```cpp
1  #include<cmath>
2  #include<string.h>
3  #include<stdlib.h>
4  #include<vector> //array list
5
6  #include<GL/glew.h>
7  #include<GLFW/glfw3.h>
8  #include<cstdio>
9
10 //#include<glm/mat4x4.hpp> //rotATION TRANSLATION AND SCALE
11 #include<glm/glm.hpp>
12 #include<glm/gtc/matrix_transform.hpp>
13 #include<glm/gtc/type_ptr.hpp>
14
15 #include"Window.h"
16 #include"Mesh.h"
17 #include"Shader.h"
18
19
20 using namespace std;
21
22 //window dimensions
23 const GLint WIDTH=800, HEIGHT = 600;
24 const float toRadians = 3.14159265f / 180.0f;
25
26 Window mainWindow;
27 std::vector<Mesh*> MeshList;
28 std::vector<Shader>shaderList;
29
30
31
32 //GLuint shader, uniformModel, uniformProjection; //IBO index Buffer Object
33
34 bool direction = true;
35 float triOffset = 0.0f; // line 224 while ...
36 float triMaxoffset = 0.7f;
37 float triIncrement = 0.0005f;
38
39 float curAngle = 0.0f;
40
41 bool sizeDirection = true;
42 float curSize = 0.4f;
43 float maxSize = 0.8f;
44 float minSize = 0.1f;
45
46
47
48 // Vertex Shader
49 static const char* vShader = "Shaders/shader.vert.txt";
50
51 // Fragment Shader
52 static const char* fShader = "Shaders/shader.frag.txt";
```

```cpp

void CreateObjects()
{
    unsigned int indices[]=
    {
        0,3,1,
        1,3,2,
        2,3,0,
        0,1,2

    };

    GLfloat vertices[] = {
        -1.0f, -1.0f, 0.0f,
        0.0f, -1.0f, 1.0f,
        1.0f, -1.0f, 0.0f,
        0.0f, 1.0, 0.0f

    };

    Mesh *obj1 = new Mesh(); //initialize everything to zero VBO = 0 .... etc
    obj1->createMesh(vertices, indices, 12, 12); // 12 vertices for now
    MeshList.push_back(obj1);

    Mesh *obj2 = new Mesh(); //initialize everything to zero VBO = 0 .... etc
    obj2->createMesh(vertices, indices, 12, 12); // 12 vertices for now
    MeshList.push_back(obj2);
}

void createShaders() {


    Shader *shader1 = new Shader();
    shader1->createFromFiles(vShader, fShader);
    shaderList.push_back(*shader1);
}

int main()
{

    mainWindow = Window(800, 600);
    mainWindow.initialise();


    //create Triangle
    CreateObjects();
    createShaders();

    GLuint uniformProjection = 0 , uniformModel = 0;

    glm::mat4 projection = glm::perspective(45.0f, mainWindow.getBufferWidth() / 
      mainWindow.getBufferHeight(), 0.1f, 100.0f);
    //loop untill window closes
    while (!mainWindow.getShouldclose())
    {
        //Get + Handle user input ... any event keyboard mouse stuff user moving
        glfwPollEvents();

        if (direction)
        {
            triOffset += triIncrement;
        }


        else
        {
            triOffset -= triIncrement;
        }

        if (abs(triOffset) >= triMaxoffset) //abs means absolute

        {
            direction = !direction;  //its a switch
        }


        curAngle += 0.01f;
        if (curAngle >= 360)
        {
            curAngle -= 360;
        }


        if (sizeDirection) {
            curSize += 0.0001f;

        }

        else
        {
            curSize -= 0.0001f;
        }

        if (curSize >= maxSize||curSize<=minSize)
        {
            sizeDirection = !sizeDirection;
        }

        //clear windpw
        glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);


        //glUseProgram(shader);
        shaderList[0].useShader();
        uniformModel = shaderList[0].GetModelLocation();
        uniformProjection = shaderList[0].GetProjectionLocation();
        //glm
        glm::mat4 model;

        model = glm::translate(model, glm::vec3(0.0f, 0.0f, -2.0f));
        model = glm::rotate(model, curAngle * toRadians, glm::vec3(0.0f, 1.0f, 0.0f)); // order of transforming is important which one comes 1st

        // glUniform1f(uniformXMove,triOffset); //set uniformXmove to the value of triOffset

        glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
        glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));

        MeshList[0]->RenderMesh();

        model = glm::mat4();
        model = glm::translate(model, glm::vec3(-triOffset, 1.0f, -2.5f));
        model = glm::scale(model, glm::vec3(curSize, curSize, 1.0f));
        glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
        MeshList[1]->RenderMesh();

        glUseProgram(0);

        mainWindow.swapbuffers();

    }

    return 0;
}

```