a triangle we should apply the same transformation to all the vertices. In the case of simple 2D translation, we just need to alter the position of all the vertice by the same amount. We can use a Uniform variable for this - we'll call it `translationVector`. We'll initialise it to a value of (-0.5, -0.5), so that we can see that the triangle is moved.

```
#version 330
in vec2 position;
in vec4 vertexColor;
out vec4 fragmentColor;
uniform vec2 translationVector = vec2(-0.5, -0.5);

void main()
{
  gl_Position = vec4(position + translationVector, 0.0, 1.0);
  fragmentColor = vertexColor;
}
```

1.1.2. translationVector (C++)

If you run the program with above Vertex Shader, you'll see the triangle appears in a new position. We now need to control that position from C++. We need a number of things:

1. a C++ variable to store `translationVector`
2. a C++ variable to store `translationVelocityVector` - how fast the translation should change

3. a C++ variable to store the location of `translationVector` in the GLSL program, filled appropriately once the GLSL program has been compiled.
4. update the C++ `translationVector` over time
5. set the `translationVector` uniform on our program every frame

C++ variable to store `translationVector` and `translationVelocityVector`

We'll just use a `GLfloat` array (of length two), for the X and Y components, for each.

```
//the translation vector we'll pass to our GLSL program
GLfloat translationVector[] = { -0.5f, -0.5f };
GLfloat translationVelocityVector[] = { 0.1f, 0.1f};
```

C++ variable to store the location of `translationVector` in the GLSL program

We just need an `int` for storage, and to ask OpenGL for the value.

```
//our GL and GLSL variables
GLuint theProgram; //GLuint that we'll fill in to refer to the GLSL
 program (only have 1 at this point)
GLint positionLocation; //GLuint that we'll fill in with the location of
 the `position` attribute in the GLSL
GLint vertexColorLocation; //GLuint that we'll fill in with the location
 of the `vertexColor` attribute in the GLSL
GLint translationVectorLocation;

GLuint vertexDataBufferObject;
GLuint vertexArrayObject;
```

```
void initializeProgram()
{
 std::vector<GLuint> shaderList;

 shaderList.push_back(createShader(GL_VERTEX_SHADER,
 loadShader("vertexShader.glsl")));
 shaderList.push_back(createShader(GL_FRAGMENT_SHADER,
 loadShader("fragmentShader.glsl")));
```

```
 theProgram = createProgram(shaderList);
 if (theProgram == 0)
 {
  cerr << "GLSL program creation error." << std::endl;
  SDL_Quit();
  exit(1);
 }
 else {
  cout << "GLSL program creation OK! GLUint is: " << theProgram <<
 std::endl;
 }

 positionLocation = glGetAttribLocation(theProgram, "position");
 vertexColorLocation = glGetAttribLocation(theProgram, "vertexColor");

 translationVectorLocation =
 glGetUniformLocation(theProgram, "translationVector");

 //clean up shaders (we don't need them anymore as they are no in
 theProgram
 for_each(shaderList.begin(), shaderList.end(), glDeleteShader);
}
```

Update the C++ `translationVector` over time

Our main loop calls updateSimulation every frame. How long we should simulate for is a parameter for this function, we is presently defaulting to 0.02 seconds. At some point we should calculate this correctly. For now, we are just assuming (hoping) that our frame rate is around 50 FPS.

```
void updateSimulation(double simLength = 0.02) //update simulation with an
 amount of time to simulate for (in seconds)
{
 //WARNING - we should calculate an appropriate amount of time to simulate
 - not always use a constant amount of time
   // see, for example, http://headerphile.blogspot.co.uk/2014/07/part-9-
no-more-delays.html

 translationVector[0] += (float)simLength * translationVelocityVector[0];
 translationVector[1] += (float)simLength * translationVelocityVector[1];

}
```

as `translationVector` is an array, we need to change both the 0th and 1st value.

Set the `translationVector` uniform on our program every frame

```
void render()
{
 glUseProgram(theProgram); //installs the program object specified by
 program as part of current rendering state

 //load data to GLSL that **may** have changed
 glUniform2f(translationVectorLocation, translationVector[0],
 translationVector[1]);

 glBindVertexArray(vertexArrayObject);

 glDrawArrays(GL_TRIANGLES, 0, 3); //Draw something, using Triangles, and
 3 vertices - i.e. one lonely triangle

 glBindVertexArray(0);

 glUseProgram(0); //clean up

}
```