#include <stdio.h>

#include <string.h>

#include <cmath>

#include <GL\glew.h>

#include <GLFW\glfw3.h>

#include <glm/mat4x4.hpp>

#include <glm/glm.hpp>

#include <glm\gtc\matrix\_transform.hpp>

#include <glm\gtc\type\_ptr.hpp>

// Window Dimensions

const GLint WIDTH = 900, HEIGHT = 600;

float curAngle = 0.0f;

bool directionSize = true;

float curSize = 0.04f, maxSize = 0.8f, minSize = 0.1f;

GLuint VAO, VBO, shader, uniformModel;

bool direction = true;

float triOffset = 0.0f, triMaxOffset = 0.7f, triIncrement = 0.0005f;

// Vertex Shader

static const char\* vShader = " \n\

#version 330 \n\

\n\

layout(location = 0) in vec3 pos; \n\

\n\

out vec4 vCol; \n\

\n\

uniform mat4 model; \n\

\n\

void main() \n\

{ \n\

gl\_Position = model \* vec4(pos, 1.0); \n\

vCol = vec4(clamp(pos, 0.0f, 1.0f), 1.0f); \n\

}";

// Vertex Shader

// Fragment Shader

static const char\* fShader = " \n\

#version 330 \n\

\n\

in vec4 vCol; \n\

\n\

out vec4 color; \n\

\n\

void main() \n\

{ \n\

color = vCol; \n\

}";

// Fragment Shader

// Create Triangle Function 1st Method Complete

void CreateTriangle()

{

GLfloat vertices[] = { -1.2f, -1.2f, 0.0f,

1.2f, -1.2f, 0.0f,

0.0f, 1.5f, 0.0f };

glGenVertexArrays(1, &VAO);

glBindVertexArray(VAO);

glGenBuffers(1, &VBO);

glBindBuffer(GL\_ARRAY\_BUFFER, VBO);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(vertices), vertices, GL\_STATIC\_DRAW);

glVertexAttribPointer(0, 3, GL\_FLOAT, GL\_FALSE, 0, 0);

glEnableVertexAttribArray(0);

glBindBuffer(GL\_ARRAY\_BUFFER, 0);

glBindVertexArray(0);

}

// Create Triangle Function 1st Method Complete

void AddShader(GLuint theprogram, const char\* shadercode, GLenum shadertype)

{

GLuint theshader = glCreateShader(shadertype);

const GLchar\* theCode[1];

theCode[0] = shadercode;

GLint codelength[1];

codelength[0] = strlen(shadercode);

glShaderSource(theshader, 1, theCode, codelength);

glCompileShader(theshader);

GLint result = 0;

GLchar elog[1024] = { 0 };

glGetShaderiv(theshader, GL\_COMPILE\_STATUS, &result);

if (!result)

{

glGetShaderInfoLog(theshader, sizeof(elog), NULL, elog);

printf("Error Compiling the %d Shader: '%s' \n.", shadertype, elog);

return;

}

glAttachShader(theprogram, theshader);

}

void CompileShader()

{

shader = glCreateProgram();

if (!shader)

{

printf("Error Creating Shader Program.");

return;

}

AddShader(shader, vShader, GL\_VERTEX\_SHADER);

AddShader(shader, fShader, GL\_FRAGMENT\_SHADER);

GLint result = 0;

GLchar elog[1024] = { 0 };

glLinkProgram(shader);

glGetProgramiv(shader, GL\_LINK\_STATUS, &result);

if (!result)

{

glGetProgramInfoLog(shader, sizeof(elog), NULL, elog);

printf("Error Linking Program: '%s' \n.", elog);

return;

}

}

int main()

{

// Initialize GLFW

if (!glfwInit())

{

printf("GLFW Initialization Failed!");

glfwTerminate();

return 1;

}

// Setup GLFW Window Properties

// Open GL Version

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MAJOR, 3);

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MINOR, 3);

// Core Profile = No Backwards Compatibility

glfwWindowHint(GLFW\_OPENGL\_PROFILE, GLFW\_OPENGL\_CORE\_PROFILE);

// Allow Forward Compatibility

glfwWindowHint(GLFW\_OPENGL\_FORWARD\_COMPAT, GL\_TRUE);

GLFWwindow\* mainWindow = glfwCreateWindow(WIDTH, HEIGHT, "Test Window", NULL, NULL);

if (!mainWindow)

{

printf("GLFW window creation failed!");

glfwTerminate();

return 1;

}

// Get Buffer Size Information

int bufferWidth, bufferHeight;

glfwGetFramebufferSize(mainWindow, &bufferWidth, &bufferHeight);

// Get Context For GLEW to use

glfwMakeContextCurrent(mainWindow);

// Allow Modern Extension Features

glewExperimental = GL\_TRUE;

if (glewInit())

{

printf("GLEW Initialization Failed!");

glfwDestroyWindow(mainWindow);

glfwTerminate();

return 1;

}

// Setup Viewport Size

glViewport(0, 0, bufferWidth, bufferHeight);

CreateTriangle();

CompileShader();

// Loop Until Window Closed

while (!glfwWindowShouldClose(mainWindow))

{

// Get + Handle User Input Events

glfwPollEvents();

if (direction)

{

triOffset += triIncrement;

}

else

{

triOffset -= triIncrement;

}

if (abs(triOffset) >= triMaxOffset)

{

direction = !direction;

}

// Clear Window

glClearColor(0.4f, 0.2f, 0.4f, 0.2f);

glClear(GL\_COLOR\_BUFFER\_BIT);

glUseProgram(shader);

curAngle += 0.03f;

if (curAngle >= 360)

{

curAngle -= 360;

}

if (directionSize)

{

curSize += 0.001f;

}

else

{

curSize -= 0.001f;

}

if (curSize >= maxSize || curSize <= minSize)

{

directionSize = !directionSize;

}

glm::mat4 model;

model = glm::scale(model, glm::vec3(0.4f, 0.4f, 1.0f));

glUniformMatrix4fv(uniformModel, 1, GL\_FALSE, glm::value\_ptr(model));

glBindVertexArray(VAO);

glDrawArrays(GL\_TRIANGLES, 0, 3);

glBindVertexArray(0);

glUseProgram(0);

glfwSwapBuffers(mainWindow);

}

return 0;

}

