Computer Science Department

Sam Houston State University

COSC 4332 Computer Graphics

Modern OpenGL Transformations The Analog Clock Example

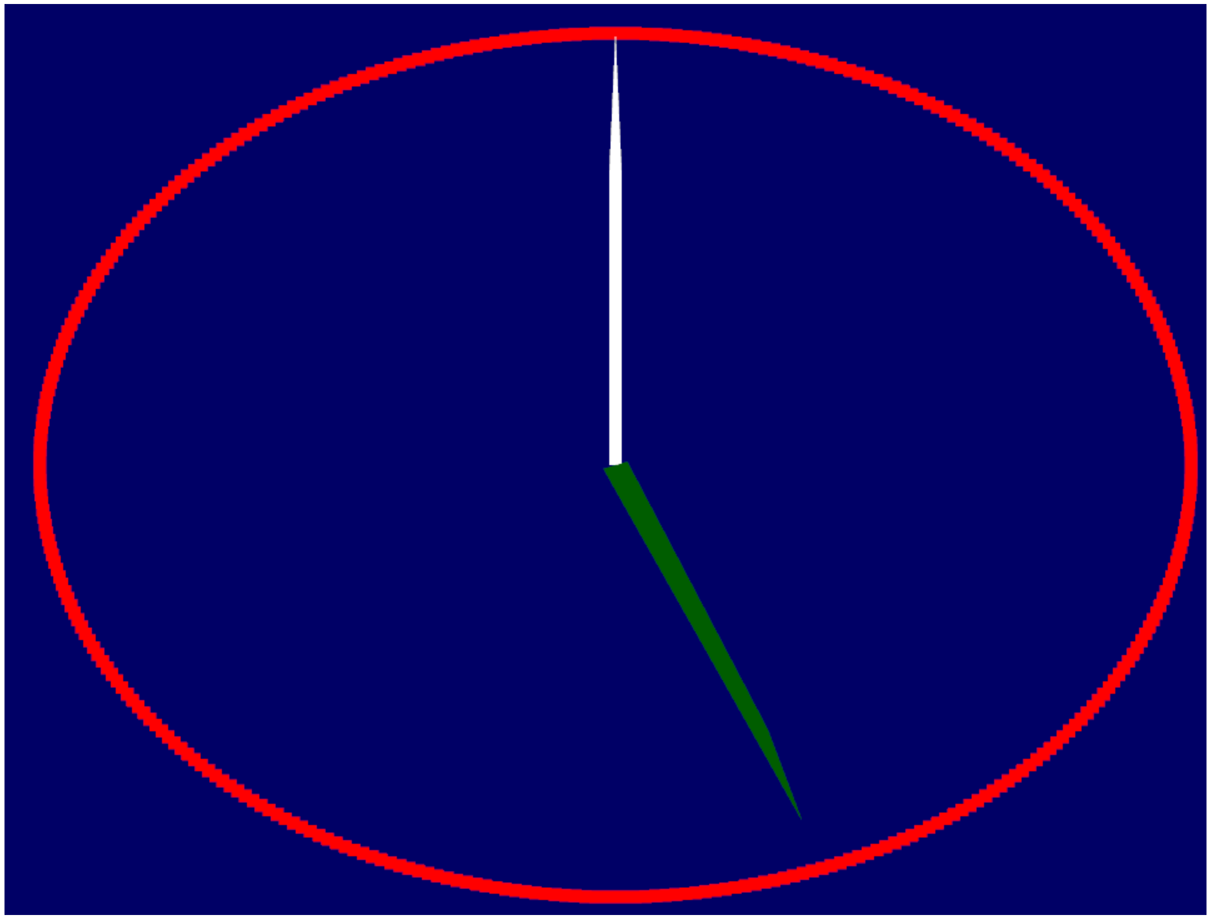
Dr. Khaled Rabieh

Outline

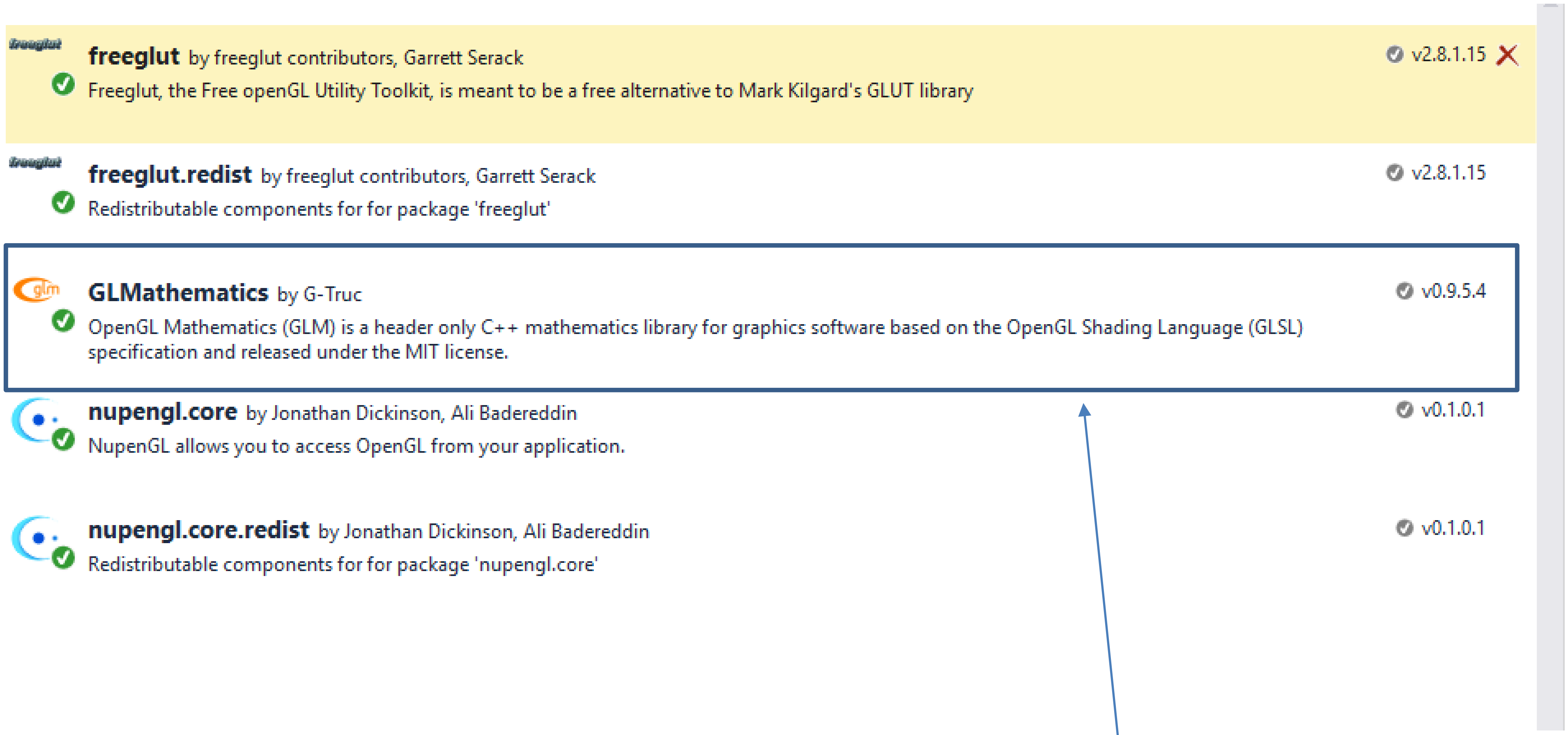
1. **Modern OpenGL Transformations**
2. Controlling the behavior of shaders at runtime.
3. Introducing uniforms

The Analog Clock

- The Frame
- The Polygons (hands)
- 2D Transformations



Needed Packages



freeglut by freeglut contributors, Garrett Serack
Freeglut, the Free openGL Utility Toolkit, is meant to be a free alternative to Mark Kilgard's GLUT library
v2.8.1.15

freeglut.redist by freeglut contributors, Garrett Serack
Redistributable components for for package 'freeglut'
v2.8.1.15

GLMathematics by G-Truc
OpenGL Mathematics (GLM) is a header only C++ mathematics library for graphics software based on the OpenGL Shading Language (GLSL) specification and released under the MIT license.
v0.9.5.4

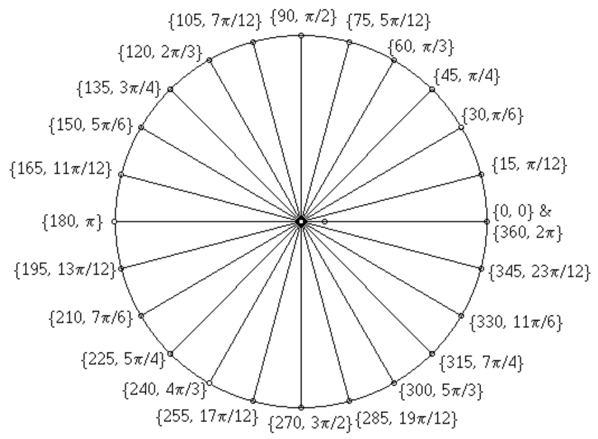
nupengl.core by Jonathan Dickinson, Ali Badereddin
NupenGL allows you to access OpenGL from your application.
v0.1.0.1

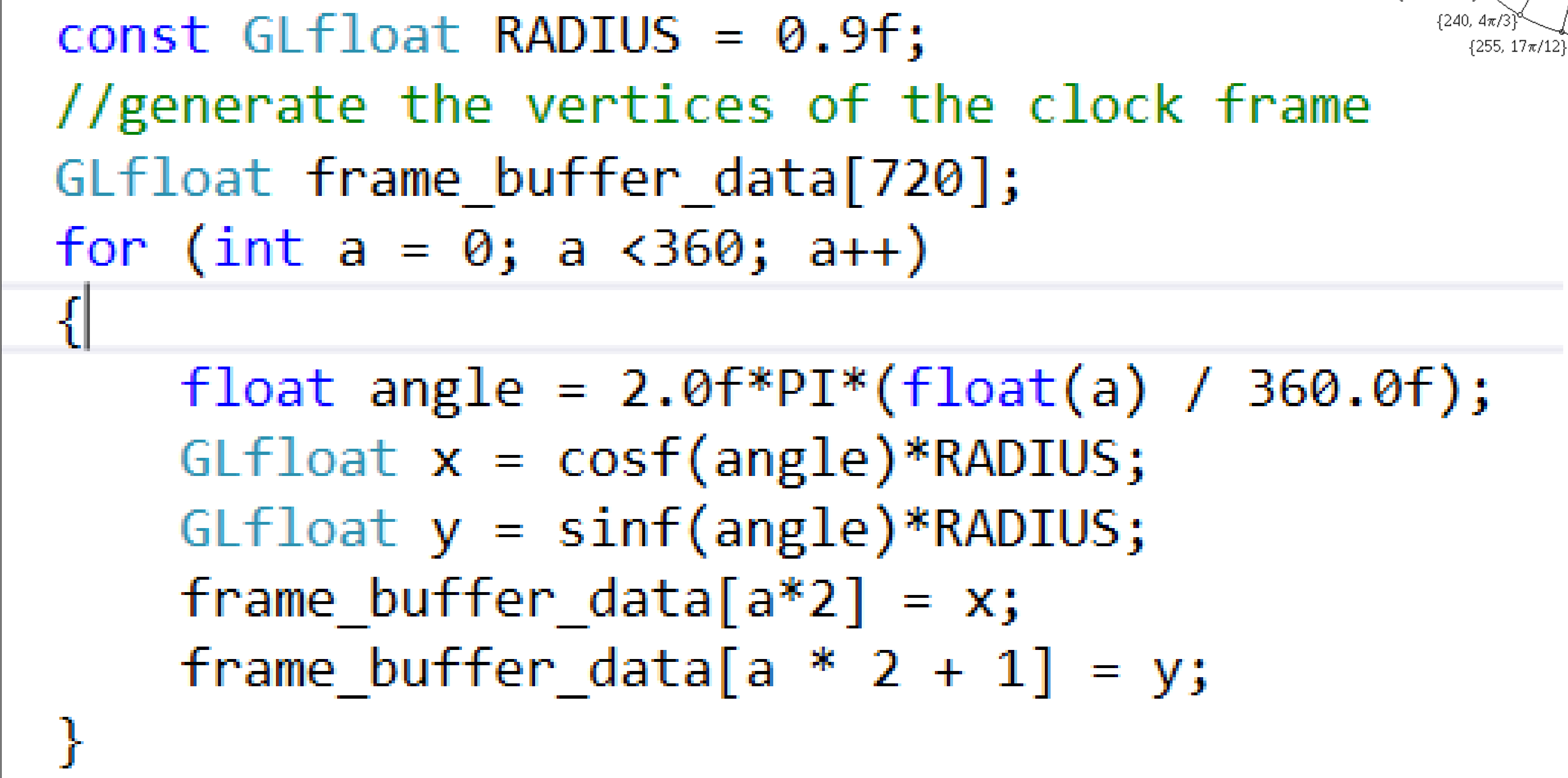
nupengl.core.redist by Jonathan Dickinson, Ali Badereddin
Redistributable components for for package 'nupengl.core'
v0.1.0.1

Needed for transformations

The Frame of the Clock

- Radius = 0.9
- An array of 2D vertices




```
const GLfloat RADIUS = 0.9f;
//generate the vertices of the clock frame
GLfloat frame_buffer_data[720];
for (int a = 0; a <360; a++)
{
    float angle = 2.0f*PI*(float(a) / 360.0f);
    GLfloat x = cosf(angle)*RADIUS;
    GLfloat y = sinf(angle)*RADIUS;
    frame_buffer_data[a*2] = x;
    frame_buffer_data[a * 2 + 1] = y;
}
```

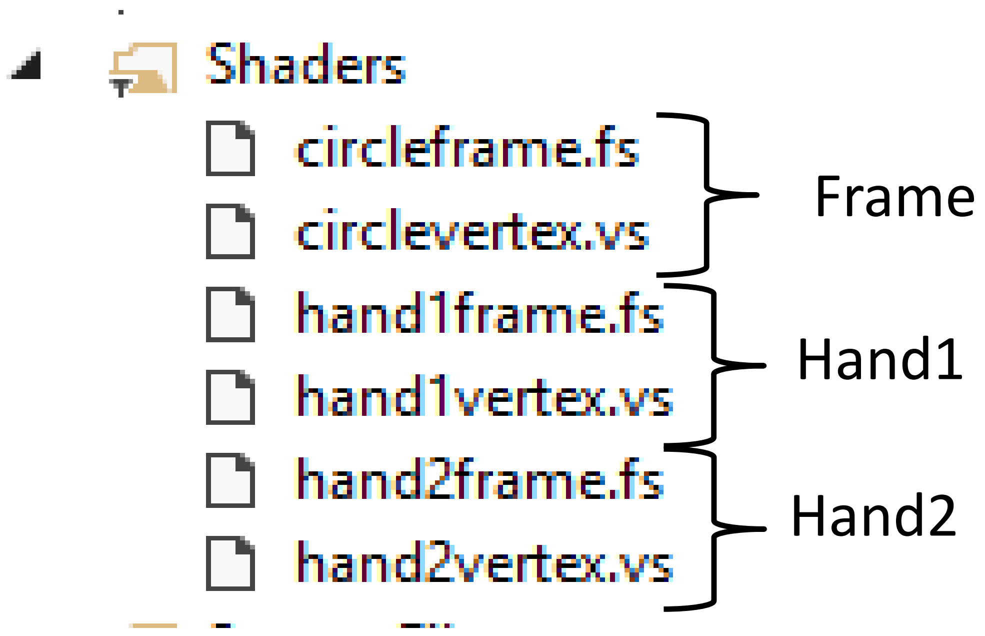
The Hands of the Clock

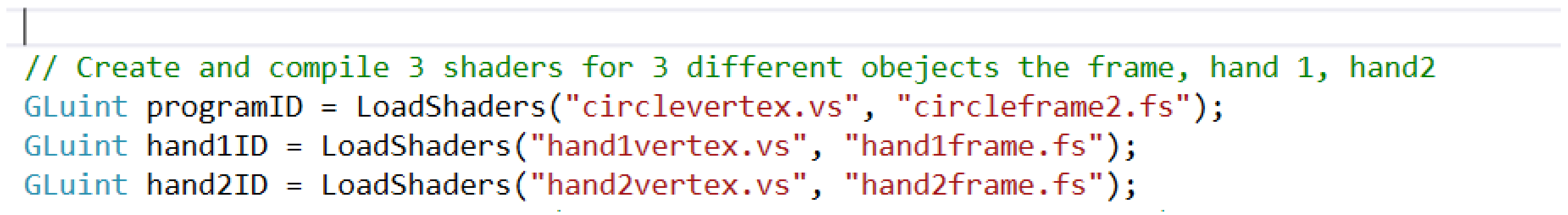
```
// the seconds hand
GLfloat hand1_buffer_data[10];
hand1_buffer_data[0] = -0.01;
hand1_buffer_data[1] = 0;
hand1_buffer_data[2] = 0.01;
hand1_buffer_data[3] = 0;
hand1_buffer_data[4] = 0.01;
hand1_buffer_data[5] = 0.6;
hand1_buffer_data[6] = 0;
hand1_buffer_data[7] = 0.9;
hand1_buffer_data[8] = -0.01;
hand1_buffer_data[9] = 0.6;

// the minutes hand
GLfloat hand2_buffer_data[10];
hand2_buffer_data[0] = -0.02;
hand2_buffer_data[1] = 0;
hand2_buffer_data[2] = 0.02;
hand2_buffer_data[3] = 0;
hand2_buffer_data[4] = 0.02;
hand2_buffer_data[5] = 0.02;
hand2_buffer_data[6] = 0;
hand2_buffer_data[7] = 0.8;
hand2_buffer_data[8] = -0.02;
hand2_buffer_data[9] = 0.6;
```

- Each Polygon (hand) consists of 5 vertices
- The vertex array have 10 entries

Compiling necessary Shaders

Every object have at least two shaders

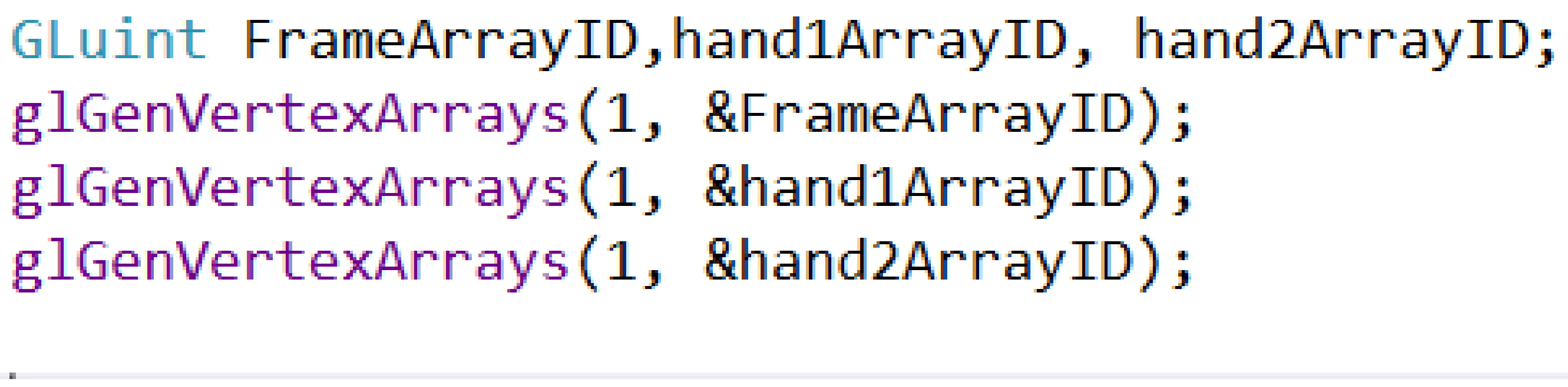




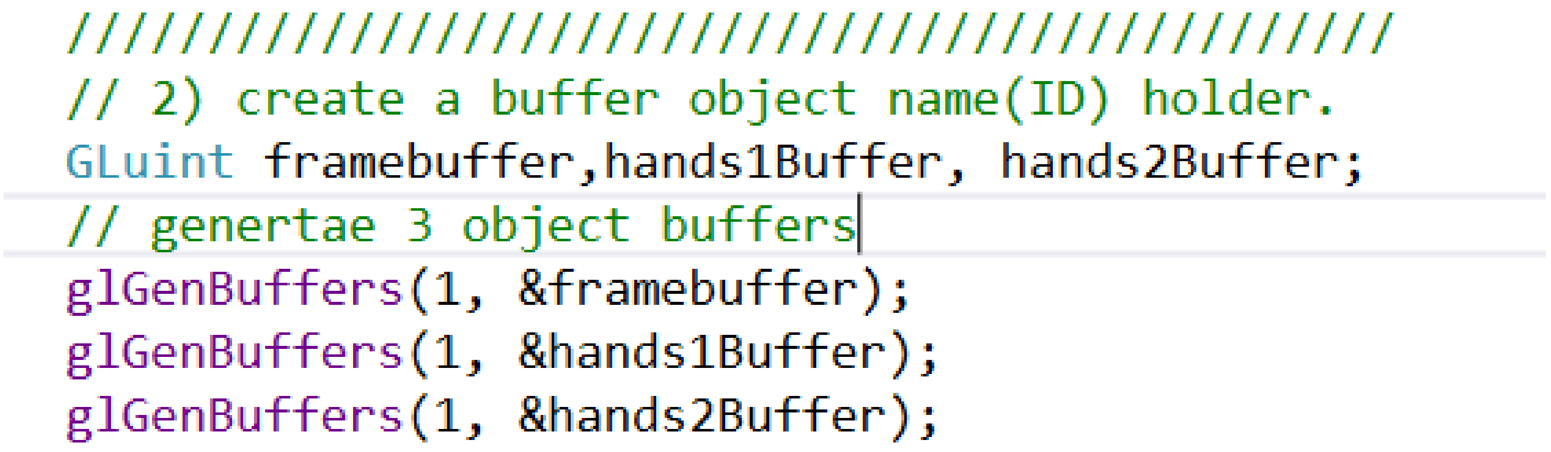
```
// Create and compile 3 shaders for 3 different obejects the frame, hand 1, hand2
GLuint programID = LoadShaders("circlevertex.vs", "circleframe2.fs");
GLuint hand1ID = LoadShaders("hand1vertex.vs", "hand1frame.fs");
GLuint hand2ID = LoadShaders("hand2vertex.vs", "hand2frame.fs");
```

Vertex Arrays and buffer Objects

Creating 3 vertex arrays and 3 buffer objects

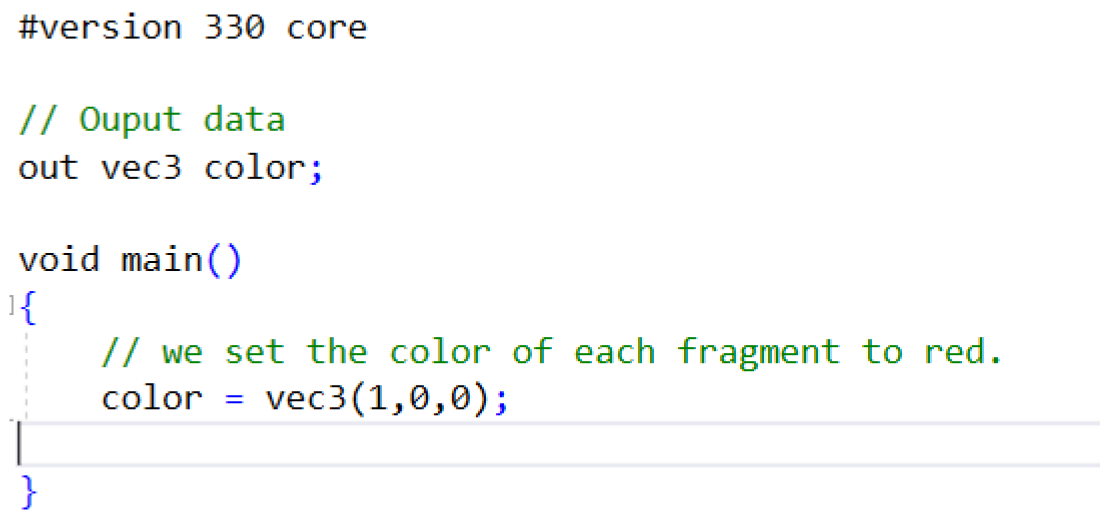


```
GLuint FrameArrayID,hand1ArrayID, hand2ArrayID;
glGenVertexArrays(1, &FrameArrayID);
glGenVertexArrays(1, &hand1ArrayID);
glGenVertexArrays(1, &hand2ArrayID);
```



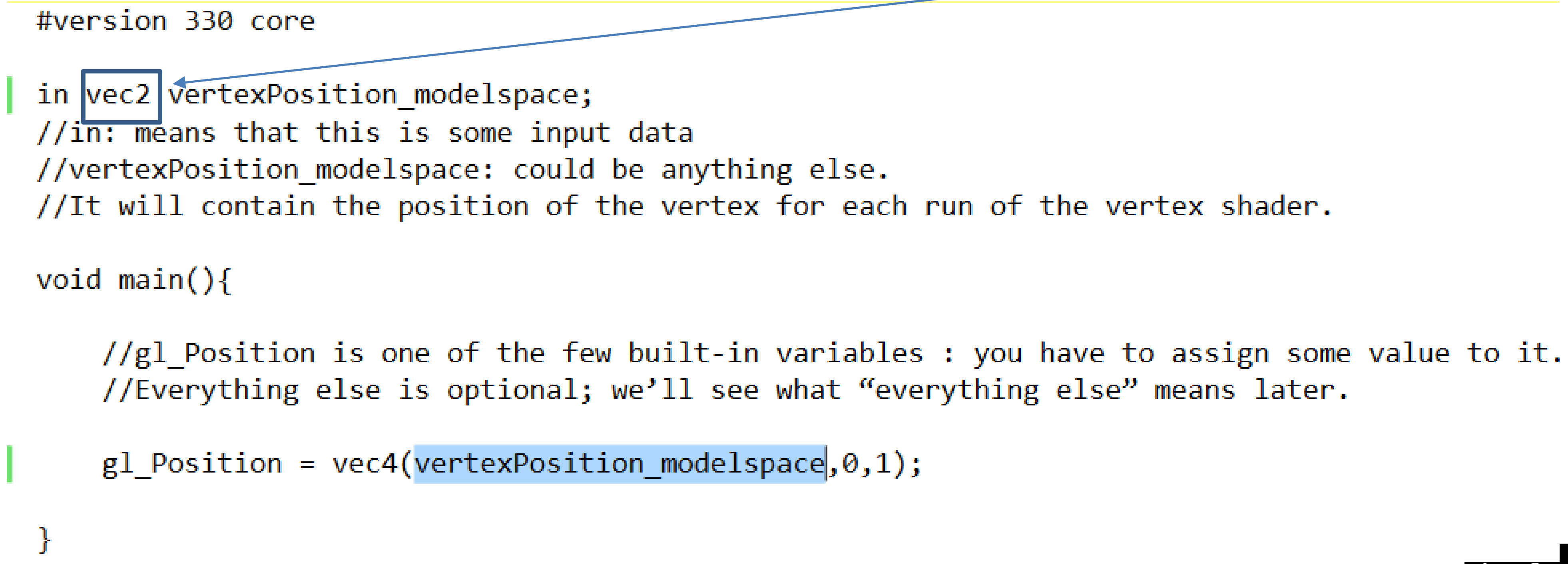
```
////////////////////////////////////////////////
// 2) create a buffer object name(ID) holder.
GLuint framebuffer,hands1Buffer, hands2Buffer;
// genertae 3 object buffers
glGenBuffers(1, &framebuffer);
glGenBuffers(1, &hands1Buffer);
glGenBuffers(1, &hands2Buffer);
```

The Frame Vertex and Fragment Shaders



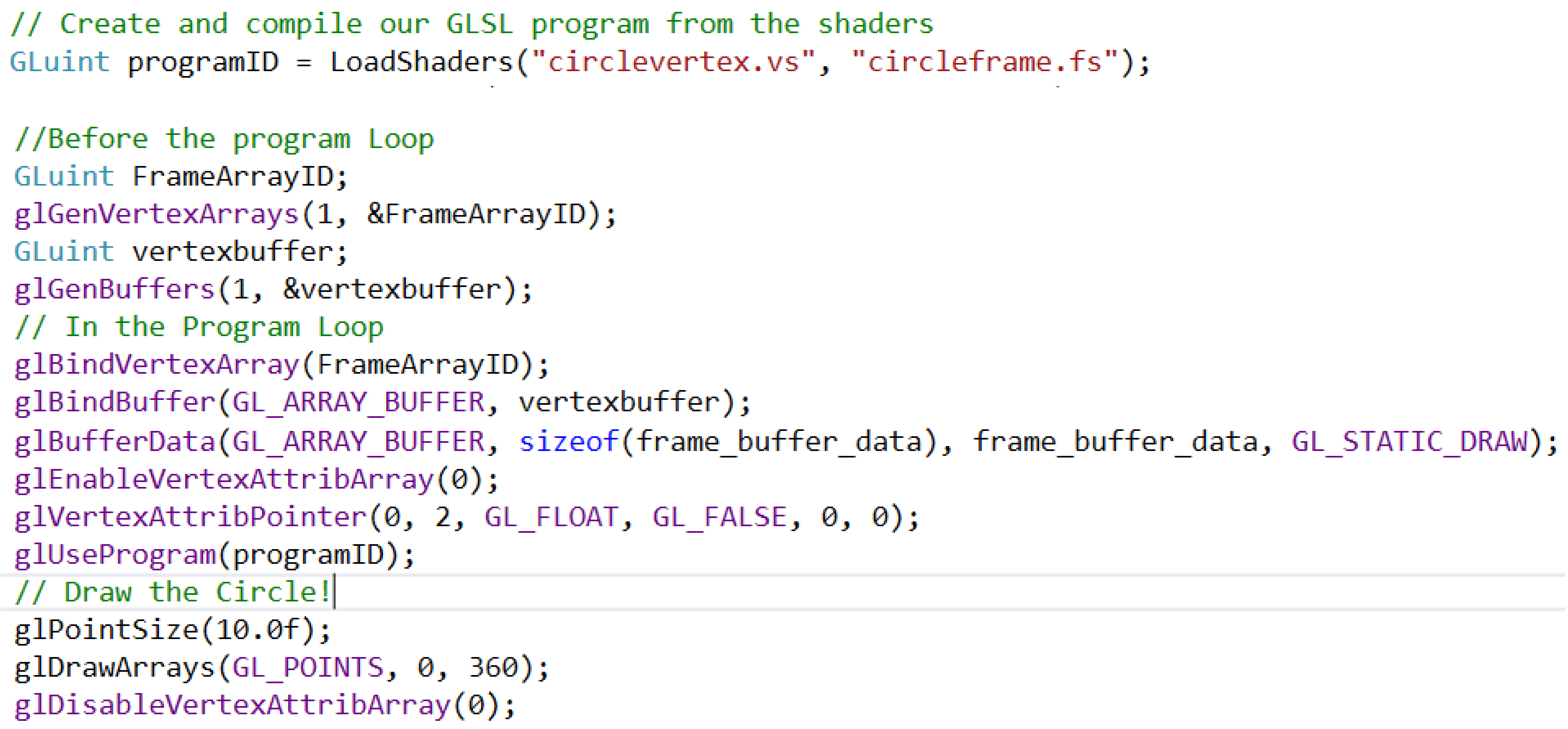
```
#version 330 core

// Ouput data
out vec3 color;

void main()
{
    // we set the color of each fragment to red.
    color = vec3(1,0,0);

}
```

Why vec2 not vec3?



```
#version 330 core

in vec2 vertexPosition_modelspace;
//in: means that this is some input data
//vertexPosition_modelspace: could be anything else.
//It will contain the position of the vertex for each run of the vertex shader.

void main(){

    //gl_Position is one of the few built-in variables : you have to assign some value to it.
    //Everything else is optional; we’ll see what “everything else” means later.

    gl_Position = vec4(vertexPosition_modelspace,0,1);

}
```

Drawing The frame

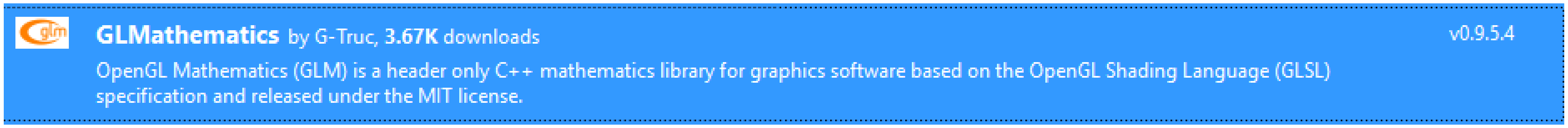


```
// Create and compile our GLSL program from the shaders
GLuint programID = LoadShaders("circlevertex.vs", "circleframe.fs");

//Before the program Loop
GLuint FrameArrayID;
glGenVertexArrays(1, &FrameArrayID);
GLuint vertexbuffer;
glGenBuffers(1, &vertexbuffer);
// In the Program Loop
glBindVertexArray(FrameArrayID);
glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(frame_buffer_data), frame_buffer_data, GL_STATIC_DRAW);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 0, 0);
glUseProgram(programID);
// Draw the Circle!
glPointSize(10.0f);
glDrawArrays(GL_POINTS, 0, 360);
glDisableVertexAttribArray(0);
```

Adding Transformations

1- Add GLM package from package manager

- A Math library for matrix operations



GLMathematics by G-Truc, **3.67K** downloads v0.9.5.4
OpenGL Mathematics (GLM) is a header only C++ mathematics library for graphics software based on the OpenGL Shading Language (GLSL) specification and released under the MIT license.

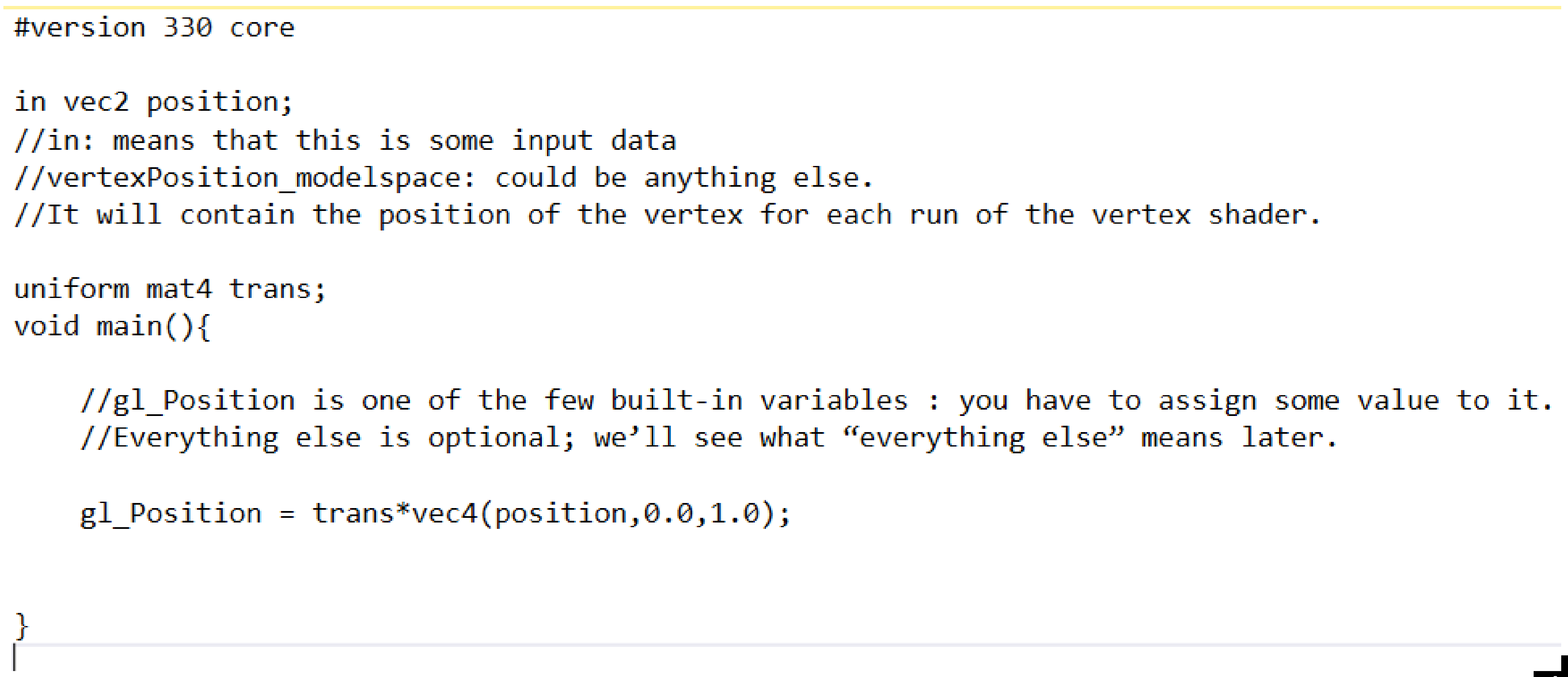
2-Add the appropriate header glm library files

```
#include <glm\glm.hpp>
#include <glm\gtc\matrix_transform.hpp>
#include <glm\gtc\type_ptr.hpp>
```

Modifying the Vertex Shader

- Add a matrix of type uniform
- Uniform data type is used to allow two-way communication

  between shaders and your program



```
#version 330 core

in vec2 position;
//in: means that this is some input data
//vertexPosition_modelspace: could be anything else.
//It will contain the position of the vertex for each run of the vertex shader.

uniform mat4 trans;
void main(){

    //gl_Position is one of the few built-in variables : you have to assign some value to it.
    //Everything else is optional; we’ll see what “everything else” means later.

    gl_Position = trans*vec4(position,0.0,1.0);


}
```

Hands Transformation

This is how we update a variable declared in the shaders from the program

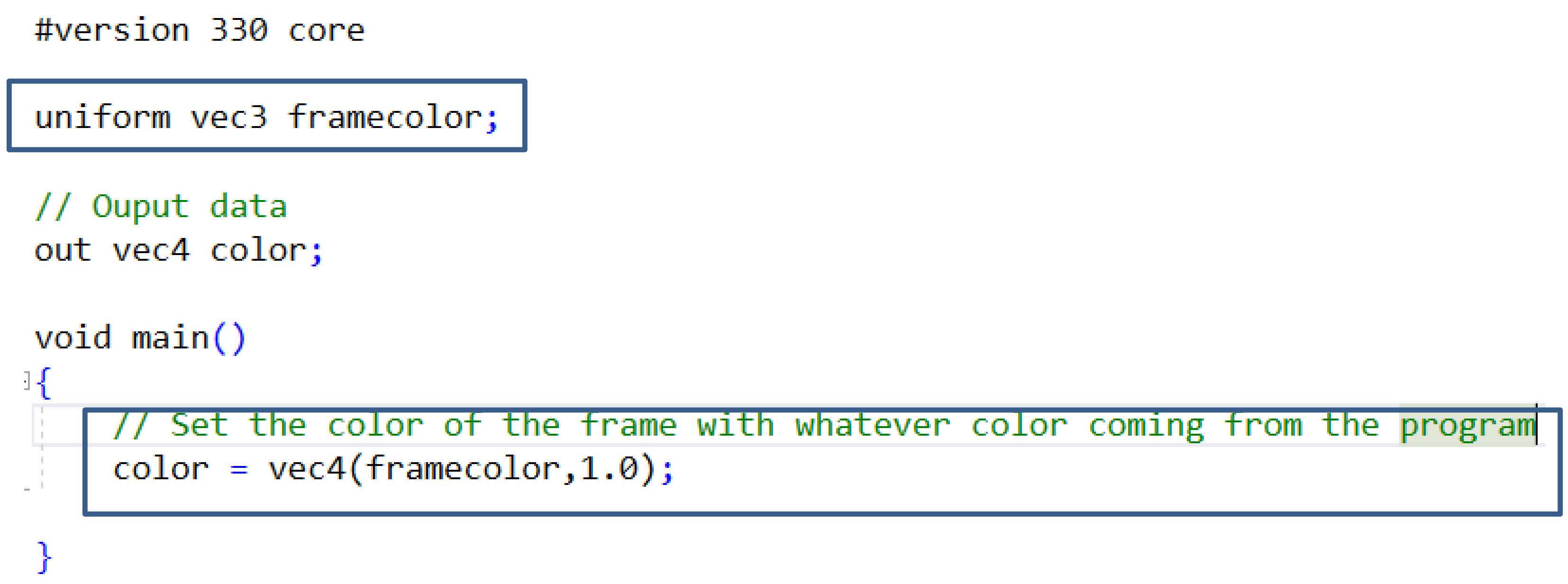
```
clock_t begin = clock();
```

```
//Transformation of the minutes hand
//https://open.gl/transformations
clock_t end = clock();
float elapsed_secs = double(end - begin) / CLOCKS_PER_SEC;
//printf("%f", elapsed_secs);
//*glm::radians(25.0f)
glm::mat4 trans;
trans = glm::rotate(trans, -elapsed_secs, glm::vec3(0.0f, 0.0f, 1.0f));
GLint uniTrans = glGetUniformLocation(hand2ID, "trans");
glUniformMatrix4fv(uniTrans, 1, GL_FALSE, glm::value_ptr(trans));
```

Rotation about Z axis

Changing the Frame color from the our Program

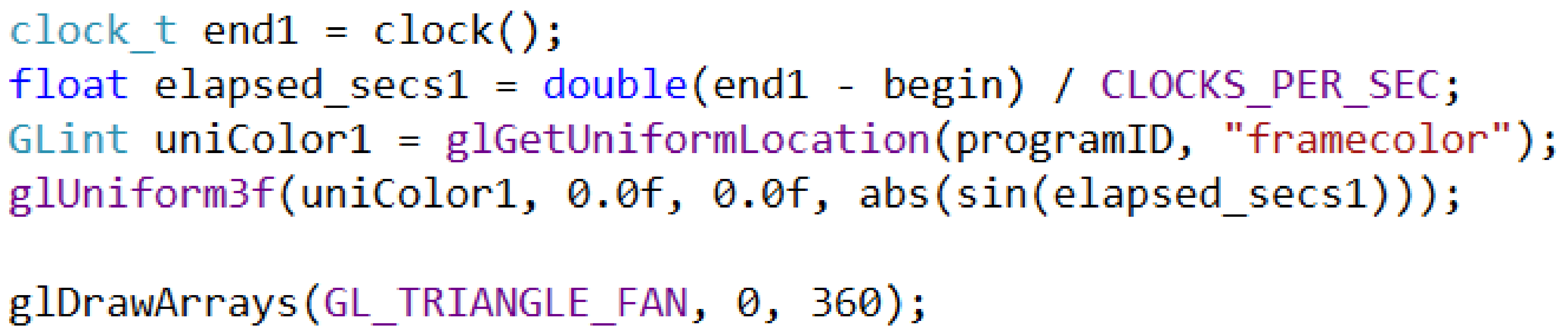
1- Modifying the fragment shader by adding a uniform variable



```
#version 330 core

uniform vec3 framecolor;

// Ouput data
out vec4 color;

void main()
{
    // Set the color of the frame with whatever color coming from the program
    color = vec4(framecolor,1.0);

}
```

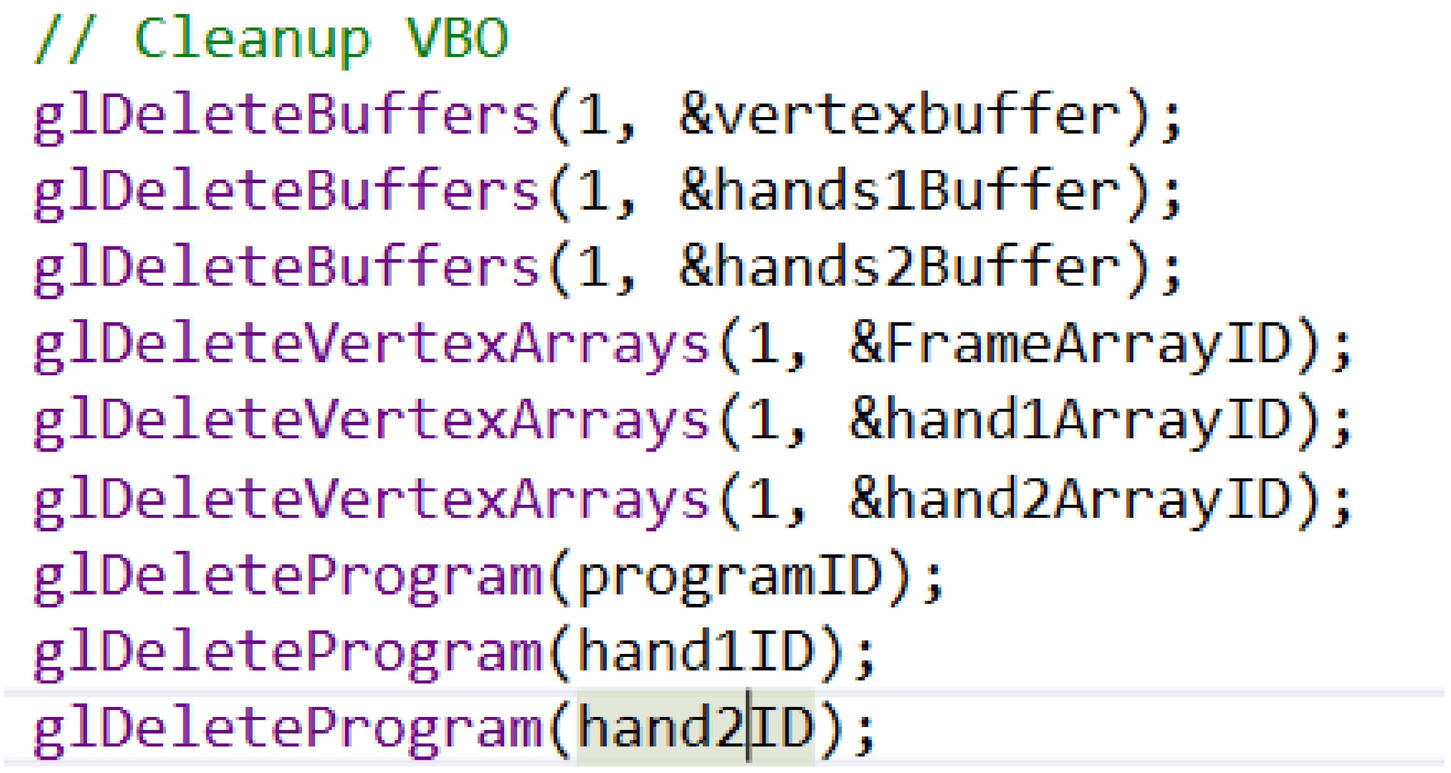
Changing the Frame color from the our Program

- 2- Set the uniform variable from inside your code



```
clock_t end1 = clock();
float elapsed_secs1 = double(end1 - begin) / CLOCKS_PER_SEC;
GLint uniColor1 = glGetUniformLocation(programID, "framecolor");
glUniform3f(uniColor1, 0.0f, 0.0f, abs(sin(elapsed_secs1)));

glDrawArrays(GL_TRIANGLE_FAN, 0, 360);
```

CleanUp



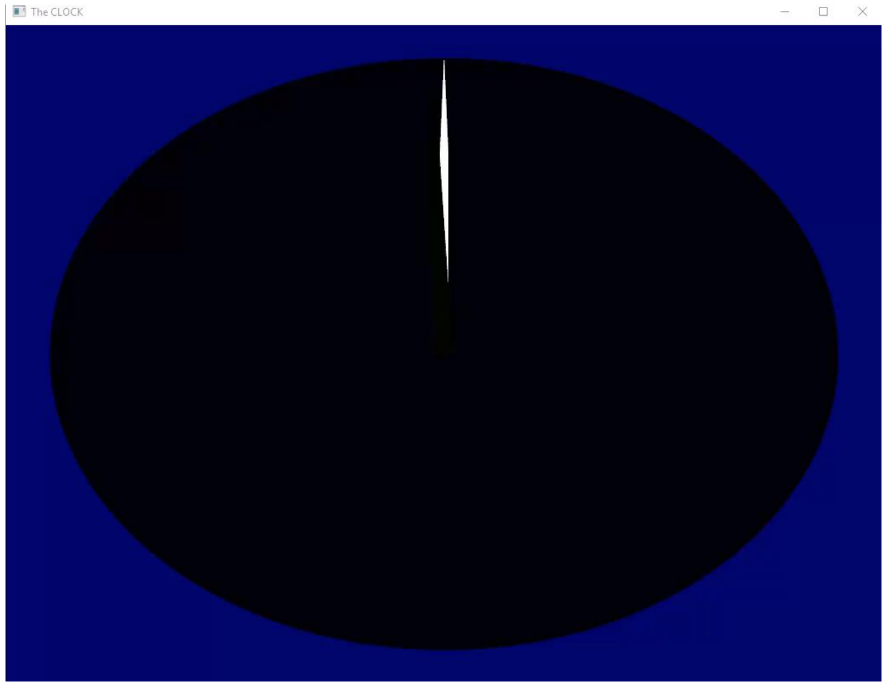
```
// Cleanup VBO
glDeleteBuffers(1, &vertexbuffer);
glDeleteBuffers(1, &hands1Buffer);
glDeleteBuffers(1, &hands2Buffer);
glDeleteVertexArrays(1, &FrameArrayID);
glDeleteVertexArrays(1, &hand1ArrayID);
glDeleteVertexArrays(1, &hand2ArrayID);
glDeleteProgram(programID);
glDeleteProgram(hand1ID);
glDeleteProgram(hand2ID);
```

Your turn

1- Modify the program to change the color of the seconds hands

You will need to do the following:

- Update the hands frame shader (hands2frame.fs)
- Update the program accordingly



Thank You

Questions

Khaled Rabieh