南京工程学院

**实 验 报 告**

课程名称 虚拟现实2020

实验项目名称 光照与材质

实验学生班级 数嵌172

实验学生姓名 朱广锋

学　　　　号 202170638

同组学生姓名 无

实验时间 2020/5/16

实验地点

1. 实验主题

利用VC集成开发环境，实现利用OpenGL编写绘制具有真实感效果的图形方法，包括如何添加光源、设置颜色、材质属性等方法。

1. 实验准备

1.打开Visual Studio并且设置好工作目录；

2.下载OpenGL安装包所需文件（http://d.download.csdn.net/down/2560229/ssagnn23），主要包括GL.H GLAUX.H GLU.H glut.h

GLAUX.LIB GLU32.LIB glut32.lib glut.lib OPENGL32.LIB

glaux.dll glu32.dll glut32.dll glut.dll opengl32.dll

3.复制并配置OpenGL库函数到指定的目录（.h、.lib、.dll分别放到MSVC include、lib和系统Path路径如System32），检查复制后是否文件已经存在于指定目录下。

 在Vistudio Studio中建立一个空类型的项目，项目名为Excer3\_LM。其下有四个子任务：

---Excer3\_LM

--Excer3\_triangle.cpp

--Excer3\_depth.cpp

--Excer3\_Movelight.cpp

--Excer3\_material.cpp

1. 主要数据源、库函数、变量、涉及函数及其解释

|  |
| --- |
| // Excer3\_triangle.cpp  #include <GL/glut.h>  #include <stdlib.h>  void init(void)  {  glClearColor(0.0, 0.0, 0.0, 0.0);  glShadeModel(GL\_SMOOTH);  }  void triangle(void)  {  glBegin(GL\_TRIANGLES);  glColor3f(1.0, 0.0, 0.0);  glVertex2f(5.0, 5.0);  glColor3f(0.0, 1.0, 0.0);  glVertex2f(25.0, 5.0);  glColor3f(0.0, 0.0, 1.0);  glVertex2f(5.0, 25.0);  glEnd();  }  void display(void)  {  glClear(GL\_COLOR\_BUFFER\_BIT);  triangle();  glFlush();  }  void reshape(int w, int h)  {  glViewport(0, 0, (GLsizei)w, (GLsizei)h);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  if (w <= h)  gluOrtho2D(0.0, 30.0, 0.0, 30.0 \* (GLfloat)h / (GLfloat)w);  else  gluOrtho2D(0.0, 30.0 \* (GLfloat)w / (GLfloat)h, 0.0, 30.0);  glMatrixMode(GL\_MODELVIEW);  }  void keyboard(unsigned char key, int x, int y)  {  switch (key) {  case 27:  exit(0);  break;  }  }  int main(int argc, char\*\* argv)  {  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);  glutInitWindowSize(500, 500);  glutInitWindowPosition(100, 100);  glutCreateWindow(argv[0]);  init();  glutDisplayFunc(display);  glutReshapeFunc(reshape);  glutKeyboardFunc(keyboard);  glutMainLoop();  return 0;  } |
| 添加代码段含义：  设置光滑着色模式，依次设置三个顶点颜色和位置，绘制三角形。  函数解释：  glShadeModel 设置着色模式，GL\_SMOOTH为平滑，对两点之间的颜色进行插值；  参数默认值GL\_FLAT使用顶点颜色来对顶点之间的区域、线段进行着色。  glBegin (GL\_TRIANGLES) 开始绘制三角形，每三个点作为一个独立的三角形；  glEnd结束绘制；  glColor3f设定绘制颜色；  glVertex2f设置顶点坐标。 |
| // Excer3\_ depth.cpp  #include <GL/glut.h>  #include <stdlib.h>  void display(void)  {  glClearColor(0.0f, 0.0f, 0.0f, 0.0f);  glClear(GL\_COLOR\_BUFFER\_BIT);  glClearDepth(1.0);  glClear (GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  //绘制一个红色三角形，z轴位置为-1.0f  glColor3f(1.0, 0.0, 0.0);  glBegin(GL\_POLYGON);  glVertex3f(-0.5f, -0.3f, -1.0f);  glVertex3f(0.5f, -0.3f, -1.0f);  glVertex3f(0.0f, 0.4f, -1.0f);  glEnd();  //绘制一个蓝色的倒立三角形，z轴位置为-2.0f  glColor3f(0.0, 0.0, 1.0);  glBegin(GL\_POLYGON);  glVertex3f(-0.5f, 0.3f, -2.0f);  glVertex3f(0.0f, -0.4f, -2.0f);  glVertex3f(0.5f, 0.3f, -2.0f);  glEnd();  glFlush();  }  void reshape(int w, int h)  {  glViewport(0, 0, (GLsizei)w, (GLsizei)h);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  glOrtho(-1.0, 1.0, -1.0, 1.0, 0.0, 10.0);  //glOrtho(-3.0, 3.0, -3.0, 3.0, 0.0, 10.0);  }  int main(int argc, char\*\* argv)  {  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB | GLUT\_DEPTH);  glutInitWindowSize(400, 400);  glutInitWindowPosition(100, 100);  glutCreateWindow(argv[0]);  glEnable(GL\_DEPTH\_TEST) ;//启用深度测试  glutDisplayFunc(display);  glutReshapeFunc(reshape);  glutMainLoop();  return 0;  } |
| 添加代码段含义：  启用深度测试后，绘制两个z轴位置不同的三角形  函数解释：  glClearDepth(1.0); 清除深度缓存时使用的深度值；  glClear (GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);清除缓存区，GL\_COLOR\_BUFFER\_BIT颜色缓冲，GL\_DEPTH\_BUFFER\_BIT深度缓冲；  glEnable(GL\_DEPTH\_TEST)启用深度测试。 |
| // Excer3\_MoveLight.cpp  #include <GL/glut.h>  #include <stdlib.h>  static int spin = 0;  /\* Initialize material property, light source, lighting model,  \* and depth buffer.  \*/  void init(void)  {  glClearColor(0.0, 0.0, 0.0, 0.0);  glShadeModel(GL\_SMOOTH);  glEnable(GL\_LIGHTING);  glEnable(GL\_LIGHT0);  glEnable(GL\_DEPTH\_TEST);  }  /\* Here is where the light position is reset after the modeling  \* transformation (glRotated) is called. This places the  \* light at a new position in world coordinates. The cube  \* represents the position of the light.  \*/  void display(void)  {  GLfloat position[] = { 0.0, 0.0, 1.5, 1.0 };  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  glPushMatrix();  gluLookAt(0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);  glPushMatrix();  glRotated((GLdouble)spin, 1.0, 0.0, 0.0);  glLightfv(GL\_LIGHT0, GL\_POSITION, position);  glTranslated(0.0, 0.0, 1.5);  glDisable(GL\_LIGHTING);  glColor3f(0.0, 1.0, 1.0);  glutWireCube(0.1);  glEnable(GL\_LIGHTING);  glPopMatrix();  glutSolidTorus(0.275, 0.85, 8, 15);  glPopMatrix();  glFlush();  }  void reshape(int w, int h)  {  glViewport(0, 0, (GLsizei)w, (GLsizei)h);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluPerspective(40.0, (GLfloat)w / (GLfloat)h, 1.0, 20.0);  glMatrixMode(GL\_MODELVIEW);  glLoadIdentity();  }  void mouse(int button, int state, int x, int y)  {  switch (button) {  case GLUT\_LEFT\_BUTTON:  if (state == GLUT\_DOWN) {  spin = (spin + 30) % 360;  glutPostRedisplay();  }  break;  default:  break;  }  }  void keyboard(unsigned char key, int x, int y)  {  switch (key) {  case 27:  exit(0);  break;  }  }  int main(int argc, char\*\* argv)  {  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB | GLUT\_DEPTH);  glutInitWindowSize(500, 500);  glutInitWindowPosition(100, 100);  glutCreateWindow(argv[0]);  init();  glutDisplayFunc(display);  glutReshapeFunc(reshape);  glutMouseFunc(mouse);  glutKeyboardFunc(keyboard);  glutMainLoop();  return 0;  } |
| 添加代码段含义：  开启光照并设置使用光源  函数解释：  glEnable(GL\_LIGHTING);开启光照  glEnable(GL\_LIGHT0);使用0号光源  glRotated ((GLdouble) spin, 1.0, 0.0, 0.0);让0号光源绕x轴旋转  glTranslated (0.0, 0.0, 1.5);沿z轴平移0号光源 |
| // Excer3\_ material.cpp  #include <GL/glut.h>  #include <stdlib.h>  void display(void) {  GLfloat no\_mat[] = { 0.0, 0.0, 0.0, 1.0 };  GLfloat mat\_ambient[] = { 0.7, 0.7, 0.7, 1.0 };  GLfloat mat\_ambient\_color[] = { 0.8, 0.8, 0.2, 1.0 };  GLfloat mat\_diffuse[] = { 0.1, 0.5, 0.8, 1.0 };  GLfloat mat\_specular[] = { 1.0, 1.0, 1.0, 1.0 };  GLfloat no\_shininess[] = { 0.0 };  GLfloat low\_shininess[] = { 5.0 };  GLfloat high\_shininess[] = { 100.0 };  GLfloat mat\_emission[] = { 0.3, 0.2, 0.2, 0.0 };  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);    glPushMatrix();  glTranslatef(-3.75, 3.0, 0.0);  glMaterialfv(GL\_FRONT, GL\_AMBIENT, no\_mat);  glMaterialfv(GL\_FRONT, GL\_DIFFUSE, mat\_diffuse);  glMaterialfv(GL\_FRONT, GL\_SPECULAR, no\_mat);  glMaterialfv(GL\_FRONT, GL\_SHININESS, no\_shininess);  glMaterialfv(GL\_FRONT, GL\_EMISSION, no\_mat);  glutSolidSphere(1.0, 16, 16);  glPopMatrix();    glPushMatrix();  glTranslatef(-1.25, 3.0, 0.0);  glMaterialfv(GL\_FRONT, GL\_AMBIENT, no\_mat);  glMaterialfv(GL\_FRONT, GL\_DIFFUSE, mat\_diffuse);  glMaterialfv(GL\_FRONT, GL\_SPECULAR, mat\_specular);  glMaterialfv(GL\_FRONT, GL\_SHININESS, low\_shininess);  glMaterialfv(GL\_FRONT, GL\_EMISSION, no\_mat);  glutSolidSphere(1.0, 16, 16);  glPopMatrix();  glPushMatrix();  glTranslatef(1.25, 3.0, 0.0);  glMaterialfv(GL\_FRONT, GL\_AMBIENT, no\_mat);  glMaterialfv(GL\_FRONT, GL\_DIFFUSE, mat\_diffuse);  glMaterialfv(GL\_FRONT, GL\_SPECULAR, mat\_specular);  glMaterialfv(GL\_FRONT, GL\_SHININESS, high\_shininess);  glMaterialfv(GL\_FRONT, GL\_EMISSION, no\_mat);  glutSolidSphere(1.0, 16, 16);  glPopMatrix();    glPushMatrix();  glTranslatef(3.75, 3.0, 0.0);  glMaterialfv(GL\_FRONT, GL\_AMBIENT, no\_mat);  glMaterialfv(GL\_FRONT, GL\_DIFFUSE, mat\_diffuse);  glMaterialfv(GL\_FRONT, GL\_SPECULAR, no\_mat);  glMaterialfv(GL\_FRONT, GL\_SHININESS, no\_shininess);  glMaterialfv(GL\_FRONT, GL\_EMISSION, mat\_emission);  glutSolidSphere(1.0, 16, 16);  glPopMatrix();  glFlush();  }  void init(void)  {  GLfloat ambient[] = { 1.0, 1.0, 1.0, 0.0 };  GLfloat diffuse[] = { 1.0, 1.0, 1.0, 0.0 };  GLfloat light0\_position[] = { 0.0, 1.0, 1.0,0.0 };  glClearColor(0.0, 0.0, 0.0, 0.0);  glShadeModel(GL\_SMOOTH);  glLightfv(GL\_LIGHT0, GL\_POSITION, light0\_position);  glLightfv(GL\_LIGHT0, GL\_DIFFUSE, diffuse);  glColorMaterial(GL\_FRONT, GL\_DIFFUSE);  glEnable(GL\_LIGHTING);  glEnable(GL\_LIGHT0);  glEnable(GL\_DEPTH\_TEST);  glEnable(GL\_COLOR\_MATERIAL);  }  void reshape(int w, int h)  {  glViewport(0, 0, (GLsizei)w, (GLsizei)h);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  if (w <= h)  glOrtho(-5, 5, -5 \* (GLfloat)h / (GLfloat)w, 5 \* (GLfloat)h / (GLfloat)w, -10.0, 10.0);  else  glOrtho(-5 \* (GLfloat)w / (GLfloat)h, 5 \* (GLfloat)w / (GLfloat)h, -5, 5, -10.0, 10.0);  glMatrixMode(GL\_MODELVIEW);  glLoadIdentity();  }  int main(int argc, char\*\* argv)  {  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB | GLUT\_DEPTH);  glutInitWindowSize(500, 500);  glutInitWindowPosition(100, 100);  glutCreateWindow(argv[0]);  init();  glutDisplayFunc(display);  glutReshapeFunc(reshape);  glutMainLoop();  return 0;  } |
| 函数解释：  glTranslatef (-3.75, 3.0, 0.0)：沿 x 轴负方向平移 3.75，沿 y 轴正方向平 移3；  glTranslatef (-1.25, 3.0, 0.0)：沿x轴负方向平移1.25，沿 y轴正方向平移3； glTranslatef (1.25, 3.0, 0.0)：沿 x 轴正方向平移 1.25，沿 y 轴正方向平移 3。 |

1. 实验任务

任务1：使用光滑着色模式绘制三角形的应用源程序，源程序名Excer3\_triangle.cpp

注意：主要应用的函数为：

glBegin glColor3f () glVertex2f () glEnd() glShadeModel()

 在Excer3\_LM中添加C++ File(.cpp)文件，文件名为Excer3\_triangle.cpp

 在Excer3\_triangle.cpp中添加代码，实现光滑着色模式绘制一个三角形的功能。

任务2：建立一个绘制两个三角形，查看启用和不启用深度缓存的效果，源程序名为Excer3\_ depth.cpp

主要应用的函数为：

glEnable(GL\_DEPTH\_TEST) glDisable () glRectf() glPolygonStipple()

 在Excer3\_LM中添加C++ File(.cpp)文件，文件名为Excer3\_ depth.cpp

 在Excer3\_ depth.cpp中添加代码，实现绘制两个三角形，并启用和不启用深度缓存，分析结果功能。

任务3：建立一个模型变换来移动光源的应用源程序，源程序名为Excer3\_MoveLight.cpp

主要应用的函数为：

glutWireCube() glRotated () glPushMatrix() glPopMatrix() glLightfv( )

 在Excer3\_LM中添加C++ File(.cpp)文件，文件名为Excer3\_MoveLight.cpp

 在Excer3\_MoveLight.cpp中添加代码，实现绘制一个光源，并能实行视角变换移动光源。

 光源位置的初始位置设定后，在经过调用模型变换（glRotated）函数后，光源的位置会随着模型变换重新设置，将变换后的光源位置放置在世界坐标系内，并将Cube的位置代表了光源的位置

任务4：建立一个设置不同材质属性的应用源程序，源程序名为Excer3\_ material.cpp

主要应用的函数为：

glPushMatrix(); glPopMatrix(); glutPostRedisplay(); glRotatef(); glTranslatef(); glutWireSphere()

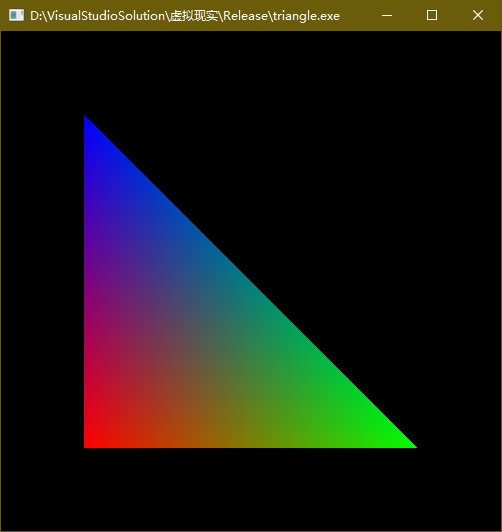
 在Excer3\_LM中添加C++ File(.cpp)文件，文件名为Excer3\_ material.cpp

 在Excer3\_ material中添加代码。

1. 主要分析和解释

运行结果：

任务1 如图为使用光滑着色模式绘制三角形的结果；



任务2 如图为启用和不启用深度缓存的效果；

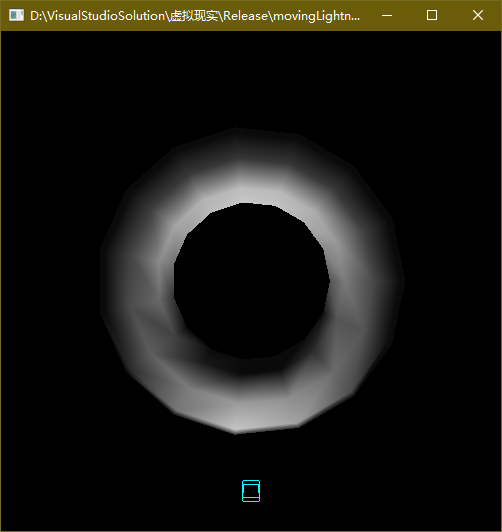
|  |  |
| --- | --- |
| 启用深度测试 | 不启用深度测试 |

深度测试会更新帧缓存中对应深度缓冲区每个位置的最小z值的颜色。

绘制两个长边形需要修改原文中的代码及结果如下：

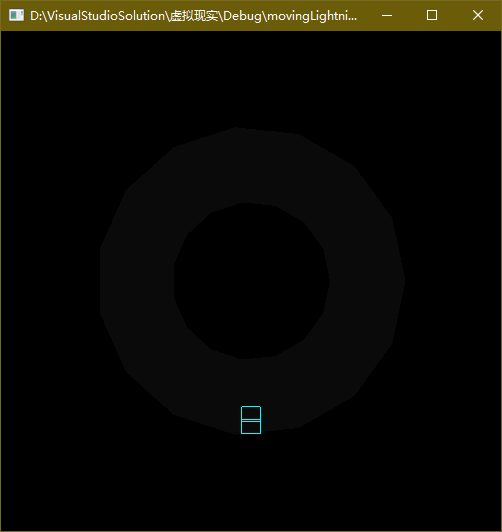
|  |
| --- |
| glColor3f(1.0, 0.0, 0.0);  glBegin(GL\_QUADS);  glVertex3f(-2.0f, 1.0f, 0.0f);  glVertex3f(2.0f, 1.0f, 0.0f);  glVertex3f(2.0f, -1.0f, 0.0f);  glVertex3f(-2.0f, -1.0f, 0.0f);  glEnd();  // Blue  glColor3f(0.0, 0.0, 1.0);  glBegin(GL\_QUADS);  glVertex3f(-1.0f, 2.0f, 0.0f);  glVertex3f(1.0f, 2.0f, 0.0f);  glVertex3f(1.0f, -2.0f, 0.0f);  glVertex3f(-1.0f, -2.0f, 0.0f);  glEnd();  glOrtho(-3.0, 3.0, -3.0, 3.0, 0.0, 10.0);<reshape内>  红色矩形显示在蓝色矩形上方。 |

任务3 如图为移动光源效果。

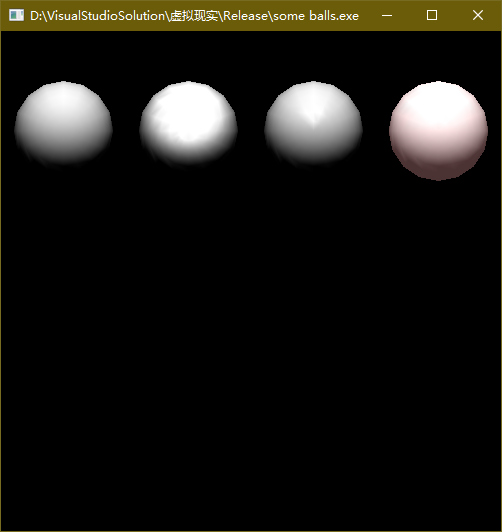


glEnable(Lignting)启动光源

关闭光源如下图



任务4 如图为设置不同材质属性的效果



glMatrixPop()和glMatrixPush()函数对堆栈矩阵进行压栈和出栈，实现了绘制设备上下文的恢复，避免两次球体绘制之间的影响。

范例3.6的八个球体效果如下实实现

|  |
| --- |
| glEnable(GL\_COLOR\_MATERIAL)  glColorMaterial(GL\_FRONT, GL\_DIFFUSE)  设置材质球颜色 |

1. 展望

本次实验，通过光滑着色、深度测试、光源、材质、颜色，联系绘制具有真实感效果的图形方法，让我熟悉了光源的添加和设置方法，颜色和材质属性的设置方法，着色模式、深度测试、光照、颜色、材质等的设置是游戏、建模等需要绘制真实感物体应用的基础。

教师评阅：

|  |  |
| --- | --- |
| 评阅项目及内容 | 得分 |
| 1．考勤（10分） |  |
| 2．实验完成情况（50分） |  |
| 3．报告撰写内容（40分） |  |
| 合 计 |  |
| 成绩评定 |  |