Tutorial 4

Views

CS4052 Computer Graphics

Alexandru Sulea
D Stream
#12315152
28 October 2016

1 Overview

I used the mathfuncs and the sample code provided for the lab to achieve the initial views. The orthographic projection function was written into mathfuncs.cpp and the lookat function was also used from the given mathfuncs.cpp. Rotations was simply achieved by incrementally incresing the angle of rotation.

2 Rotation

For this part of the assignmeant I used the rotation functions provided by mathfunc.cpp

```
//Some Windows Headers (For Time, IO, etc.)
#include <windows.h>
#include <mmsystem.h>
#include <GL/glew.h>
#include <GL/freeglut.h>
#include <iostream>
#include "maths_funcs.h"
#include "teapot.h" // teapot mesh
#include <string>
#include <fstream>
#include <iostream>
#include <sstream>

// Macro for indexing vertex buffer
#define BUFFER_OFFSET(i) ((char *)NULL + (i))

using namespace std;
GLuint shaderProgramID;

unsigned int teapot_vao = 0;
int width = 800;
int height = 600;
```

```
GLuint loc1;
GLuint loc2;

GLfloat rotate1x = 0.0f;
GLfloat rotate2x = 0.0f;
GLfloat rotate3x = 0.0f;
GLfloat rotate4x = 0.0f;
GLfloat rotate5x = 0.0f;
GLfloat rotate6x = 0.0f;
GLfloat rotate7x = 0.0f;
GLfloat rotate8x = 0.0f;

GLfloat rotate1y = 0.0f;
GLfloat rotate2y = 0.0f;
GLfloat rotate3y = 0.0f;
GLfloat rotate4y = 0.0f;
GLfloat rotate5y = 0.0f;
GLfloat rotate6y = 0.0f;
GLfloat rotate7y = 0.0f;
GLfloat rotate8y = 0.0f;

GLfloat rotate1z = 0.0f;
GLfloat rotate2z = 0.0f;
GLfloat rotate3z = 0.0f;
GLfloat rotate4z = 0.0f;
GLfloat rotate5z = 0.0f;
GLfloat rotate6z = 0.0f;
GLfloat rotate7z = 0.0f;
GLfloat rotate8z = 0.0f;


GLfloat translate1x = 0.0f;
GLfloat translate2x = 0.0f;
GLfloat translate3x = 0.0f;
GLfloat translate4x = 0.0f;
GLfloat translate5x = 0.0f;
GLfloat translate6x = 0.0f;
GLfloat translate7x = 0.0f;
GLfloat translate8x = 0.0f;

GLfloat translate1y = 0.0f;
GLfloat translate2y = 0.0f;
GLfloat translate3y = 0.0f;
GLfloat translate4y = 0.0f;
GLfloat translate5y = 0.0f;
GLfloat translate6y = 0.0f;
GLfloat translate7y = 0.0f;
GLfloat translate8y = 0.0f;

GLfloat translate1z = 0.0f;
GLfloat translate2z = 0.0f;
GLfloat translate3z = 0.0f;
GLfloat translate4z = 0.0f;
GLfloat translate5z = 0.0f;
GLfloat translate6z = 0.0f;
```

```
GLfloat translate7z = 0.0f;
GLfloat translate8z = 0.0f;

GLfloat testing = 0.0f;

float red = 1.0f, blue = 1.0f, green = 1.0f;

// Shader Functions- click on + to expand
#pragma region SHADER_FUNCTIONS

// Create a NULL-terminated string by reading the provided file
std::string readShaderSource(const std::string& fileName)
{
   std::ifstream file(fileName.c_str());
   if (file.fail()) {
      cout << "error loading shader called " << fileName;
      exit(1);
   }

   std::stringstream stream;
   stream << file.rdbuf();
   file.close();

   return stream.str();
}


static void AddShader(GLuint ShaderProgram, const char* pShaderText, GLenum ShaderType)
{
   // create a shader object
   GLuint ShaderObj = glCreateShader(ShaderType);

   if (ShaderObj == 0) {
      fprintf(stderr, "Error creating shader type %d\n", ShaderType);
      exit(0);
   }
   std::string outShader = readShaderSource(pShaderText);
   const char* pShaderSource = outShader.c_str();

   // Bind the source code to the shader, this happens before compilation
   glShaderSource(ShaderObj, 1, (const GLchar**)&pShaderSource, NULL);
   // compile the shader and check for errors
   glCompileShader(ShaderObj);
   GLint success;
   // check for shader related errors using glGetShaderiv
   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);
   }
   // Attach the compiled shader object to the program object
   glAttachShader(ShaderProgram, ShaderObj);
}
```

```
GLuint CompileShaders()
{
   //Start the process of setting up our shaders by creating a program ID
   //Note: we will link all the shaders together into this ID
   shaderProgramID = glCreateProgram();
   if (shaderProgramID == 0) {
      fprintf(stderr, "Error creating shader program\n");
      exit(1);
   }

   // Create two shader objects, one for the vertex, and one for the fragment shader
   AddShader(shaderProgramID, "../Project4/Shaders/simpleVertexShader.txt", GL_VERTEX_SHADER);
   AddShader(shaderProgramID, "../Project4/Shaders/simpleFragmentShader.txt", GL_FRAGMENT_SHADER);

   GLint Success = 0;
   GLchar ErrorLog[1024] = { 0 };
   // After compiling all shader objects and attaching them to the program, we can finally link it
   glLinkProgram(shaderProgramID);
   // check for program related errors using glGetProgramiv
   glGetProgramiv(shaderProgramID, GL_LINK_STATUS, &Success);
   if (Success == 0) {
      glGetProgramInfoLog(shaderProgramID, sizeof(ErrorLog), NULL, ErrorLog);
      fprintf(stderr, "Error linking shader program: '%s'\n", ErrorLog);
      exit(1);
   }

   // program has been successfully linked but needs to be validated to check whether the program
       can execute given the current pipeline state
   glValidateProgram(shaderProgramID);
   // check for program related errors using glGetProgramiv
   glGetProgramiv(shaderProgramID, GL_VALIDATE_STATUS, &Success);
   if (!Success) {
      glGetProgramInfoLog(shaderProgramID, sizeof(ErrorLog), NULL, ErrorLog);
      fprintf(stderr, "Invalid shader program: '%s'\n", ErrorLog);
      exit(1);
   }
   // Finally, use the linked shader program
   // Note: this program will stay in effect for all draw calls until you replace it with another
       or explicitly disable its use
   glUseProgram(shaderProgramID);
   return shaderProgramID;
}
#pragma endregion SHADER_FUNCTIONS

// VBO Functions - click on + to expand
#pragma region VBO_FUNCTIONS

void generateObjectBufferTeapot() {
   GLuint vp_vbo = 0;

   loc1 = glGetAttribLocation(shaderProgramID, "vertex_position");
   loc2 = glGetAttribLocation(shaderProgramID, "vertex_normals");

   glGenBuffers(1, &vp_vbo);
   glBindBuffer(GL_ARRAY_BUFFER, vp_vbo);
```

```
  glBufferData(GL_ARRAY_BUFFER, 3 * teapot_vertex_count * sizeof(float), teapot_vertex_points,
      GL_STATIC_DRAW);
  GLuint vn_vbo = 0;
  glGenBuffers(1, &vn_vbo);
  glBindBuffer(GL_ARRAY_BUFFER, vn_vbo);
  glBufferData(GL_ARRAY_BUFFER, 3 * teapot_vertex_count * sizeof(float), teapot_normals,
      GL_STATIC_DRAW);

  glGenVertexArrays(1, &teapot_vao);
  glBindVertexArray(teapot_vao);

  glEnableVertexAttribArray(loc1);
  glBindBuffer(GL_ARRAY_BUFFER, vp_vbo);
  glVertexAttribPointer(loc1, 3, GL_FLOAT, GL_FALSE, 0, NULL);
  glEnableVertexAttribArray(loc2);
  glBindBuffer(GL_ARRAY_BUFFER, vn_vbo);
  glVertexAttribPointer(loc2, 3, GL_FLOAT, GL_FALSE, 0, NULL);
}


#pragma endregion VBO_FUNCTIONS


void display() {
  // tell GL to only draw onto a pixel if the shape is closer to the viewer
  glEnable(GL_DEPTH_TEST); // enable depth-testing
  glDepthFunc(GL_LESS); // depth-testing interprets a smaller value as "closer"
  glClearColor(red, green, blue, 1.0f);
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glUseProgram(shaderProgramID);
  /*START*/
  /*1*/
  //Declare your uniform variables that will be used in your shader
  int matrix_location = glGetUniformLocation(shaderProgramID, "model");
  int view_mat_location = glGetUniformLocation(shaderProgramID, "view");
  int proj_mat_location = glGetUniformLocation(shaderProgramID, "proj");

  // Hierarchy of Teapots

  // Root of the Hierarchy
  mat4 view = identity_mat4();
  mat4 persp_proj = perspective(45.0, (float)width / (float)height, 0.1, 100.0);
  mat4 local1;
  //parent hiararchy
  /*right arm*/
  local1 = identity_mat4();
  local1 = rotate_z_deg(local1, rotate1z);
  local1 = rotate_y_deg(local1, rotate1y);
  local1 = translate(local1, vec3(8.0+translate1x, translate1y, -60.0f+translate1z));
  mat4 global1 = local1;


  // update uniforms & draw
  glUniformMatrix4fv(proj_mat_location, 1, GL_FALSE, persp_proj.m);
  glUniformMatrix4fv(view_mat_location, 1, GL_FALSE, view.m);
  glUniformMatrix4fv(matrix_location, 1, GL_FALSE, global1.m);
```

```
glDrawArrays(GL_TRIANGLES, 0, teapot_vertex_count);
/*2 - rotating arm*/
mat4 local2 = identity_mat4();
local2 = rotate_y_deg(local2, rotate2z);
local2 = translate(local2, vec3(20.0+translate2x, 4.0, translate2z));
mat4 global2 = global1*local2;
// update uniform & draw
glUniformMatrix4fv(matrix_location, 1, GL_FALSE, global2.m);
glDrawArrays(GL_TRIANGLES, 0, teapot_vertex_count);

/*3 - head*/
mat4 local3 = identity_mat4();
local3 = rotate_y_deg(local3, rotate3z);
local3 = translate(local3, vec3(translate3x, 10.0, 0.0));
mat4 global3 = global1*local3;
// update uniform & draw
glUniformMatrix4fv(matrix_location, 1, GL_FALSE, global3.m);
glDrawArrays(GL_TRIANGLES, 0, teapot_vertex_count);

/*4 - left arm*/
mat4 local4 = identity_mat4();
local4 = rotate_y_deg(local4, 180);
local4 = rotate_x_deg(local4, 180);
local4 = translate(local4, vec3(-8.0, -3.0, translate4z));
mat4 global4 = global1*local4;
// update uniform & draw
glUniformMatrix4fv(matrix_location, 1, GL_FALSE, global4.m);
glDrawArrays(GL_TRIANGLES, 0, teapot_vertex_count);

/*5 - left leg*/
mat4 local5 = identity_mat4();
local5 = rotate_y_deg(local5, 0);
local5 = rotate_x_deg(local5, 180);
local5 = rotate_z_deg(local5, 260);
local5 = translate(local5, vec3(-12.0, -15.0, translate5z));
mat4 global5 = global1*local5;
// update uniform & draw
glUniformMatrix4fv(matrix_location, 1, GL_FALSE, global5.m);
glDrawArrays(GL_TRIANGLES, 0, teapot_vertex_count);

/*6 - right leg*/
mat4 local6 = identity_mat4();
local6 = rotate_y_deg(local6, 0);
local6 = rotate_x_deg(local6, 0);
local6 = rotate_z_deg(local6, 285);
local6 = translate(local6, vec3(5.0, -15.0, translate6z));
mat4 global6 = global1*local6;
// update uniform & draw
glUniformMatrix4fv(matrix_location, 1, GL_FALSE, global6.m);
glDrawArrays(GL_TRIANGLES, 0, teapot_vertex_count);

/*7 - sub left arm*/
mat4 local7 = identity_mat4();
local7 = rotate_y_deg(local7, 0);
local7 = rotate_x_deg(local7, 0);
local7 = rotate_z_deg(local7, rotate7z);
```

```
   local7 = translate(local7, vec3(-14.0, -3.0, translate7z));
   mat4 global7 = global1*local7*local4;
   // update uniform & draw
   glUniformMatrix4fv(matrix_location, 1, GL_FALSE, global7.m);
   glDrawArrays(GL_TRIANGLES, 0, teapot_vertex_count);

   /*8 - sub sub left arm*/
   mat4 local8 = identity_mat4();
   local8 = rotate_y_deg(local8, 180);
   local8 = rotate_x_deg(local8, 0);
   local8 = rotate_z_deg(local8, rotate8z);
   local8 = translate(local8, vec3(5, -3.0, translate8z));
   mat4 global8 = global7*local8;
   // update uniform & draw
   glUniformMatrix4fv(matrix_location, 1, GL_FALSE, global8.m);
   glDrawArrays(GL_TRIANGLES, 0, teapot_vertex_count);

   /*END*/
   glutSwapBuffers();
}



void updateScene() {

   // Placeholder code, if you want to work with framerate
   // Wait until at least 16ms passed since start of last frame (Effectively caps framerate at
       ~60fps)
   static DWORD last_time = 0;
   DWORD curr_time = timeGetTime();
   float delta = (curr_time - last_time) * 0.001f;
   if (delta > 0.03f)
      delta = 0.03f;
   last_time = curr_time;

   rotate2z += 0.2f;
   // Draw the next frame
   glutPostRedisplay();
}


void init()
{
   // Set up the shaders
   GLuint shaderProgramID = CompileShaders();
   // load teapot mesh into a vertex buffer array
   generateObjectBufferTeapot();

}

float color()
{
   float r = ((double)rand() / (RAND_MAX));
   float g = ((double)rand() / (RAND_MAX));
   float b = ((double)rand() / (RAND_MAX));
   return red = r, green = g, blue = b;
```

```
}

// Placeholder code for the keypress
float temp = 1;
void keypress(unsigned char key, int x, int y) {

   if (key == 'p') {
      color();
      rotate2z = rotate2z + temp;
      printf("%f", rotate2z);

   }
   if (key == 'l') {
      color();
      rotate2z = rotate2z - temp;
      printf("%f", rotate2z);

   }
   if (key == '3') {
      color();
      rotate1z = rotate1z + temp;
      printf("%f", rotate1z);
   }

   if (key == '1') {
      color();
      rotate1z = rotate1z - temp;
      printf("%f", rotate1z);
   }

   if (key == 'l') {
      color();
      rotate2z = rotate2z - temp;
      printf("%f", rotate2z);
   }
   if (key == 'd') {
      color();
      translate1x = translate1x + temp;
      printf("%f", translate1x);
   }

   if (key == 'a') {
      color();
      translate1x = translate1x - temp;
      printf("%f", translate1x);
   }
   if (key == 'w') {
      color();
      translate1z = translate1z - temp;
      printf("%f", translate1z);
   }

   if (key == 's') {
      color();
      translate1z = translate1z + temp;
      printf("%f", translate1z);
```

```
    }
    if (key == 'q') {
       color();
       rotate1y = rotate1y + temp;
       printf("%f", rotate1y);
    }
    if (key == 'e') {
       color();
       rotate1y = rotate1y - temp;
       printf("%f", rotate1y);
    }
    if (key == '6') {
       color();
       translate2x = translate2x + temp;
       printf("%f", translate2x);
    }
    if (key == '4') {
       color();
       translate2x = translate2x - temp;
       printf("%f", translate2x);
    }
    if (key == 't') {
       color();
       rotate3z = rotate3z + temp;
       printf("%f", rotate3z);
    }
    if (key == 'y') {
       color();
       rotate3z = rotate3z - temp;
       printf("%f", rotate3z);

    }
    if (key == 'z') {
       color();
       rotate7z = rotate7z + temp;
       printf("%f", rotate7z);
    }
    if (key == 'x') {
       color();
       rotate7z = rotate7z - temp;
       printf("%f", rotate7z);
    }
    if (key == 'c') {
       color();
       rotate8z = rotate8z + temp;
       printf("%f", rotate8z);
    }
    if (key == 'v') {
       color();
       rotate8z = rotate8z - temp;
       printf("%f", rotate8z);

    }

}
```

```
int main(int argc, char** argv) {

   // Set up the window
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
   glutInitWindowSize(width, height);
   glutCreateWindow("Tutorial 4");

   // Tell glut where the display function is
   glutDisplayFunc(display);
   glutIdleFunc(updateScene);
   glutKeyboardFunc(keypress);

   // A call to glewInit() must be done after glut is initialized!
   GLenum res = glewInit();
   // Check for any errors
   if (res != GLEW_OK) {
      fprintf(stderr, "Error: '%s'\n", glewGetErrorString(res));
      return 1;
   }
   // Set up your objects and shaders
   init();
   // Begin infinite event loop
   glutMainLoop();
   return 0;
}
```

3 Static

For this part of the assignmeant I didnt really have to do anything as the default position of the teapots was non-spinning.

4 Two different Perspectives

For this part of the assignmeant I changed the variables in oone of the perspective functions to achieve a different overall perspective

```
mat4 persp_proj = perspective(45.0, (float)width / (float)height, 0.1, 100.0);
   mat4 persp_proj1 = perspective(90.0, (float)width / (float)height, 0.1, 100.0);
```

5 Orthographic Projection

For this part of the assignment I wrote an orthographic function in mathfunc.cpp based on the matrix given in the slides. The function simply implements that matrix and multiplies it depending on the inputs given.

```
mat4 Ort = orthograp(-20,20,-20,20,20,-20);
mat4 orthograp(float left, float right, float near, float far, float top, float bottom ) {
   mat4 m = zero_mat4(); // make sure bottom-right corner is zero
   m.m[0] = 2/(right-left);
   m.m[5] = 2/(top-bottom);
   m.m[10] = -2/(far-near);
   m.m[3] = -(right + left) /(right-left);
```

```
    m.m[7] = -(top + bottom) / (top - bottom);
    m.m[11] = -(far + near)/ (far - near);
    m.m[15] = 1.0f;
    return m;
}
```

6 Look At Function

Look at Function is implemented using the camera position the teapot piosition and the direction in which

```
vec3 cam_pos = { 0.0, 0.0, -40.0 };
vec3 teapot_pos = { 0.0,0.0,0.0 };
vec3 up = {0.0,1.0,0.0};
mat4 La = look_at(cam_pos, teapot_pos, up);
```