**Homework 3 보고서**

**Shadow Mapping**

**이름: 진방달**

**학번: 2016-23145**

**컴퓨터공학부**

**제출일자: 20170605**

|  |  |
| --- | --- |
| **Y** | Create the scene：   * The scene has one ground. * The scene has one model loaded from an obj file (bunny.obj). * Create obj loader by myself. |
| **Y** | Create the Light sources:   * Make two point light sources (green, red) and one directional light sources (blue).       Toggle the Light sources:   * For each light source bind a key to “turn on” and “turn off” the light source.   (F1 to turn on/off light source1)    (F2 to turn on/off light source2)    (F3 to turn on/off light source3) |
| **Y** | Depth Overlay:   * Create overlay image which shows “depth image” from light source. |
| **Y** | Shadow for moving object and light:   * Each shadow map updated when the object (use mouse’s scroll bar to change model’s position) / light object (use keyboard ‘p’ to rotate light sources’ position) changes the position * Implement 3 light sources and all objects’ position change. |

**Instructions:**

**Mouse:**

USE scroll bar to resize scene, push scroll bar to move model

USE left button to rotate model, right button to rotate scene

**Keyboard:**

F1: open/close light1, F2: open/close light2, F3: open/close light3

USE P to (start/end) light rotation

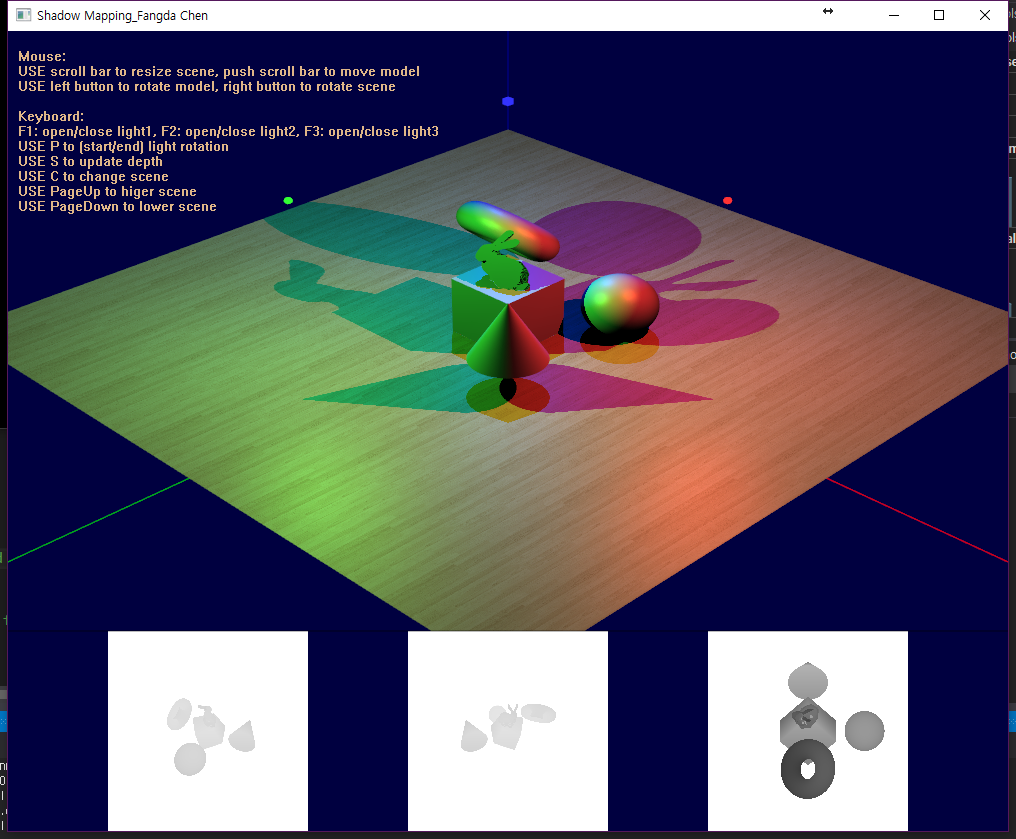
USE S to update depth

USE C to change scene

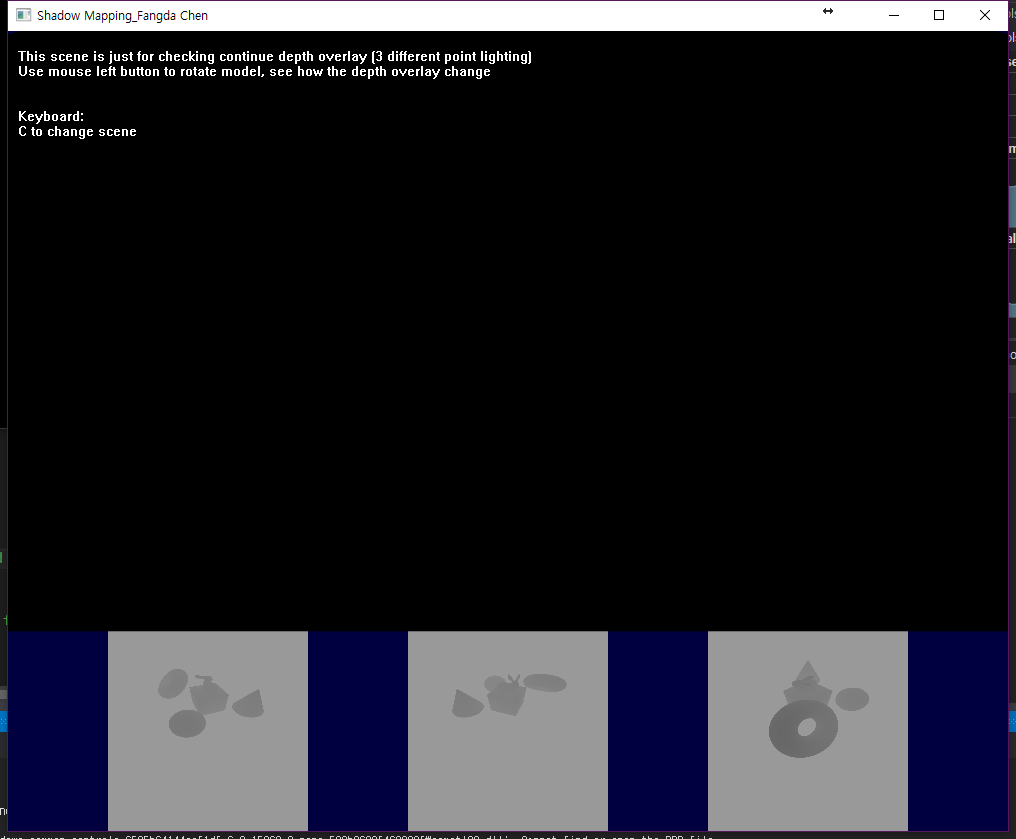
USE PageUp to higer scene

USE PageDown to lower scene

Results:



**<Scene 1>**



**<Scene 2>**

**Source code:**

|  |
| --- |
| Main.cpp  #include "setup.h"  /\*  glActiveTexture(GL\_TEXTURE0) Ordinary texture  glActiveTexture(GL\_TEXTURE1) Shadow texture  \*/  //Initialize texture  void tex\_init()  {  glActiveTexture(GL\_TEXTURE0); // ordinary texture: GL\_TEXTURE0  glTexEnvi(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE);  void\* data;  int w, h;  il\_readImg(L"d0.png", &data, &w, &h);  glGenTextures(1, &tex\_depth[0]);  glBindTexture(GL\_TEXTURE\_2D, tex\_depth[0]);  glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, w, h, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, data);  delete data;  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);  il\_readImg(L"d1.png", &data, &w, &h);  glGenTextures(1, &tex\_depth[1]);  glBindTexture(GL\_TEXTURE\_2D, tex\_depth[1]);  glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, w, h, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, data);  delete data;  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);  il\_readImg(L"d2.png", &data, &w, &h);  glGenTextures(1, &tex\_depth[2]);  glBindTexture(GL\_TEXTURE\_2D, tex\_depth[2]);  glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, w, h, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, data);  delete data;  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);  il\_readImg(L"world.png", &data, &w, &h);  glGenTextures(1, &tex\_walls);  glBindTexture(GL\_TEXTURE\_2D, tex\_walls);  glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, w, h, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, data);  delete data;  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);  glActiveTexture(GL\_TEXTURE1); // shadow texuture: GL\_TEXTURE1  glTexEnvi(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE);  glTexGeni(GL\_S, GL\_TEXTURE\_GEN\_MODE, GL\_EYE\_LINEAR);  glTexGeni(GL\_T, GL\_TEXTURE\_GEN\_MODE, GL\_EYE\_LINEAR);  glTexGeni(GL\_R, GL\_TEXTURE\_GEN\_MODE, GL\_EYE\_LINEAR);  glTexGeni(GL\_Q, GL\_TEXTURE\_GEN\_MODE, GL\_EYE\_LINEAR);  glEnable(GL\_TEXTURE\_GEN\_S);  glEnable(GL\_TEXTURE\_GEN\_T);  glEnable(GL\_TEXTURE\_GEN\_R);  glEnable(GL\_TEXTURE\_GEN\_Q);  glGenTextures(lights\_num, tex\_shadow);  for (int i = 0; i < lights\_num; ++i) {  glBindTexture(GL\_TEXTURE\_2D, tex\_shadow[i]);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_CLAMP\_TO\_BORDER);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_CLAMP\_TO\_BORDER);  GLfloat c[4] = { 1,1,1, 1 };  glTexParameterfv(GL\_TEXTURE\_2D, GL\_TEXTURE\_BORDER\_COLOR, c);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_COMPARE\_MODE, GL\_COMPARE\_R\_TO\_TEXTURE);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_COMPARE\_FUNC, GL\_LEQUAL);  glTexParameteri(GL\_TEXTURE\_2D, GL\_DEPTH\_TEXTURE\_MODE, GL\_LUMINANCE);  glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_DEPTH\_COMPONENT, shadow\_w, shadow\_h, 0, GL\_DEPTH\_COMPONENT, GL\_FLOAT, 0);  }  glActiveTexture(GL\_TEXTURE2); // shadow texuture: GL\_TEXTURE1  glTexEnvi(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE);  for (int i = 0; i < lights\_num; i++)  {  glGenTextures(1, &tex\_sh[i]);  glBindTexture(GL\_TEXTURE\_2D, tex\_sh[i]);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_CLAMP\_TO\_EDGE);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_CLAMP\_TO\_EDGE);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);  glTexParameteri(GL\_TEXTURE\_2D, GL\_DEPTH\_TEXTURE\_MODE, GL\_LUMINANCE);  }  // frame buffer object  glGenFramebuffers(1, &frame\_buffer\_s);  glBindFramebuffer(GL\_DRAW\_FRAMEBUFFER, frame\_buffer\_s);  // glFramebufferTexture2D(GL\_DRAW\_FRAMEBUFFER, GL\_DEPTH\_ATTACHMENT, GL\_TEXTURE\_2D, tex\_shadow, 0);  }  void dlight(glm::vec4& light\_pos, int index) // draw a sphere in light position  {  GLboolean li = glIsEnabled(GL\_LIGHTING);  if (li)  glDisable(GL\_LIGHTING);  GLfloat cc[4];  glGetFloatv(GL\_CURRENT\_COLOR, cc);  if(index == 0)  glColor3f(1.0f, 0.2f, 0.2f);  else if (index == 1)  glColor3f(0.2f, 1.0f, 0.2f);  else if (index == 2)  glColor3f(0.2f, 0.2f, 1.0f);  glMatrixMode(GL\_MODELVIEW);  glPushMatrix();  glTranslatef(light\_pos[0], light\_pos[1], light\_pos[2]);  //draw sphere  if (index < 2 && toggle[index])  glutSolidSphere(0.1f, 50, 50);  else if (index == 2 && toggle[index])  glutSolidCube(0.2f);  glPopMatrix();  if (li)  glEnable(GL\_LIGHTING);  glColor4fv(cc);  }  //Draw a world, a wood floor  void draw\_world()  {  glMatrixMode(GL\_MODELVIEW);  glPushMatrix();  glRotatef(90, -1, 0, 0);  glTranslatef(-10, -10, 0);  floor(20, 3, 100);  glPopMatrix();  }  //Draw all models upon wood floor  void draw\_model()  {  glMatrixMode(GL\_MODELVIEW);  glPushMatrix();  drawobj(numVertex, numFaces, vertices, faces);  glTranslatef(0, 0, 0);  glutSolidCube(2);  glTranslatef(2, 0, 2);  glRotatef(-90, 1, 0, 0);  glutSolidCone(1, 2, 50, 50);  glRotatef(90, 1, 0, 0);  glTranslatef(0, 0, -4);  glutSolidSphere(1, 50, 50);  glTranslatef(-4, 1, 0);  glRotatef(45, 1, 0, 0);  glutSolidTorus(0.5f, 1, 50, 50);  glRotatef(-45, 1, 0, 0);  glPopMatrix();  }  void draw\_tex() // draw a floor in the world  {  glm::vec4 v1(-30, 0, -30, 1), v2(-30, 0, 30, 1), v3(30, 0, 30, 1), v4(30, 0, -30, 1);//四个顶点  // glm::vec4 v1(-50, -50, 0, 1), v2(-50, 50, 0, 1), v3(50, 50, 0, 1), v4(50, -50, 0, 1);//四个顶点  glm::mat4 m = glm::translate(glm::vec3(0.5f, 0.5f, 0.5f)) \* glm::scale(glm::vec3(0.5f, 0.5f, 0.5f)); // change clipping coordinate[-1,+1] to [0,1]  glm::vec4 t;  glColor3f(1.0, 1.0, 1.0);  glBegin(GL\_POLYGON);  glNormal3f(0, 1, 0);  t = m\*shadow\_mat\_p\*shadow\_mat\_v[0] \* v1; // caculate texture coordinate  glTexCoord4fv(&t[0]); glVertex3fv(&v1[0]);  t = m\*shadow\_mat\_p\*shadow\_mat\_v[0] \* v2;  glTexCoord4fv(&t[0]); glVertex3fv(&v2[0]);  t = m\*shadow\_mat\_p\*shadow\_mat\_v[0] \* v3;  glTexCoord4fv(&t[0]); glVertex3fv(&v3[0]);  t = m\*shadow\_mat\_p\*shadow\_mat\_v[0] \* v4;  glTexCoord4fv(&t[0]); glVertex3fv(&v4[0]);  glEnd();  glColor3f(1.0, 1.0, 1.0);  }  void draw2(const glm::mat4& mat\_model, const glm::mat4& mat\_view)  {  if (scene2\_tex == true)  {  scene2\_tex = false;  tex\_init();  }  //very important  glBindFramebuffer(GL\_FRAMEBUFFER, 0);  //shadow[lights\_num]  for (int i = 0; i < lights\_num; i++)  {  glDisable(GL\_LIGHTING);  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  shadow\_mat\_p = glm::perspective(glm::radians(90.0f), 1.0f, 1.0f, 1.0e10f);  shadow\_mat\_v[i] = glm::lookAt(glm::vec3(light\_pos[i]), glm::vec3(0), glm::vec3(0, 1, 0));  glMatrixMode(GL\_PROJECTION);  glPushMatrix();  glLoadMatrixf(&shadow\_mat\_p[0][0]);  glMatrixMode(GL\_MODELVIEW);  glPushMatrix();  glLoadMatrixf(&shadow\_mat\_v[i][0][0]);  glMultMatrixf(&mat\_model[0][0]);  draw\_model();  glMatrixMode(GL\_PROJECTION);  glPopMatrix();  glMatrixMode(GL\_MODELVIEW);  glPopMatrix();  glBindTexture(GL\_TEXTURE\_2D, tex\_sh[i]);  glCopyTexImage2D(GL\_TEXTURE\_2D, 0, GL\_DEPTH\_COMPONENT, 0, 0, get\_frame\_width(), get\_frame\_height(), 0);  glEnable(GL\_TEXTURE\_2D);  }  //divide windows  // glClearColor(1.0f, 1.0f, 1.0f, 1.0f);  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  glViewport(0, 0, 1000, 800);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  glMatrixMode(GL\_MODELVIEW);  glLoadIdentity();  glColor3f(0, 0, 0);  //draw divide line  glBegin(GL\_LINES);  glVertex3f(-1, -0.5, 0);  glVertex3f(1, -0.5, 0);  glEnd();  //3 small windows  for (int i = 0; i < lights\_num; i++)  {  //gap is 100, start from (100, 0)  glViewport(100 \* (i + 1) + 200 \* i, 0, 200, 200);  glLoadIdentity();  gluOrtho2D(0, 200, 0, 200);  glColor3f(0.6, 0.6, 0.6);  glActiveTextureARB(GL\_TEXTURE0);  glEnable(GL\_TEXTURE\_2D);  // if(i<2)  glBindTexture(GL\_TEXTURE\_2D, tex\_sh[i]);  /\*else  glBindTexture(GL\_TEXTURE\_2D, tex\_walls);\*/  glBegin(GL\_POLYGON);  glTexCoord2d(1, 0);  glVertex3f(0, 0, 0);  glTexCoord2d(1, 1);  glVertex3f(0, 200, 0);  glTexCoord2d(0, 1);  glVertex3f(200, 200, 0);  glTexCoord2d(0, 0);  glVertex3f(200, 0, 0);  glEnd();  }  // glBindTexture(GL\_TEXTURE\_2D, tex\_sh[0]);  /\* --------3D-------- \*/  glViewport(0, 200, 1000, 600);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  //Designer annotation  glColor3f(1.0, 1.0, 1.0);  glRasterPos2f(2.0f, 190.0f);  drawString("This scene is just for checking continue depth overlay (3 different point lighting)");  glRasterPos2f(2.0f, 185.0f);  drawString("Use mouse left button to rotate model, see how the depth overlay change");  glRasterPos2f(2.0f, 170.0f);  drawString("Keyboard:");  glRasterPos2f(2.0f, 165.0f);  drawString("C to change scene");    gluPerspective(25, 8 / 6.0, 0.1, 25000);  glMatrixMode(GL\_MODELVIEW);  glLoadIdentity();  gluLookAt(50, 50, 50, 0, 0, 0, 0, 1, 0);  //Draw x,y,z coordinates  glBegin(GL\_LINES);  glColor3f(1.0f, 0.0f, 0.0f);  glVertex3f(0, 0, 0);  glVertex3f(50.0f, 0, 0);  glColor3f(0.0f, 0.0f, 1.0f);  glVertex3f(0, 0, 0);  glVertex3f(0, 50.0f, 0);  glColor3f(0.0f, 1.0f, 0.0f);  glVertex3f(0, 0, 0);  glVertex3f(0, 0, 50.0f);  glEnd();  //-----------------------------second path------------  glEnable(GL\_LIGHTING);  glMatrixMode(GL\_MODELVIEW);  glLoadMatrixf(&mat\_view[0][0]);  draw\_tex();  glMultMatrixf(&mat\_model[0][0]);  draw\_model();  }  void draw1(const glm::mat4& mat\_model, const glm::mat4& mat\_view)  {  if(save\_shadow==true)  tex\_init();  //------------------------First draw -------------------------  glBindFramebuffer(GL\_DRAW\_FRAMEBUFFER, frame\_buffer\_s);  glColorMask(GL\_FALSE, GL\_FALSE, GL\_FALSE, GL\_FALSE);  glViewport(0, 0, shadow\_w, shadow\_h);  // No need light and texture  GLboolean li = glIsEnabled(GL\_LIGHTING);  if (li)  glDisable(GL\_LIGHTING);  glActiveTexture(GL\_TEXTURE1);  glDisable(GL\_TEXTURE\_2D);  glActiveTexture(GL\_TEXTURE0);  glDisable(GL\_TEXTURE\_2D);  //9:24  glActiveTexture(GL\_TEXTURE2);  glDisable(GL\_TEXTURE\_2D);  glEnable(GL\_CULL\_FACE);  glCullFace(GL\_FRONT);  glMatrixMode(GL\_PROJECTION);  glPushMatrix();  glMatrixMode(GL\_MODELVIEW);  glPushMatrix();  glm::mat4 shadow\_mat\_pd = glm::ortho(-5.0f, 5.0f, -5.0f, 5.0f, -5.0f, 5.0f);  glm::mat4 shadow\_mat\_pp = glm::perspective(glm::radians(100.0f), 1.0f, 1.0f, 1.0e10f);  glm::vec3 cen = glm::vec3(get\_mat\_model()\*glm::vec4(0, 0, 0, 1));  glm::mat4 shadow\_mat\_v[lights\_num];  for (int i = 0; i < lights\_num; ++i)  {  glFramebufferTexture2D(GL\_DRAW\_FRAMEBUFFER, GL\_DEPTH\_ATTACHMENT, GL\_TEXTURE\_2D, tex\_shadow[i], 0);  glClear(GL\_DEPTH\_BUFFER\_BIT);  if (i < 2)  shadow\_mat\_v[i] = glm::lookAt(glm::vec3(light\_pos[i]), cen, glm::vec3(0, 1, 0));  else  shadow\_mat\_v[i] = glm::lookAt(glm::vec3(cen), glm::vec3(-light\_pos[i]), glm::vec3(0, 1, 0));  //point lighting  if (i < 2)  {  glMatrixMode(GL\_PROJECTION);  glLoadMatrixf(&shadow\_mat\_pp[0][0]);  }  //directional lighting  else  {  glMatrixMode(GL\_PROJECTION);  glLoadMatrixf(&shadow\_mat\_pd[0][0]);  }  glMatrixMode(GL\_MODELVIEW);  glLoadMatrixf(&shadow\_mat\_v[i][0][0]);  draw\_world();  glMultMatrixf(&mat\_model[0][0]);  draw\_model();  }  glMatrixMode(GL\_PROJECTION);  glPopMatrix();  glMatrixMode(GL\_MODELVIEW);  glPopMatrix();  glDisable(GL\_CULL\_FACE);  glCullFace(GL\_BACK);  //shadow[lights\_num]  /\*glBindFramebuffer(GL\_FRAMEBUFFER, 0);  for (int i = 0; i < lights\_num; i++)  {  glDisable(GL\_LIGHTING);  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  shadow\_mat\_p = glm::perspective(glm::radians(90.0f), 1.0f, 1.0f, 1.0e10f);  shadow\_mat\_v[i] = glm::lookAt(glm::vec3(light\_pos[i]), glm::vec3(0), glm::vec3(0, 1, 0));  glMatrixMode(GL\_PROJECTION);  glPushMatrix();  glLoadMatrixf(&shadow\_mat\_p[0][0]);  glMatrixMode(GL\_MODELVIEW);  glPushMatrix();  glLoadMatrixf(&shadow\_mat\_v[i][0][0]);  glMultMatrixf(&mat\_model[0][0]);  draw\_model();  glMatrixMode(GL\_PROJECTION);  glPopMatrix();  glMatrixMode(GL\_MODELVIEW);  glPopMatrix();  glActiveTexture(GL\_TEXTURE2);  glBindTexture(GL\_TEXTURE\_2D, tex\_sh[i]);  glCopyTexImage2D(GL\_TEXTURE\_2D, 0, GL\_DEPTH\_COMPONENT, 0, 0, get\_frame\_width(), get\_frame\_height(), 0);  glEnable(GL\_TEXTURE\_2D);  }  \*/  if (save\_shadow) {  save\_shadow = false;  glActiveTexture(GL\_TEXTURE1);  GLfloat\* data = new GLfloat[shadow\_w\*shadow\_h];  for (int i = 0; i < lights\_num; ++i) {  glBindTexture(GL\_TEXTURE\_2D, tex\_shadow[i]);  glGetTexImage(GL\_TEXTURE\_2D, 0, GL\_DEPTH\_COMPONENT, GL\_FLOAT, data);//get texture data  wchar\_t ss[50]; swprintf(ss, L"d%d.png", i);  il\_saveImgDep(ss, data, shadow\_w, shadow\_w);  }  delete[] data;  }  glViewport(0, 0, get\_frame\_width(), get\_frame\_height());  glColorMask(GL\_TRUE, GL\_TRUE, GL\_TRUE, GL\_TRUE);  if (li)  glEnable(GL\_LIGHTING);  //------------------------------- second draw scene ----------------------------  glBindFramebuffer(GL\_FRAMEBUFFER, 0);  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  glViewport(0, 0, 1000, 800);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  glMatrixMode(GL\_MODELVIEW);  glLoadIdentity();  glColor3f(0, 0, 0);  //draw divide line  glBegin(GL\_LINES);  glVertex3f(-1, -0.5, 0);  glVertex3f(1, -0.5, 0);  glEnd();  //3 small windows  glDisable(GL\_LIGHTING);  for (int i = 0; i < lights\_num; i++)  {  //gap is 100, start from (100, 0)  glViewport(100 \* (i + 1) + 200 \* i, 0, 200, 200);  glLoadIdentity();  gluOrtho2D(0, 200, 0, 200);  glColor3f(1.0, 1.0, 1.0);  glActiveTextureARB(GL\_TEXTURE0);  glBindTexture(GL\_TEXTURE\_2D, tex\_depth[i]);  glActiveTexture(GL\_TEXTURE0);  glEnable(GL\_TEXTURE\_2D);  //glActiveTextureARB(GL\_TEXTURE1);  //glBindTexture(GL\_TEXTURE\_2D, tex\_shadow[i]);  //glActiveTexture(GL\_TEXTURE1);  //glEnable(GL\_TEXTURE\_2D);  glBegin(GL\_POLYGON);  glTexCoord2d(1, 0);  glVertex3f(0, 0, 0);  glTexCoord2d(1, 1);  glVertex3f(0, 200, 0);  glTexCoord2d(0, 1);  glVertex3f(200, 200, 0);  glTexCoord2d(0, 0);  glVertex3f(200, 0, 0);  glEnd();  }  glBindTexture(GL\_TEXTURE\_2D, tex\_walls);  glActiveTexture(GL\_TEXTURE0);  /\* --------3D-------- \*/  glViewport(0, 200, 1000, 600);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  //Designer annotation  glColor3f(1.0, 1.0, 1.0);  glRasterPos2f(2.0f, 190.0f);  drawString("Mouse:");  glRasterPos2f(2.0f, 185.0f);  drawString("USE scroll bar to resize scene, push scroll bar to move model");  glRasterPos2f(2.0f, 180.0f);  drawString("USE left button to rotate model, right button to rotate scene");  glRasterPos2f(2.0f, 170.0f);  drawString("Keyboard:");  glRasterPos2f(2.0f, 165.0f);  drawString("F1: open/close light1, F2: open/close light2, F3: open/close light3");  glRasterPos2f(2.0f, 160.0f);  drawString("USE P to (start/end) light rotation");  glRasterPos2f(2.0f, 155.0f);  drawString("USE S to update depth");  glRasterPos2f(2.0f, 150.0f);  drawString("USE C to change scene");  glRasterPos2f(2.0f, 145.0f);  drawString("USE PageUp to higer scene");  glRasterPos2f(2.0f, 140.0f);  drawString("USE PageDown to lower scene");  gluPerspective(25, 8 / 6.0, 0.1, 25000);  glMatrixMode(GL\_MODELVIEW);  glLoadIdentity();  gluLookAt(50, 50, 50, 0, 0, 0, 0, 1, 0);    //Draw x,y,z coordinates  glBegin(GL\_LINES);  glColor3f(1.0f, 0.0f, 0.0f);  glVertex3f(0, 0, 0);  glVertex3f(50.0f, 0, 0);  glColor3f(0.0f, 0.0f, 1.0f);  glVertex3f(0, 0, 0);  glVertex3f(0, 50.0f, 0);  glColor3f(0.0f, 1.0f, 0.0f);  glVertex3f(0, 0, 0);  glVertex3f(0, 0, 50.0f);  glEnd();  glEnable(GL\_LIGHTING);  // 1 Environment lighting  for (int i = 0; i < lights\_num; ++i)  glDisable(GL\_LIGHT0 + i);  glActiveTexture(GL\_TEXTURE1);  glDisable(GL\_TEXTURE\_2D);  glActiveTexture(GL\_TEXTURE0);  glEnable(GL\_TEXTURE\_2D);  glMatrixMode(GL\_MODELVIEW);  glLoadMatrixf(&mat\_view[0][0]);  draw\_world();  glMultMatrixf(&mat\_model[0][0]);  draw\_model();  // 2 Point Lighting  GLfloat la[4];  glGetFloatv(GL\_LIGHT\_MODEL\_AMBIENT, la);  float gac[4] = { 0,0,0,1 };  glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, gac); // black  glActiveTexture(GL\_TEXTURE1);  glEnable(GL\_TEXTURE\_2D);  glActiveTexture(GL\_TEXTURE0);  glEnable(GL\_TEXTURE\_2D);  glDepthFunc(GL\_EQUAL);  glBlendFunc(GL\_ONE, GL\_ONE);  for (int i = 0; i < lights\_num; ++i)  {  if (toggle[i] == true)  {  glActiveTexture(GL\_TEXTURE1);  glBindTexture(GL\_TEXTURE\_2D, tex\_shadow[i]);  // When the eye planes are specified, the GL will automatically post-multiply them  // with the inverse of the current modelview matrix.  glMatrixMode(GL\_MODELVIEW);  glLoadMatrixf(&mat\_view[0][0]); //glLoadIdentity();    glm::mat4 mat;  if (i < 2)  {  mat = glm::translate(glm::vec3(0.5f, 0.5f, 0.5f)) \* glm::scale(glm::vec3(0.5f, 0.5f, 0.5f)) \* shadow\_mat\_pp \* shadow\_mat\_v[i];  }  else  {  mat = glm::translate(glm::vec3(0.5f, 0.5f, 0.5f)) \* glm::scale(glm::vec3(0.5f, 0.5f, 0.5f)) \* shadow\_mat\_pd \* shadow\_mat\_v[i];  }  mat = glm::transpose(mat);  glTexGenfv(GL\_S, GL\_EYE\_PLANE, &mat[0][0]);  glTexGenfv(GL\_T, GL\_EYE\_PLANE, &mat[1][0]);  glTexGenfv(GL\_R, GL\_EYE\_PLANE, &mat[2][0]);  glTexGenfv(GL\_Q, GL\_EYE\_PLANE, &mat[3][0]);  glMatrixMode(GL\_MODELVIEW);  glLoadMatrixf(&mat\_view[0][0]);  glLightfv(GL\_LIGHT0 + i, GL\_POSITION, &light\_pos[i][0]);  glEnable(GL\_LIGHT0 + i);  glActiveTexture(GL\_TEXTURE0);  glEnable(GL\_TEXTURE\_2D);  glMatrixMode(GL\_MODELVIEW);  glLoadMatrixf(&mat\_view[0][0]);  draw\_world();  glMultMatrixf(&mat\_model[0][0]);  glDisable(GL\_TEXTURE\_2D);  draw\_model();  glDisable(GL\_LIGHT0 + i);  }  }  // glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, la); // Regain environment lighting  glDepthFunc(GL\_LESS);  glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);  // 3 draw sphere in light position  glActiveTexture(GL\_TEXTURE1);  glDisable(GL\_TEXTURE\_2D);  glActiveTexture(GL\_TEXTURE0);  glDisable(GL\_TEXTURE\_2D);  glMatrixMode(GL\_MODELVIEW);  glLoadMatrixf(&mat\_view[0][0]);  for (int i = 0; i < lights\_num; ++i)  dlight(light\_pos[i], i);  if (light\_rotate)  {  for (int i = 0; i < lights\_num; ++i)  {  light\_pos[i] = glm::rotate(glm::radians(1.0f), glm::vec3(0, 1, 0)) \* light\_pos[i];  }  }  }  void key\_p()  {  light\_rotate = !light\_rotate;  }  int main(void)  {  init\_win(1000, 800, "Shadow Mapping\_Fangda Chen", "msyh.ttf");  init\_gl();  init\_light();  readfile("bunny.obj", &numVertex, &numFaces, vertices, faces);  set\_mat\_model(glm::translate(glm::vec3(0, 1, 0)));  set\_mat\_view(glm::lookAt(glm::vec3(25, 25, 25), glm::vec3(0, 0, 0), glm::vec3(0, 1, 0)));  add\_key\_callback('P', key\_p, L"Rotate the light");  renderLoop(draw1, draw2);  glutMainLoop();  } |

|  |
| --- |
| Setup.h  #include <iostream>  #include <fstream>  #include <string>  #include <algorithm>  #include <map>  #include <cmath>  #include <cwchar>  #include <cstdio>  #include <ctime>  #include <windows.h>  #include <omp.h>  #include "GL/glew.h"  #include "GL/freeglut.h"  #include "GLFW/glfw3.h"  #include "il\_inc.h"  #define GLM\_FORCE\_RADIANS  #include "glm/glm.hpp"  #include "glm/ext.hpp"  using namespace std;  #pragma comment (lib, "glew32.lib")  #pragma comment (lib, "freeglut.lib")  #pragma comment (lib, "glfw3dll.lib")  #pragma comment (lib, "glu32.lib") // link OpenGL Utility lib  #pragma comment (lib, "opengl32.lib") // link Microsoft OpenGL lib  #pragma warning( disable : 4996 ) // disable warning of sprintf,swprintf  bool toggle[3];  bool depth = true;  bool save\_shadow = true;  const int lights\_num = 3;  GLuint tex\_walls, tex\_depth[3], tex\_shadow[lights\_num];  GLuint frame\_buffer\_s;  const int shadow\_w = 2048, shadow\_h = 2048;  glm::vec4 light\_pos[lights\_num]; // 3 lights' world position  //Draw2  glm::mat4 shadow\_mat\_p; // shadow map projection  glm::mat4 shadow\_mat\_v[lights\_num]; // shadow map view  GLuint tex\_sh[lights\_num]; // tex\_sh[3]  int choose\_scene = 1;  bool scene2\_tex = true;  bool light\_rotate = false;  void \*font = GLUT\_BITMAP\_8\_BY\_13;  float vertices[5000][3]; //obj file information  int faces[5000][3];  int numVertex;  int numFaces;  class keyFunc { public: void(\*f)(); const wchar\_t\* s; };  static glm::mat4 mat\_model, mat\_view;// transformation matrix  static glm::mat4 mat\_projection; // projection matrix  static float speed\_scale = 0.2f; // interactive speed factor  static float frustum\_fovy = 45; // fovy of frustum  static GLFWwindow\* curr\_window; // current window  static bool help\_display; // should display help content or not  static int font\_size = 16; // font size in pixels  static bool fps\_display = true; // display fps at lover left or not  static std::map<int, keyFunc> key\_funcs; // the key-function map  //Draw annotation  void drawString(const char\* str)  {  static int isFirstCall = 1;  static GLuint lists;  if (isFirstCall)  {  isFirstCall = 0;  lists = glGenLists(128);  wglUseFontBitmaps(wglGetCurrentDC(), 0, 128, lists);  }  for (; \*str != '\0'; ++str)  glCallList(lists + \*str);  }  //file reader  void readfile(char\* obj, int\* numV, int\* numF, float vertices[5000][3], int faces[5000][3])  {  int numVertex = 0;  int numFaces = 0;  char line[256];//get line  //load model  FILE\* fp = fopen(obj, "r");  if (fp == NULL)  {  printf("%s file can not open", obj);  exit(1);  }  while (!feof(fp))  {  fgets(line, 256, fp);  if (line[0] == 'v')  {  if (line[1] == ' ') //vertex:v \*\*\*  numVertex++;  }  else if (line[0] == 'f') { //face:f \*\*\*  numFaces++;  }  }  \*numV = numVertex;  \*numF = numFaces;  //back to start point of file  rewind(fp);  printf("%s's infomation:\n", obj);  printf("number of v : %d\n", numVertex);  printf("number of f : %d\n", numFaces);  int IdxVertex = 0;  int IdxFace = 0;  int length;  while (!feof(fp))  {  fgets(line, 256, fp);  if (line[0] == 'v')  {  if (line[1] == ' ')  {  length = strlen(line) + 1;  fseek(fp, -length, SEEK\_CUR); //한줄 앞으로  float x, y, z;  fscanf(fp, "%s %f %f %f", line, &x, &y, &z);  vertices[IdxVertex][0] = x;  vertices[IdxVertex][1] = y;  vertices[IdxVertex][2] = z;  IdxVertex++;  }  }  else if (line[0] == 'f')  {  length = strlen(line) + 1;  fseek(fp, -length, SEEK\_CUR); //한줄 앞으로  int x1, x2, y1, y2, z1, z2;  fscanf(fp, "%s %d//%d %d//%d %d//%d", line, &x1, &x2, &y1, &y2, &z1, &z2);  faces[IdxFace][0] = x1 - 1;  faces[IdxFace][1] = y1 - 1;  faces[IdxFace][2] = z1 - 1;  IdxFace++;  }  }  }  //Draw object  void drawobj(int numV, int numF, float vertices[5000][3], int faces[5000][3])  {  glPushMatrix();  glTranslatef(0.0, 1.6, 0.0);  glScalef(10, 10, 10);  glColor3f(1.0, 1.0, 1.0);  for (int i = 0; i < numF - 1; i++)  {  float p1[3] = { vertices[faces[i][0]][0], vertices[faces[i][0]][1], vertices[faces[i][0]][2] };  float p2[3] = { vertices[faces[i][1]][0], vertices[faces[i][1]][1], vertices[faces[i][1]][2] };  float p3[3] = { vertices[faces[i][2]][0], vertices[faces[i][2]][1], vertices[faces[i][2]][2] };  glBegin(GL\_TRIANGLES);  glVertex3fv(p1);  glVertex3fv(p2);  glVertex3fv(p3);  glEnd();  }  glPopMatrix();  }  //Draw the floor with tex-repeat and subdivision  void floor(float len, float tex\_repeat, int subdivision)  {  GLfloat color[] = { .8f, 0.8f, 0.8f, 1 };  glMaterialfv(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE, color);  glMatrixMode(GL\_MODELVIEW);  glPushMatrix();  glScalef(len, len, 1);  glNormal3f(0, 0, 1);  float u = 1.0f / subdivision;  float tu = tex\_repeat / subdivision;  for (int i = 0; i < subdivision; ++i)  for (int j = 0; j < subdivision; ++j) {  float ox = u\*i, oy = u\*j;  float tox = tu\*i, toy = tu\*j;  glBegin(GL\_POLYGON);  glTexCoord2f(tox, toy);  glVertex3f(ox, oy, 0);  glTexCoord2f(tox + tu, toy);  glVertex3f(ox + u, oy, 0);  glTexCoord2f(tox + tu, toy + tu);  glVertex3f(ox + u, oy + u, 0);  glTexCoord2f(tox, toy + tu);  glVertex3f(ox, oy + u, 0);  glEnd();  }  glPopMatrix();  }  //----------------------------------- get , set ----------------------------------  const glm::mat4& get\_mat\_model() {  return mat\_model;  }  void set\_mat\_model(const glm::mat4& mat) {  mat\_model = mat;  }  const glm::mat4& get\_mat\_view() {  return mat\_view;  }  void set\_mat\_view(const glm::mat4& mat) {  mat\_view = mat;  }  const glm::mat4& get\_mat\_projection() {  return mat\_projection;  }  int get\_frame\_width() {  int width, height;  glfwGetFramebufferSize(curr\_window, &width, &height);  return width;  }  int get\_frame\_height() {  int width, height;  glfwGetFramebufferSize(curr\_window, &width, &height);  return height;  }  void get\_frame\_size(int\* width, int\* height) {  glfwGetFramebufferSize(curr\_window, width, height);  }  // do not use F1,WSAD,Up,Down,Left,Right,Home,End,PageUp,PageDown  void add\_key\_callback(int key, void(\*func)(), const wchar\_t\* description)  {  if (func == 0) return;  keyFunc kf;  kf.f = func;  kf.s = description;  key\_funcs[key] = kf;  }  //----------------------------------- callbacks -----------------------------------  void callback\_error(int error, const char\* description)  {  std::cout << "GLFW Error code: " << error  << "\t\tDescription: " << description << '\n';  std::cin.get(); // hold the screen  }  // at pressent, this is the same as window size callback  void callback\_frameBufferSize(GLFWwindow\* window, int width, int height)  {  glViewport(0, 0, width, height);  glMatrixMode(GL\_PROJECTION);  mat\_projection = glm::perspective(  glm::radians(frustum\_fovy), float(width) / height, 1.0f, 1.0e10f);  glLoadMatrixf(&mat\_projection[0][0]);  glMatrixMode(GL\_MODELVIEW);  }  void trackball(float\* theta, glm::vec3\* normal, float ax, float ay, float bx, float by, float r)  {  float r2 = r \* 0.9f;  float da = std::sqrt(ax\*ax + ay\*ay);  float db = std::sqrt(bx\*bx + by\*by);  if (std::max(da, db) > r2) {  float dx, dy;  if (da > db) {  dx = (r2 / da - 1)\*ax;  dy = (r2 / da - 1)\*ay;  }  else {  dx = (r2 / db - 1)\*bx;  dy = (r2 / db - 1)\*by;  }  ax += dx; ay += dy; bx += dx; by += dy;  }  float az = std::sqrt(r\*r - (ax\*ax + ay\*ay));  float bz = std::sqrt(r\*r - (bx\*bx + by\*by));  glm::vec3 a = glm::vec3(ax, ay, az);  glm::vec3 b = glm::vec3(bx, by, bz);  \*theta = std::acos(glm::dot(a, b) / (r\*r));  \*normal = glm::cross(a, b);  }  // action: press, release, repeat mods: mod\_shift,ctrl,alt  void callback\_key(GLFWwindow\* window, int key, int scancode, int action, int mods)  {  if (action != GLFW\_RELEASE) {  switch (key)  {  case GLFW\_KEY\_PAGE\_UP: {  glm::vec3 v = glm::vec3(mat\_view\*glm::vec4(0, 1, 0, 0));  mat\_view = glm::translate(-speed\_scale\*v) \* mat\_view; }  break;  case GLFW\_KEY\_PAGE\_DOWN: {  glm::vec3 v = glm::vec3(mat\_view\*glm::vec4(0, 1, 0, 0));  mat\_view = glm::translate(speed\_scale\*v) \* mat\_view; }  break;  case GLFW\_KEY\_S: {  save\_shadow = true;  }  break;  case GLFW\_KEY\_C: {  if (choose\_scene == 1)  {  scene2\_tex = true;  choose\_scene = 2;  }  else if (choose\_scene == 2)  {  scene2\_tex = true;  choose\_scene = 1;  }  }  break;  case GLFW\_KEY\_F1: {  if (toggle[0] == 0)  toggle[0] = 1;  else  toggle[0] = 0;  }  break;  case GLFW\_KEY\_F2: {  if (toggle[1] == 0)  toggle[1] = 1;  else  toggle[1] = 0;  }  break;  case GLFW\_KEY\_F3: {  if (toggle[2] == 0)  toggle[2] = 1;  else  toggle[2] = 0;  }  break;  default:  if (key\_funcs.find(key) != key\_funcs.end()) {  if (key\_funcs[key].f) (\*key\_funcs[key].f)();  }  }  }  }  // button: left, right, mid action: press, release  void callback\_mousePress(GLFWwindow\* window, int button, int action, int mods)  {  // if (x > 100)  {  if (action == GLFW\_PRESS && button == GLFW\_MOUSE\_BUTTON\_RIGHT) {  glfwSetInputMode(window, GLFW\_CURSOR, GLFW\_CURSOR\_DISABLED);  }  if (action == GLFW\_RELEASE && button == GLFW\_MOUSE\_BUTTON\_RIGHT) {  glfwSetInputMode(window, GLFW\_CURSOR, GLFW\_CURSOR\_NORMAL);  }  }  }  void callback\_scroll(GLFWwindow\* window, double xoffset, double yoffset)  {  mat\_view = glm::translate(glm::vec3(0, 0, -speed\_scale \* 5 \* float(yoffset))) \* mat\_view;  }  // xpos,ypos: the new xy-coordinate, in screen coordinates, of the cursor  void callback\_mouseMove(GLFWwindow\* window, double xpos, double ypos)  {  static double xpos\_last, ypos\_last;  ypos = get\_frame\_height() - ypos; // window use upper left as origin, but gl use lower left  // if (xpos > 100)  {  if (glfwGetMouseButton(window, GLFW\_MOUSE\_BUTTON\_LEFT) == GLFW\_PRESS)  {  float dx = float(xpos - xpos\_last), dy = float(ypos - ypos\_last);  if (glfwGetKey(window, GLFW\_KEY\_LEFT\_CONTROL) != GLFW\_RELEASE) { // key left Ctrl is pressed  mat\_view \*= glm::rotate(speed\_scale / 50 \* dx, glm::vec3(0, 1, 0));  glm::vec3 v = glm::vec3(glm::affineInverse(mat\_view)\*glm::vec4(1, 0, 0, 0));  mat\_view \*= glm::rotate(-speed\_scale / 50 \* dy, v);  }  else {  float theta; glm::vec3 n;  int width, height; glfwGetFramebufferSize(window, &width, &height);  trackball(&theta, &n,  float(xpos\_last) - width / 2.0f, float(ypos\_last) - height / 2.0f,  float(xpos) - width / 2.0f, float(ypos) - height / 2.0f, std::min(width, height) / 4.0f);  glm::vec3 normal = glm::vec3(  glm::affineInverse(mat\_model) \* glm::affineInverse(mat\_view)  \* glm::vec4(n.x, n.y, n.z, 0));  mat\_model \*= glm::rotate(theta, normal);  }  }  if (glfwGetMouseButton(window, GLFW\_MOUSE\_BUTTON\_RIGHT) == GLFW\_PRESS)  {  float dx = float(xpos - xpos\_last), dy = float(ypos - ypos\_last);  if (dy != 0)  mat\_view = glm::rotate(-speed\_scale / 50 \* dy, glm::vec3(1, 0, 0)) \* mat\_view;  if (dx != 0)  mat\_view = glm::rotate(speed\_scale / 50 \* dx, glm::vec3(mat\_view \* glm::vec4(0, 1, 0, 0))) \* mat\_view;  }  if (glfwGetMouseButton(window, GLFW\_MOUSE\_BUTTON\_MIDDLE) == GLFW\_PRESS)  {  float dx = float(xpos - xpos\_last), dy = float(ypos - ypos\_last);  if (glfwGetKey(window, GLFW\_KEY\_LEFT\_CONTROL) != GLFW\_RELEASE) { // key left Ctrl is pressed  mat\_view = glm::translate(glm::vec3(speed\_scale / 5 \* dx, speed\_scale / 5 \* dy, 0)) \* mat\_view;  }  else {  glm::vec4 v = glm::affineInverse(mat\_view) \* glm::vec4(dx, dy, 0, 0);  mat\_model = glm::translate(glm::vec3(speed\_scale / 5 \* v)) \* mat\_model;  }  }  xpos\_last = xpos; ypos\_last = ypos;  }  }  //----------------------------------- utilities -----------------------------------  /\* hue:0-360; saturation:0-1; lightness:0-1  \* hue: red(0) -> green(120) -> blue(240) -> red(360)  \* saturation: gray(0) -> perfect colorful(1)  \* lightness: black(0) -> perfect colorful(0.5) -> white(1)  \*/  void hsl\_to\_rgb(float h, float s, float l, float\* rgb)  {  if (s == 0) { rgb[0] = rgb[1] = rgb[2] = l; return; }  float q, p, hk, t[3];  if (l < 0.5f) { q = l \* (1 + s); }  else { q = l + s - l \* s; }  p = 2 \* l - q;  hk = h / 360;  t[0] = hk + 1 / 3.0f;  t[1] = hk;  t[2] = hk - 1 / 3.0f;  for (int i = 0; i < 3; ++i) {  if (t[i] < 0) { t[i] += 1; }  else if (t[i] > 1) { t[i] -= 1; }  }  for (int i = 0; i < 3; ++i) {  if (t[i] < 1 / 6.0f) { rgb[i] = p + (q - p) \* 6 \* t[i]; }  else if (t[i] < 1 / 2.0f) { rgb[i] = q; }  else if (t[i] < 2 / 3.0f) { rgb[i] = p + (q - p) \* 6 \* (2 / 3.0f - t[i]); }  else { rgb[i] = p; }  }  }  float rgb\_to\_gray(float r, float g, float b)  {  return r\*0.299f + g\*0.587f + b\*0.114f;  }  //----------------------------------- initialize ----------------------------------  void init\_win(int width, int height, const char\* tile, const char\* font\_file)  {  // glfw init  glfwSetErrorCallback(callback\_error);  if (!glfwInit()) {  std::cout << "GLFW init Error";  std::cin.get(); // hold the screen  }  // create window  glfwWindowHint(GLFW\_SAMPLES, 8); // anti-aliase, the RGBA,depth,stencil are set by default  const GLFWvidmode\* mods = glfwGetVideoMode(glfwGetPrimaryMonitor());  curr\_window = glfwCreateWindow(width, height, tile, 0, 0);  if (!curr\_window) {  std::cout << "Create window Error";  std::cin.get(); // hold the screen  }  // window at center of screem  glfwSetWindowPos(curr\_window,  std::max(4, mods->width / 2 - width / 2), std::max(24, mods->height / 2 - height / 2));  glfwMakeContextCurrent(curr\_window);  glfwSetFramebufferSizeCallback(curr\_window, callback\_frameBufferSize);  glfwSetKeyCallback(curr\_window, callback\_key);  glfwSetCursorPosCallback(curr\_window, callback\_mouseMove);  glfwSetScrollCallback(curr\_window, callback\_scroll);  glfwSetMouseButtonCallback(curr\_window, callback\_mousePress);  // glut init  int argc = 0;  glutInit(&argc, NULL);  // glew init, have to be after the GL context has been created  GLenum err = glewInit();  if (GLEW\_OK != err) {  std::cout << "GLEW init Error: " << glewGetErrorString(err);  std::cin.get(); // hold the screen  }  // image library init  il\_init();  }  void init\_light()  {  GLfloat vec4f[4] = { 0 };  vec4f[0] = 1.0f; vec4f[1] = 0.2f; vec4f[2] = 0.2f;  glLightfv(GL\_LIGHT0, GL\_DIFFUSE, vec4f); // red  glLightfv(GL\_LIGHT0, GL\_SPECULAR, vec4f);  vec4f[0] = 0.2f; vec4f[1] = 1.0f; vec4f[2] = 0.2f;  glLightfv(GL\_LIGHT1, GL\_DIFFUSE, vec4f); // green  glLightfv(GL\_LIGHT1, GL\_SPECULAR, vec4f);  vec4f[0] = 0.0f; vec4f[1] = 0.2f; vec4f[2] = 1.0f;  glLightfv(GL\_LIGHT2, GL\_DIFFUSE, vec4f); // blue  glLightfv(GL\_LIGHT2, GL\_SPECULAR, vec4f);  vec4f[0] = 0; vec4f[1] = 0; vec4f[2] = 0;  glLightfv(GL\_LIGHT0, GL\_AMBIENT, vec4f); // black AMBIENT  glLightfv(GL\_LIGHT1, GL\_AMBIENT, vec4f);  glLightfv(GL\_LIGHT2, GL\_AMBIENT, vec4f);  float gac2[4] = { 0.15f,0.15f,0.15f,1 };  glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, gac2);  light\_pos[0] = glm::vec4(5, 6, -2, 1);  light\_pos[1] = glm::vec4(-2, 6, 5, 1);  light\_pos[2] = glm::vec4(-3, 6, -3, 1);  }  void init\_gl()  {  toggle[0] = 1;  toggle[1] = 1;  toggle[2] = 1;  // projection matrix  glMatrixMode(GL\_PROJECTION);  int w, h; get\_frame\_size(&w, &h);  mat\_projection = glm::perspective(glm::radians(frustum\_fovy), float(w) / h, 1.0f, 1.0e10f);  glLoadMatrixf(&mat\_projection[0][0]);  // model-view matrix  glMatrixMode(GL\_MODELVIEW);  glLoadIdentity();  // color  glClearColor(0, 0, 0.25f, 1);  glColor4f(.5f, .5f, .5f, 1);  glShadeModel(GL\_SMOOTH);  // material  GLfloat c[] = { .7f, .7f, .7f, 1 };  glMaterialfv(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE, c); // front, gray  c[0] = .4f; c[1] = .4f; c[2] = .4f;  glMaterialfv(GL\_FRONT, GL\_SPECULAR, c);  glMaterialf(GL\_FRONT, GL\_SHININESS, 50);  c[0] = 0; c[1] = 0; c[2] = 0;  glMaterialfv(GL\_BACK, GL\_AMBIENT\_AND\_DIFFUSE, c); // back, black  // lighting, light0  GLfloat vec4f[] = { 1, 1, 1, 1 };  glLightfv(GL\_LIGHT0, GL\_DIFFUSE, vec4f); // white DIFFUSE, SPECULAR  glLightfv(GL\_LIGHT0, GL\_SPECULAR, vec4f);  vec4f[0] = .0f; vec4f[1] = .0f; vec4f[2] = .0f;  glLightfv(GL\_LIGHT0, GL\_AMBIENT, vec4f); // black AMBIENT  glLightModeli(GL\_LIGHT\_MODEL\_LOCAL\_VIEWER, GL\_TRUE); // LOCAL\_VIEWER  glLightModeli(GL\_LIGHT\_MODEL\_TWO\_SIDE, GL\_FALSE); // single side  vec4f[0] = 0.25f; vec4f[1] = 0.25f; vec4f[2] = 0.25f;  glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, vec4f); // global AMBIENT lighting, gray  //glEnable(GL\_CULL\_FACE); glCullFace(GL\_BACK); glFrontFace(GL\_CCW);  glDisable(GL\_CULL\_FACE);  glEnable(GL\_LIGHTING);  glEnable(GL\_LIGHT0);  glEnable(GL\_DEPTH\_TEST);  // blending  glEnable(GL\_BLEND);  glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);  glEnable(GL\_NORMALIZE);  glEnable(GL\_POINT\_SMOOTH);  glEnable(GL\_LINE\_SMOOTH);  }  void renderLoop(void(\*draw1)(const glm::mat4&, const glm::mat4&), void(\*draw2)(const glm::mat4&, const glm::mat4&))  {  static double t1, t2, t3;  #define TIME\_START(n) t##n=omp\_get\_wtime()  #define TIME\_END(n) t##n=omp\_get\_wtime()-t##n  #define TIME\_TEXT(n) {char st[50]; \  while (!glfwWindowShouldClose(curr\_window))  {  TIME\_START(1);  // draw function  if(choose\_scene == 1)  draw1(mat\_model, mat\_view);  if (choose\_scene == 2)  draw2(mat\_model, mat\_view);  TIME\_START(2);  // swap buffers and poll events  glfwSwapBuffers(curr\_window);  TIME\_END(2);  TIME\_START(3);  glfwPollEvents();  TIME\_END(3);  }  // no more events will be delivered for that window and its handle becomes invalid  // glfwDestroyWindow(curr\_window);  // destroys all remaining windows, frees any allocated resources and into an uninitialized  // glfwTerminate();  } |