```cpp
#include<cmath>
#include<string.h>
#include<stdlib.h>

#include<GL/glew.h>
#include<GLFW/glfw3.h>
#include<cstdio>

//#include<glm/mat4x4.hpp> //rotATION TRANSLATION AND SCALE
#include<glm/glm.hpp>
#include<glm/gtc/matrix_transform.hpp>
#include<glm/gtc/type_ptr.hpp>

using namespace std;

//window dimensions
const GLint WIDTH=800, HEIGHT = 600;
const float toRadians = 3.14159265f / 180.0f;

GLuint VAO, VBO,IBO, shader, uniformModel; //IBO index Buffer Object

bool direction = true;
float triOffset = 0.0f; // line 224 while ...
float triMaxoffset = 0.7f;
float triIncrement = 0.0005f;

float curAngle = 0.0f;

bool sizeDirection = true;
float curSize = 0.4f;
float maxSize = 0.8f;
float minSize = 0.1f;



// Vertex Shader
static const char* vShader = "                                                  \n\
#version 330                                                                    \n\
                                                                                \n\
layout (location = 0) in vec3 pos;                                              \n\
                                                                                \n\
out vec4 vCol;                                                                  \n\
uniform mat4 model;                                                             \n\
void main()                                                                     \n\
{                                                                               \n\
    gl_Position = model * vec4(pos, 1.0);                                       \n\
    vCol = vec4(clamp(pos,0.0f,1.0f), 1.0f);                                    ↵
                                \n\
}";

// Fragment Shader
static const char* fShader = "                          \n\
```

```
52  #version 330                                                    \n\
53  in vec4 vCol;                                                   \n\
54  out vec4 colour;                                                \n\
55                                                                  \n\
56  void main()                                                     \n\
57  {                                                               \n\
58      colour = vCol;                                              \n\
59  }";
60
61  void CreateTriangle()
62  {
63      unsigned int indices[]=
64      {
65          0,3,1,
66          1,3,2,
67          2,3,0,
68          0,1,2
69
70      };
71
72      GLfloat vertices[] = {
73          -1.0f, -1.0f, 0.0f,
74          0.0f, -1.0f, 1.0f,
75          1.0f, -1.0f, 0.0f,
76          0.0f, 1.0, 0.0f
77
78      };
79      glGenVertexArrays(1, &VAO);
80      glBindVertexArray(VAO);
81
82
83              glGenBuffers(1, &IBO);
84                  glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IBO);
85              glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices,
                  GL_STATIC_DRAW);
86
87              glGenBuffers(1, &VBO);
88                  glBindBuffer(GL_ARRAY_BUFFER, VBO);
89              glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices,
                  GL_STATIC_DRAW);
90
91              glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
92              glEnableVertexAttribArray(0);
93
94          glBindBuffer(GL_ARRAY_BUFFER, 0);
95          glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0); //IMPORTANT:NOTE YOU SHOULD
              UNBIND IBO AFTER YOU UNBIND VAO
96      glBindVertexArray(0);
97  }
98
99  void AddShader(GLuint theProgram, const char* shaderCode, GLenum shaderType)
100 {
```

```
101     GLuint theShader = glCreateShader(shaderType);
102     const GLchar* theCode[1];
103     theCode[0] = shaderCode;
104
105     GLint codeLength[1];
106     codeLength[0] = strlen(shaderCode);
107
108     glShaderSource(theShader, 1, theCode, codeLength);
109     glCompileShader(theShader);//////////////////////////
110
111     GLint result = 0;
112     GLchar eLog[1024] = { 0 };
113
114
115     glGetShaderiv(theShader, GL_COMPILE_STATUS, &result);
116
117     if (!result)
118     {
119         glGetShaderInfoLog(theShader, sizeof(eLog), NULL, eLog);
120         printf("Error compiling %d shader: '%s'\n", shaderType, eLog);
121         return;
122
123     }
124     glAttachShader(theProgram, theShader);
125
126
127 }
128
129 //compileShaders
130
131 void compileShaders() {
132     shader = glCreateProgram();
133     if (!shader) {
134         printf("error fragment creating ...");
135
136         return;
137
138     }
139
140
141     AddShader(shader, vShader, GL_VERTEX_SHADER);
142     AddShader(shader, fShader, GL_FRAGMENT_SHADER);
143
144
145     //chunk of code 27:30
146
147     GLint result = 0;
148     GLchar eLog[1024] = { 0 };
149
150     glLinkProgram(shader);
151         glGetProgramiv(shader, GL_LINK_STATUS, &result);
152
```

```
153         if(!result)
154         {
155             glGetProgramInfoLog(shader, sizeof(eLog), NULL, eLog);
156             printf("Error Linking Program  %d shader: '%s ", eLog);
157             return;
158 
159         }
160     glValidateProgram(shader);
161         glGetProgramiv(shader, GL_VALIDATE_STATUS, &result);
162 
163         if (!result)
164         {
165             glGetProgramInfoLog(shader, sizeof(eLog), NULL, eLog);
166             printf("Error Validating Program  %d shader: '%s ", eLog);
167             return;
168 
169         }
170         //uniformModel = glGetUniformLocation(shader, "xMove");
171           uniformModel = glGetUniformLocation(shader, "model");
172 
173 }
174 
175 
176 int main()
177 {
178     //initialization
179 
180     if (!glfwInit())
181     {
182         printf("GLFW not working initializing ");
183         glfwTerminate();
184         return 1; //means fails
185 
186 
187     }
188 
189 
190     //setup GLFW window properties
191     //OpemGL Version
192 
193     glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
194     glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
195     glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
196     glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
197 
198     GLFWwindow *mainwindow = glfwCreateWindow(WIDTH, HEIGHT, "Test Window", NULL, ↵
        NULL);
199     if (!mainwindow)
200 
201     {
202         printf("GLFW window creation failed");
203         glfwTerminate();
```

```
204         return 1;
205
206     }
207
208     int bufferWidth, bufferHeight;
209     glfwGetFramebufferSize(mainwindow, &bufferWidth, &bufferHeight);
210
211
212
213
214
215
216
217     //set context for GLFW
218
219     glfwMakeContextCurrent(mainwindow);
220
221     //Allow modern extension features
222     glewExperimental = GL_TRUE;
223     if (glewInit() != GLEW_OK)
224     {
225         printf("GLEW initi... failed ");
226         glfwDestroyWindow(mainwindow);
227         glfwTerminate();
228
229         return 1;
230     }
231
232     //enable depth
233     glEnable(GL_DEPTH_TEST);
234
235     //setup Viewport Size
236     glViewport(0, 0, bufferWidth, bufferHeight);
237
238     //create Triangle
239     CreateTriangle();
240     compileShaders();
241
242     //loop untill window closes
243     while (!glfwWindowShouldClose(mainwindow))
244     {
245         //Get + Handle user input ... any event keyboard mouse stuff user moving
246         glfwPollEvents();
247
248         if (direction)
249         {
250             triOffset += triIncrement;
251         }
252
253
254         else
255         {
```

```
256             triOffset -= triIncrement;
257         }
258
259         if (abs(triOffset) >= triMaxoffset) //abs means absolute
260
261         {
262             direction = !direction;  //its a switch
263         }
264
265
266         curAngle += 0.01f;
267         if (curAngle >= 360)
268         {
269             curAngle -= 360;
270         }
271
272
273         if (sizeDirection) {
274             curSize += 0.0001f;
275
276         }
277
278         else
279         {
280             curSize -= 0.0001f;
281         }
282
283         if (curSize >= maxSize||curSize<=minSize)
284         {
285             sizeDirection = !sizeDirection;
286         }
287         //clear windpw
288         glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
289         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
290
291
292         glUseProgram(shader);
293         //glm
294         glm::mat4 model;
295
296         model = glm::rotate(model, curAngle * toRadians, glm::vec3(0.0f, 1.0f,
              0.0f)); // order of transforming is important which one comes 1st
297         model = glm::translate(model, glm::vec3(triOffset, 0.0f, 0.0f));
298         model = glm::scale(model, glm::vec3(curSize,curSize, 1.0f));
299
300         // glUniform1f(uniformXMove,triOffset); //set uniformXmove to the value
              of triOffset
301
302         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
303
304         glBindVertexArray(VAO);
305
```

```
306         glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IBO);
307         glDrawElements(GL_TRIANGLES, 12, GL_UNSIGNED_INT, 0);
308         glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
309
310         glBindVertexArray(0);
311
312         glUseProgram(0);
313
314         glfwSwapBuffers(mainwindow);
315
316     }
317
318     return 0;
319 }
320
```