#include <stdio.h>

#include <iostream>

#include <string.h>

#include <assert.h>

#include <math.h>

#include <GL/glew.h>

#include <GL/freeglut.h>

#include <glm/glm.hpp>

#include "pipeline3.h"

#define WINDOW\_WIDTH 1024

#define WINDOW\_HEIGHT 768

using namespace glm;

GLuint VBO;

GLuint IBO;

GLuint gWorldLocation;

static const char\* pVS = " \n\

#version 330 \n\

\n\

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

\n\

uniform mat4 gWorld; \n\

\n\

out vec4 Color; \n\

\n\

void main() \n\

{ \n\

gl\_Position = gWorld \* vec4(Position, 1.0); \n\

Color = vec4(clamp(Position, 0.0, 1.0), 1.0); \n\

}";

static const char\* pFS = " \n\

#version 330 \n\

\n\

in vec4 Color; \n\

\n\

out vec4 FragColor; \n\

\n\

void main() \n\

{ \n\

FragColor = Color; \n\

}";

void RenderSceneCB()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

static float Scale = 0.0f;

Scale += 0.1f;

Pipeline p;

p.WorldPos(0.0f, 0.0f, 5.0f);

p.SetPerspectiveProj(30.0f, WINDOW\_WIDTH, WINDOW\_HEIGHT, 1.0f, 1000.0f);

vec3 CameraPos(1.0f, 1.0f, -3.0f);

vec3 CameraTarget(0.45f, 0.0f, 1.0f);

vec3 CameraUp(0.0f, 1.0f, 0.0f)

p.SetCamera(CameraPos, CameraTarget, CameraUp);

glUniformMatrix4fv(gWorldLocation, 1, GL\_TRUE, (const GLfloat\*)p.GetTrans());

glEnableVertexAttribArray(0);

glBindBuffer(GL\_ARRAY\_BUFFER, VBO);

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

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, IBO);

glDrawElements(GL\_TRIANGLES, 12, GL\_UNSIGNED\_INT, 0);

glDisableVertexAttribArray(0);

glutSwapBuffers();

}

static void InitializeGlutCallbacks()

{

glutDisplayFunc(RenderSceneCB);

glutIdleFunc(RenderSceneCB);

}

static void CreateVertexBuffer()

{

vec3 Vertices[4];

Vertices[0] = vec3(-1.0f, -1.0f, 0.0f);

Vertices[1] = vec3(0.0f, -1.0f, 1.0f);

Vertices[2] = vec3(1.0f, -1.0f, 0.0f);

Vertices[3] = vec3(0.0f, 1.0f, 0.0f);

glGenBuffers(1, &VBO);

glBindBuffer(GL\_ARRAY\_BUFFER, VBO);

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

}

static void CreateIndexBuffer()

{

unsigned int Indices[] = { 0, 3, 1,

1, 3, 2,

2, 3, 0,

0, 2, 1 };

glGenBuffers(1, &IBO);

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, IBO);

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

}

static void AddShader(GLuint ShaderProgram, const char\* pShaderText, GLenum ShaderType)

{

GLuint ShaderObj = glCreateShader(ShaderType);

if (ShaderObj == 0) {

fprintf(stderr, "Error creating shader type %d\n", ShaderType);

exit(0);

}

const GLchar\* p[1];

p[0] = pShaderText;

GLint Lengths[1];

Lengths[0] = strlen(pShaderText);

glShaderSource(ShaderObj, 1, p, Lengths);

glCompileShader(ShaderObj);

GLint success;

glGetShaderiv(ShaderObj, GL\_COMPILE\_STATUS, &success);

if (!success) {

GLchar InfoLog[1024];

glGetShaderInfoLog(ShaderObj, 1024, NULL, InfoLog);

fprintf(stderr, "Error compiling shader type %d: '%s'\n", ShaderType, InfoLog);

exit(1);

}

glAttachShader(ShaderProgram, ShaderObj);

}

static void CompileShaders()

{

GLuint ShaderProgram = glCreateProgram();

if (ShaderProgram == 0) {

fprintf(stderr, "Error creating shader program\n");

exit(1);

}

AddShader(ShaderProgram, pVS, GL\_VERTEX\_SHADER);

AddShader(ShaderProgram, pFS, GL\_FRAGMENT\_SHADER);

glLinkProgram(ShaderProgram);

GLint Success = 0;

GLchar ErrorLog[1024] = { 0 };

glGetProgramiv(ShaderProgram, GL\_LINK\_STATUS, &Success);

if (Success == 0) {

glGetProgramInfoLog(ShaderProgram, sizeof(ErrorLog), NULL, ErrorLog);

fprintf(stderr, "Error linking shader program: '%s'\n", ErrorLog);

exit(1);

}

glValidateProgram(ShaderProgram);

glGetProgramiv(ShaderProgram, GL\_VALIDATE\_STATUS, &Success);

if (!Success) {

glGetProgramInfoLog(ShaderProgram, sizeof(ErrorLog), NULL, ErrorLog);

fprintf(stderr, "Invalid shader program: '%s'\n", ErrorLog);

exit(1);

}

glUseProgram(ShaderProgram);

gWorldLocation = glGetUniformLocation(ShaderProgram, "gWorld");

assert(gWorldLocation != 0xFFFFFFFF);

}

int main(int argc, char \*\*argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA);

glutInitWindowSize(WINDOW\_WIDTH, WINDOW\_HEIGHT);

glutInitWindowPosition(100, 100);

glutCreateWindow("Tutorial 13");

InitializeGlutCallbacks();

GLenum res = glewInit();

if (res != GLEW\_OK)

{

fprintf(stderr, "Error: '%s'\n", glewGetErrorString(res));

return 1;

}

glClearColor(0.0f, 0.0f, 0.0f, 0.0f);

CreateVertexBuffer();

CreateIndexBuffer();

CompileShaders();

glutMainLoop();

return 0;

}