## 0809702001（虚拟现实）实验三 光照与材质

**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。实验报告和程序压缩代码请于上机一周内上传到腾讯课堂作业内（截止时间为：）。

**准备**：

* 打开您的VSStudio 并且设置好您的工作目录；
* 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
* 复制并配置OpenGL库函数到制定的目录，检查复制后是否文件已经存在于指定目录下
* **在**VSStudio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**

**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。**
* **部分代码如下，请于高亮黄色部分添加必要的源代码。**

#include <GL/glut.h>

#include <stdlib.h>

void init(void)

{

glClearColor (0.0, 0.0, 0.0, 0.0);

/\*添加代码，设置光滑找色模式\*/

|  |
| --- |
| glShadeModel (GL\_SMOOTH); |

}

/\*添加triangle(void)函数的代码，主要绘制一个三角形，为三个顶点设置颜色

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;

}

* 请解释添加代码段的含义？

|  |
| --- |
| glShadeModel (GL\_SMOOTH);设定对两点之间的颜色进行插值；  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。 |

* 请解释Task1中所用函数的使用规则和参数设置的含义？

|  |
| --- |
| glShadeModel设定着色模式，参数GL\_SMOOTH 对顶点之间线段或区域的颜色进行插值，GL\_FLAT使用顶点颜色来对顶点之间的区域、线段进行着色。  glBegin开始绘制图形，GL\_TRIANGLES将以下每三个顶点作为一个三角形。 |

* 运行程序后，利用图，表，分析的形式记录所有结果，并基于它们做一个简单的解释。

|  |
| --- |
|  |

**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。**
* **部分代码如下，请于高亮黄色部分补齐必要的源代码。**

#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);

}

//main()函数中添加代码，是否启用深度测试

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;

}

* 请解释添加代码段的含义？

|  |
| --- |
| 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。 |

* glutInitDisplayMode (GLUT\_SINGLE | GLUT\_RGB| GLUT\_DEPTH);此函数中GLUT\_DEPTH参数的含义是什么？如果不设置此函数，有什么结果？

|  |
| --- |
| GLUT\_DEPTH指定显示模式使用深度缓冲区  不设置此函数，无法使用深度测试等。 |

* 完整运行**Excer3\_ depth.cpp**你可以得到什么样的结果，辅助以流程图加以描述？

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

* 请启用深度缓存测试，获得什么样的结果？请不启用深度缓存测试，获得什么样的结果？请总结这样的结果得到怎么样的启示？

|  |  |
| --- | --- |
| 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。 | |

* 如果是绘制两个长边形呢（例如一个长为2，宽4，另一个长为4，宽为2）），请修改原文中的代码，并利用图和表形式记录所有结果，并基于改变得到相关结果做一个简单的解释。

|  |
| --- |
| 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：**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**
* **部分代码如下，请于高亮黄色部分补齐必要的源代码。**

#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);

/\*init()函数中添加代码，是否启用光照计算，并启用Light0

|  |
| --- |
| glEnable(GL\_LIGHTING); |

/\*init()函数中添加代码，是否启用光照计算

|  |
| --- |
| 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);

/\*display()函数中添加代码，利用 glTranslated重新设置光源位置，调用glRotated函数随着视点一起变换spin度

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 ();

/\*display()函数中添加代码，利用 glTranslated重新设置光源位置，调用glRotated函数随着视点一起变换

}

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;

}

* 请解释添加代码段的含义，为什么如此添加？

|  |
| --- |
| 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 |

* 请解释Task3中glMatrixPop()和glMatrixPush()函数使用的作用是什么？

|  |
| --- |
| glPushMatrix复制当前矩阵到矩阵堆栈栈顶  glPopMatrix将矩阵堆栈栈顶矩阵读取到当前矩阵并弹出（删除） |

* GlEnable(Lignting) 函数的作用是什么？如果不启用会是什么样的效果？

|  |
| --- |
| glEnable(Lignting)启动光源  关闭光源如下图 |

* 补充程序后完整运行**Excer3\_MoveLight.cpp**你可以得到什么样的结果

|  |
| --- |
|  |

* 试比较电子教材的范例3.4 和范例3.5，如果要实现范例3.5的随着视点一起移动的光源，且视点沿着y轴转动，代码需要做哪些改动，请把两个范例的不同的地方此处进行比较。

|  |
| --- |
| 3.4 是视点不变，光源绕 x 轴转动； 3.5 是光源随着视点移动。不同之处在于需要在指定视点之前，指定光源位置，此后动态调整视点时，光源相对位置不变。 |

**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，。**
* **部分代码如下，请于高亮黄色部分补齐必要的源代码。**

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);

/\* draw sphere in first row, first column

\* diffuse reflection only; no ambient or specular

\*/

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();

/\* draw sphere in first row, second column

\* diffuse and specular reflection; low shininess; no ambient

\*/

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();

/\* draw sphere in first row, third column

\* diffuse and specular reflection; high shininess; no ambient

\*/

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();

/\* draw sphere in first row, fourth column

\* diffuse reflection; emission; no ambient or specular reflection

\*/

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; |

}

* 请解释三个glTranslatef的含义分别是什么，为什么如此添加？

|  |
| --- |
| 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。  设置球体位置，防止重合或遮挡。 |

* 请解释Task4中glMatrixPop()和glMatrixPush()函数使用的作用是什么？

|  |
| --- |
| 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。 |

* 为什么仅仅有display()函数绘制不出球体，请你添加出黄色加亮体中的main（）函数，Reshape()函数和Init()函数？

|  |
| --- |
| 见代码 |

* 补充程序后完整运行**Excer3\_material.cpp**你可以得到什么样的结果？

|  |
| --- |
|  |

如果想得到范例3.6的八个球体的效果，应该如何实现？

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

**数嵌172 朱广锋 202170638**