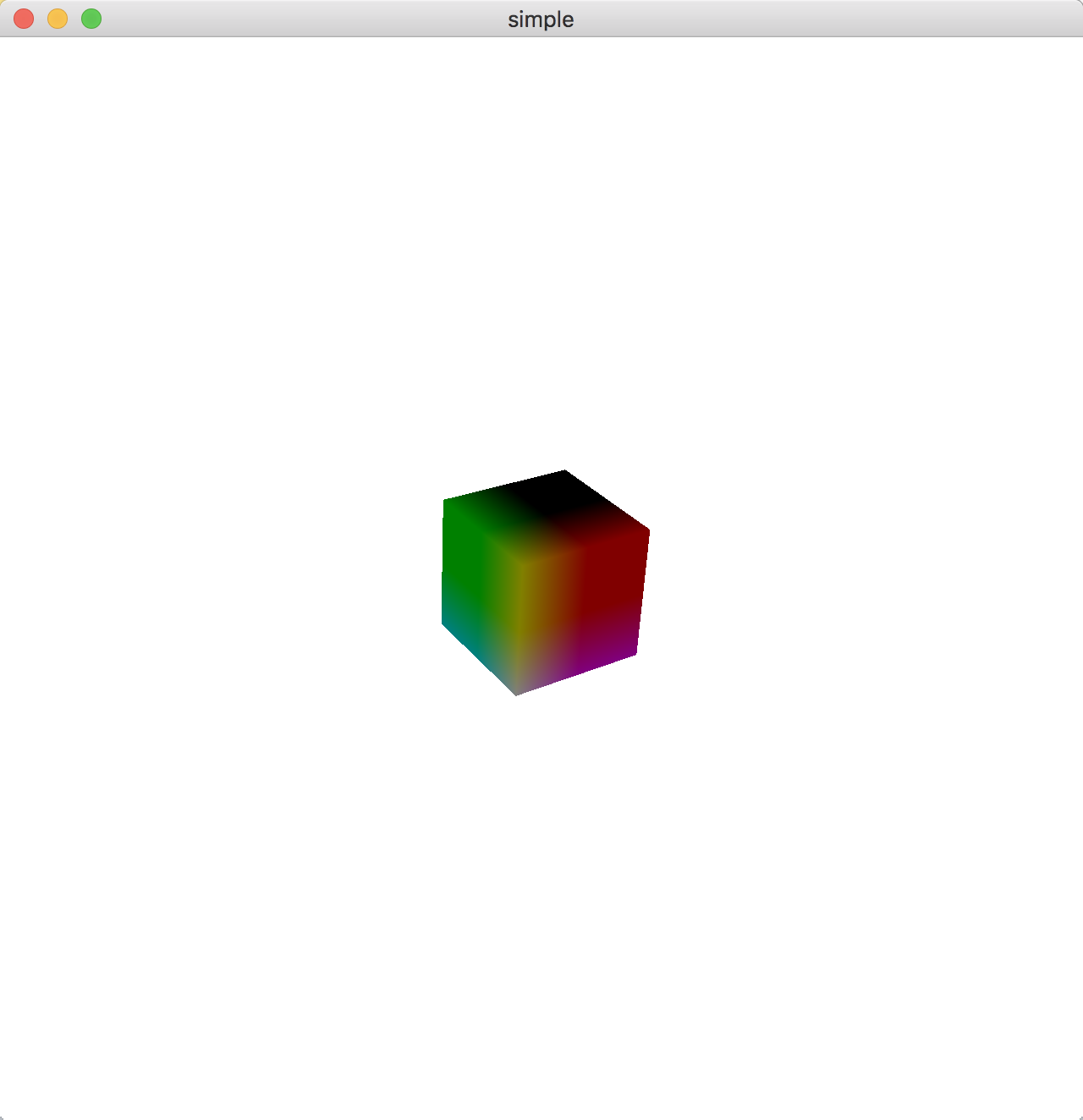
Name: Han Song

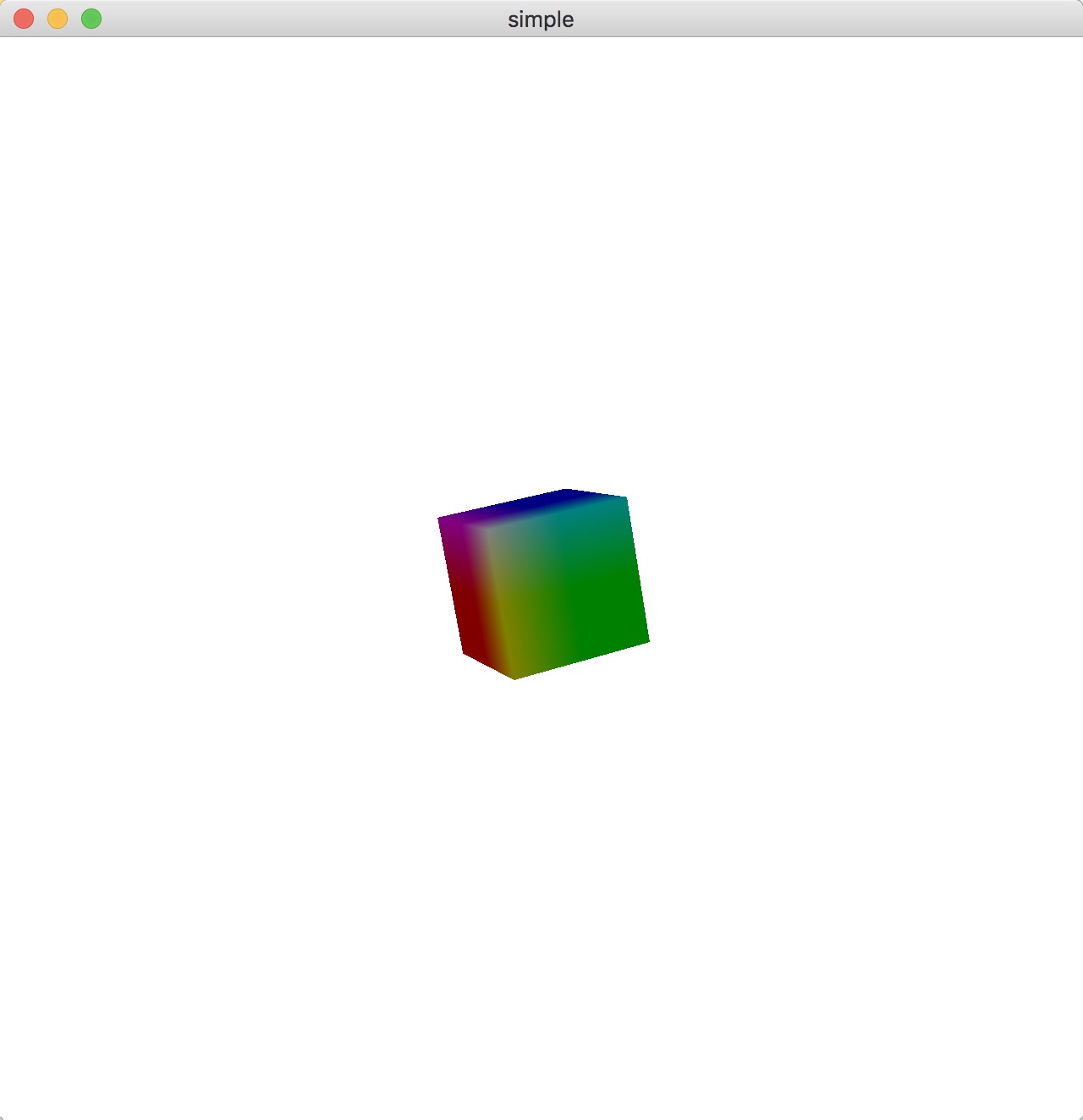
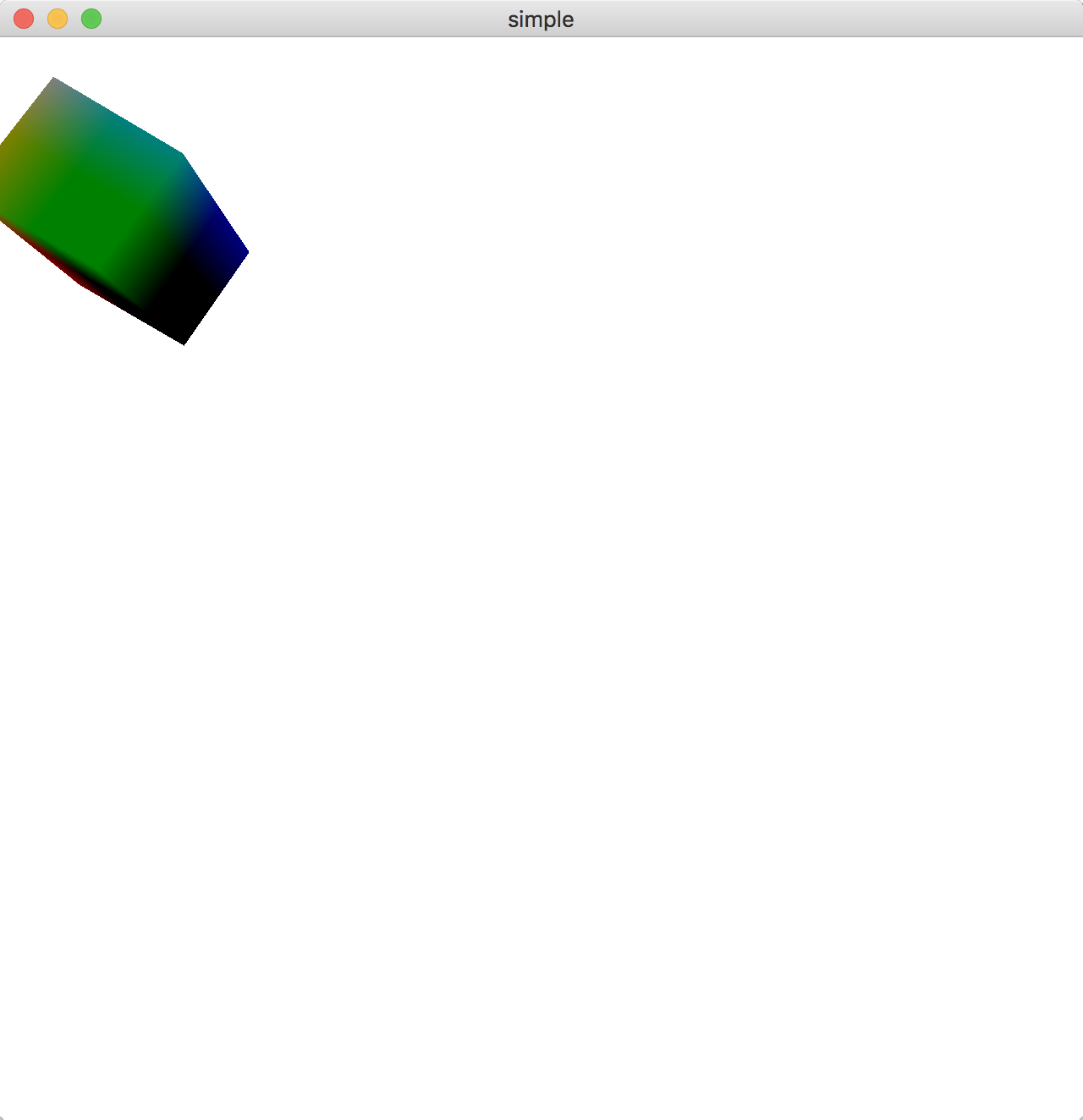
Assignment 3A

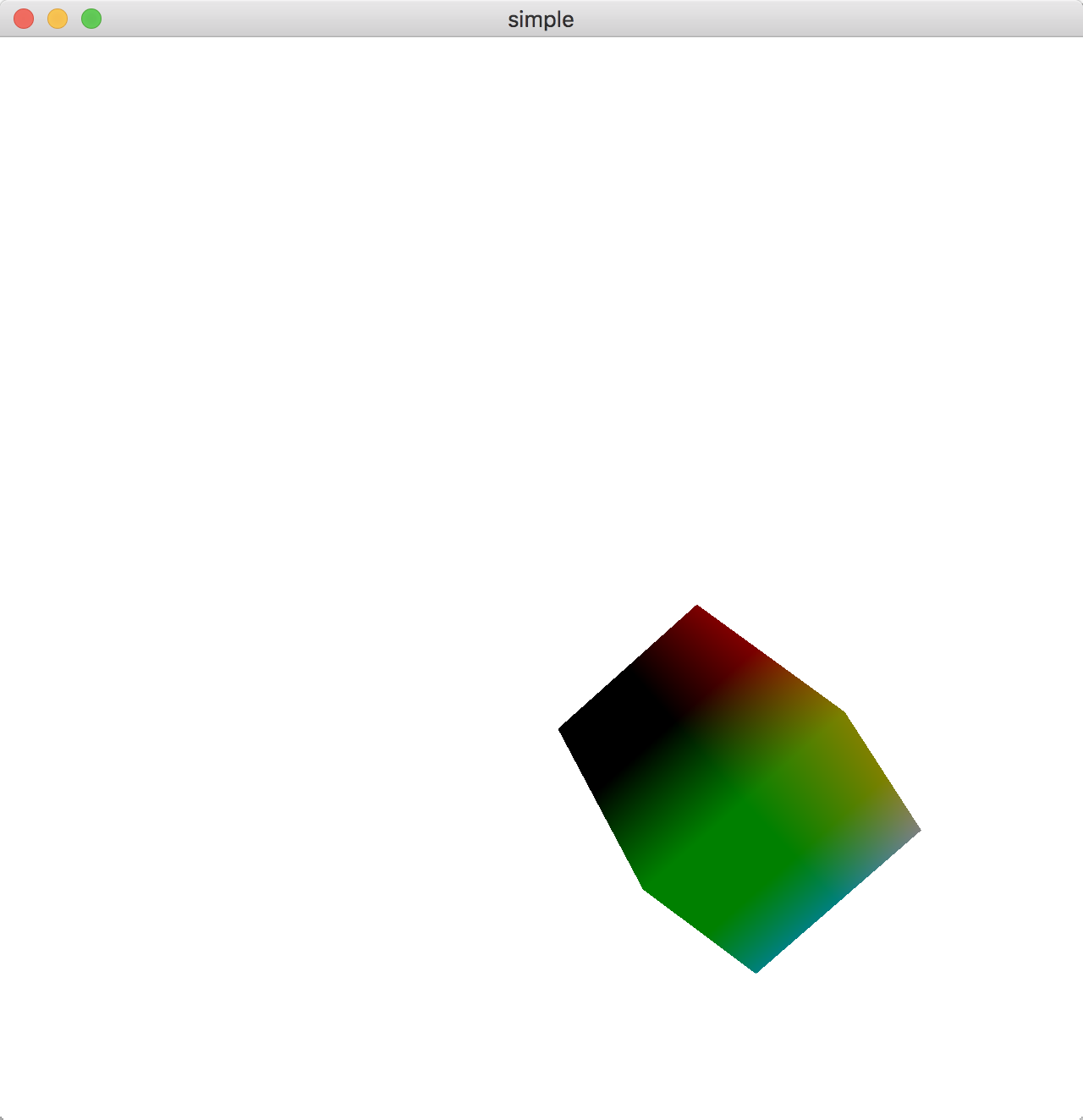
Date: Mar/21/2018

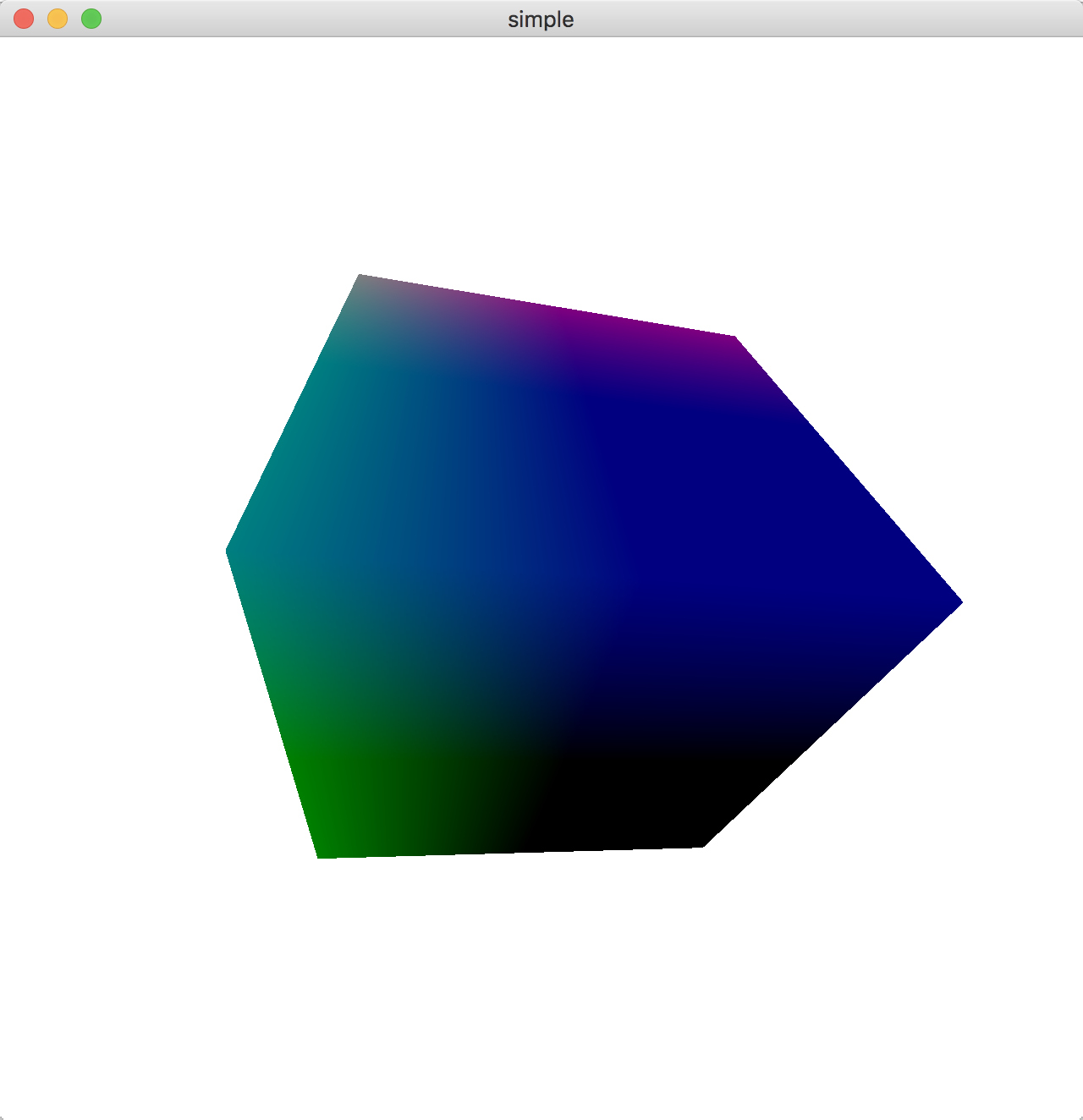
Pawprint: hs267

14253499









Zoom in feature

// RotateCube.cpp

// Generate a rotating cube on a solid background

#include <stdio.h>

#include <stdlib.h>

#include <GL/glew.h>

#include <GLFW/glfw3.h>

#include "InitShader.h"

#include <glm/glm.hpp>

#include <glm/vec4.hpp>

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

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

#include <glm/gtx/normal.hpp>

#include <glm/vec3.hpp> // glm::vec3

#include <glm/vec4.hpp> // glm::vec4

#include <glm/mat4x4.hpp> // glm::mat4

#include <glm/gtc/matrix\_transform.hpp> // glm::translate, glm::rotate, glm::scale, glm::perspective

GLuint vao;

int Index = 0;

GLuint program;

const int NumVertices = 36;

double hrotate, vrotate;

bool isPressed = NULL;

double oldX;

double oldY;

glm::vec4 points[NumVertices];

glm::vec4 colors[NumVertices];

//----------------------------------------------------------------------------

glm::vec4 vertices[8] = {

glm::vec4(-0.5, -0.5, 0.5, 1.0),

glm::vec4(-0.5, 0.5, 0.5, 1.0),

glm::vec4(0.5, 0.5, 0.5, 1.0),

glm::vec4(0.5, -0.5, 0.5, 1.0),

glm::vec4(-0.5, -0.5, -0.5, 1.0),

glm::vec4(-0.5, 0.5, -0.5, 1.0),

glm::vec4(0.5, 0.5, -0.5, 1.0),

glm::vec4(0.5, -0.5, -0.5, 1.0)

};

glm::vec4 vertex\_colors[8] = {

glm::vec4(0.0, 0.0, 0.0, 1.0), // black

glm::vec4(1.0, 0.0, 0.0, 1.0), // red

glm::vec4(1.0, 1.0, 0.0, 1.0), // yellow

glm::vec4(0.0, 1.0, 0.0, 1.0), // green

glm::vec4(0.0, 0.0, 1.0, 1.0), // blue

glm::vec4(1.0, 0.0, 1.0, 1.0), // magenta

glm::vec4(1.0, 1.0, 1.0, 1.0), // white

glm::vec4(0.0, 1.0, 1.0, 1.0) // cyan

};

enum { Xaxis = 0, Yaxis = 1, Zaxis = 2, NumAxes = 3 };

int Axis = Xaxis;

GLfloat Theta[NumAxes] = { 0.0, 0.0, 0.0 };

GLuint mvpi;

void quad(int a, int b, int c, int d)

{

colors[Index] = vertex\_colors[a]; points[Index] = vertices[a]; Index++;

colors[Index] = vertex\_colors[b]; points[Index] = vertices[b]; Index++;

colors[Index] = vertex\_colors[c]; points[Index] = vertices[c]; Index++;

colors[Index] = vertex\_colors[a]; points[Index] = vertices[a]; Index++;

colors[Index] = vertex\_colors[c]; points[Index] = vertices[c]; Index++;

colors[Index] = vertex\_colors[d]; points[Index] = vertices[d]; Index++;

}

void

colorcube()

{

quad(1, 0, 3, 2);

quad(2, 3, 7, 6);

quad(3, 0, 4, 7);

quad(6, 5, 1, 2);

quad(4, 5, 6, 7);

quad(5, 4, 0, 1);

}

void init(void)

{

colorcube();

// Create a vertex array object

GLuint vao;

glGenVertexArrays(1, &vao);

glBindVertexArray(vao);

// Create and initialize a buffer object

GLuint buffer;

glGenBuffers(1, &buffer);

glBindBuffer(GL\_ARRAY\_BUFFER, buffer);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(points) + sizeof(colors),

NULL, GL\_STATIC\_DRAW);

glBufferSubData(GL\_ARRAY\_BUFFER, 0, sizeof(points), points);

glBufferSubData(GL\_ARRAY\_BUFFER, sizeof(points), sizeof(colors), colors);

// Load shaders and use the resulting shader program

program = InitShader("vshader36.glsl", "fshader36.glsl");

glUseProgram(program);

// set up vertex arrays

GLuint vPosition = glGetAttribLocation(program, "vPosition");

glEnableVertexAttribArray(vPosition);

glVertexAttribPointer(vPosition, 4, GL\_FLOAT, GL\_FALSE, 0,

NULL);

GLuint vColor = glGetAttribLocation(program, "vColor");

glEnableVertexAttribArray(vColor);

glVertexAttribPointer(vColor, 4, GL\_FLOAT, GL\_FALSE, 0,

NULL);

mvpi = glGetUniformLocation(program, "mvp");

glEnable(GL\_DEPTH\_TEST);

glClearColor(1.0, 1.0, 1.0, 1.0);

}

void mymouse(GLFWwindow\* window, int button, int action, int mods)

{

if (button == GLFW\_MOUSE\_BUTTON\_LEFT) {

if (GLFW\_PRESS == action) {

isPressed = true;

glfwGetCursorPos(window, &oldX, &oldY);

}

else if (GLFW\_RELEASE == action) {

isPressed = false;

hrotate = 0;

vrotate = 0;

}

}

}

void mykey(GLFWwindow\* window, int key, int scancode, int action, int mods)

{

if (action == GLFW\_PRESS) {

if (key == GLFW\_KEY\_ESCAPE || key == GLFW\_KEY\_Q) {

glfwSetWindowShouldClose(window, GL\_TRUE);

}

if (key == GLFW\_KEY\_Z) {

glm::mat4 model = glm::mat4(1.0);

glm::mat4 Projection = glm::perspective(glm::radians(45.0f), 1.0f, 0.1f, 100.f);

model = glm::scale(glm::mat4(1.0f), glm::vec3(0.5f));

}

}

}

static void cursor\_pos\_callback(GLFWwindow\* window, double xpos, double ypos)

{

if (isPressed) {

hrotate += 0.01\*(xpos - oldX);

vrotate += 0.01\*(ypos - oldY);

}

}

//----------------------------------------------------------------------------

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

{

GLFWwindow \*window = NULL;

const GLubyte \*renderer;

const GLubyte \*version;

/\* start GL context and O/S window using the GLFW helper library \*/

if (!glfwInit()) {

fprintf(stderr, "ERROR: could not start GLFW3\n");

return 1;

}

/\* We must specify 3.2 core if on Apple OS X -- other O/S can specify

anything here. I defined 'APPLE' in the makefile for OS X

Remove the #ifdef #endif and play around with this - you should be starting

an explicit version anyway, and some non-Apple drivers will require this too.

\*/

#ifdef APPLE

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

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MINOR, 2);

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

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

#endif

window = glfwCreateWindow(640, 640, "simple", NULL, NULL);

if (!window) {

fprintf(stderr, "ERROR: could not open window with GLFW3\n");

glfwTerminate();

return 1;

}

glfwMakeContextCurrent(window);

/\* start GLEW extension handler \*/

glewExperimental = GL\_TRUE;

glewInit();

renderer = glGetString(GL\_RENDERER); /\* get renderer string \*/

version = glGetString(GL\_VERSION); /\* version as a string \*/

printf("Renderer: %s\n", renderer);

printf("OpenGL version supported %s\n", version);

init();

glfwSetMouseButtonCallback(window, mymouse);

glfwSetCursorPosCallback(window, cursor\_pos\_callback);

do {

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); // clear the window

glm::mat4 model = glm::mat4(1.0);

glm::vec3 eye(0.0f, 0.0f, 2.0f);

glm::vec3 at(0.0f, 0.0f, 0.0f);

glm::vec3 up(0.0, 1.0f, 0.0f);

glm::mat4 view = glm::lookAt(eye, at, up);

glm::mat4 Projection = glm::perspective(glm::radians(45.0f), 1.0f, 0.1f, 100.f);

model = glm::scale(glm::mat4(1.0f), glm::vec3(0.5f));

model = glm::translate(model, glm::vec3(0.0f, 0.0f, -5.0f));

model = glm::rotate(model, 10.0f, glm::vec3(1.0f, 0.0f, 0.0f));

model = glm::rotate(model, GLfloat(hrotate), glm::vec3(0.0, 1.0, 0.0));

model = glm::rotate(model, GLfloat(vrotate), glm::vec3(1.0, 0.0, 0.0));

glm::mat4 mvp = Projection \* view \* model;

GLuint mvp\_pos = glGetUniformLocation(program, "mvp");

glUniformMatrix4fv(mvp\_pos, 1, GL\_FALSE, glm::value\_ptr(mvp));

glUniformMatrix4fv(mvpi, 1, GL\_FALSE, glm::value\_ptr(mvp));

glDrawArrays(GL\_TRIANGLES, 0, NumVertices); // draw the points

/\* update other events like input handling \*/

glfwPollEvents();

glfwSwapBuffers(window);

} while (!glfwWindowShouldClose(window));

// Close OpenGL window and terminate GLFW

glfwTerminate();

return 0;

}