

# 《图形渲染技术》实验指导书

王 丽

电子与信息工程系计算机应用教研室  
**2010-11-18**

《图形渲染技术》是我院计算机科学与技术专业本科生的一门专业必修课，其目的是通过本课程的学习，使学生掌握图形渲染工具 OpenGL 语言的使用，同时为后继的高级课程，如虚拟现实，3D 游戏开发提供必要的理论和工程基础。

本课程总学时为 64 学时，其中理论课程学时为 40 学时，实验学时为 24 学时，实验内容安排如下：

1. OpenGL 中的二维编程（4 学时）
2. 交互与动画的应用（4 学时）
3. 三维编程基础（4 学时）
4. 几何变换的应用（2 学时）
5. 光照与材质的应用（4 学时）
6. 纹理的应用（4 学时）
7. 绘制 Bezier 曲线（2 学时）

#### **实验总的形式与要求：**

上机实验应一人一组，独立上机。上机前应事先做好准备，以提高上机实验的效率，要求如下：

1. 了解所用的计算机系统（包括 VC6.0++编译系统和工作平台）的性能和使用方法；
2. 复习和掌握与本实验有关的教学内容；
3. 准备好上机所需的程序。手编程序应书写整齐，并经人工检查无误后才能上机，以提高上机效率；
4. 对程序运行过程中可能出现的问题事先做出估计，对程序中自己有疑问的地方，应作出记号，以便上机时给予注意；
5. 每次实验前详细阅读实验指导书，熟悉实验目的和实验内容，实验后整理并提交实验报告。

## 实验一 OpenGL 中的二维编程（4 学时）

### 一、实验目的

- 1、了解 VC++6.0 开发环境；
- 2、使学生掌握 OpenGL 中二维编程及给图形着色；

### 二、实验要求

- 1、了解并掌握 VC++6.0 的使用和 OpenGL 库的添加；
- 2、掌握 OpenGL 中二维编程及给图形着色。

### 三、实验环境

- 1、windows 操作系统；
- 2、OpenGL+VC++6.0。

### 四、实验内容

- 1、在 Windows 环境中创建 OpenGL 窗口；
- 2、在 OpenGL 窗口中画出一个三角形和一个矩形，并给它们着色（如图 1-1、1-2）。



图 1-1

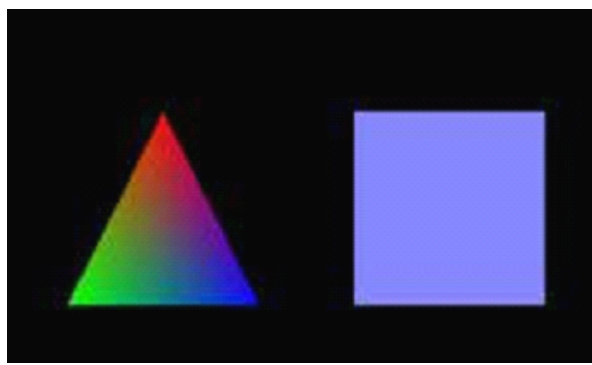


图 1-2

### 五、实验步骤

1. 启动 VC++6.0 软件；
2. 创建一个新的 Win32 程序，链接 OpenGL 库文件，在 Project-> Settings,然后单击 LINK 标签。在“Object/LibraryModules”选项中的开始处（在 kernel32.lib 前）增加 OpenGL32.lib, GLu32.lib 和 GLaux.lib 后，单击 OK 按钮；
3. 开始编写程序。
  - 一个完整的 OpenGL 窗口程序模板，后面的实验场景可以在此程序基础上，根据要求添加修改程序。

```
#include <windows.h>           // Header File For Windows
#include <gl\gl.h>              // Header File For The OpenGL32 Library
#include <gl\glu.h>             // Header File For The GLu32 Library
#include <gl\glaux.h>           // Header File For The Glaux Library
```

```

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM); //声明 WndProc
GLvoid ReSizeGLScene(GLsizei width, GLsizei height) // 设置 GL 窗口并初始化
int InitGL(GLvoid) // 所有的设置都从这里开始
int DrawGLScene(GLvoid) // 所有的绘制都在这个函数里实现
GLvoid KillGLWindow(GLvoid) // 销毁窗口
//创建 OpenGL 窗口，包括窗口标题，长，宽，位及是否全屏
BOOL CreateGLWindow(char* title, int width, int height, int bits, bool fullscreenflag)

```

- 画出一个三角形和一个矩形，只需要在之前的代码中修改即可

```

int DrawGLScene(GLvoid) // 此过程中包括所有的绘制代码
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // 清除屏幕及深度缓存
    glLoadIdentity(); // 重置当前的模型观察矩阵
    glTranslatef(-1.5f,0.0f,-6.0f); // 左移 1.5 单位，并移入屏幕 6.0
    glBegin(GL_TRIANGLES); // 绘制三角形
    glVertex3f( 0.0f, 1.0f, 0.0f); // 上顶点
    glVertex3f(-1.0f,-1.0f, 0.0f); // 左下
    glVertex3f( 1.0f,-1.0f, 0.0f); // 右下
    glEnd(); // 三角形绘制结束
    glTranslatef(3.0f,0.0f,0.0f); // 右移 3 单位
    glBegin(GL_QUADS); // 绘制正方形
    glVertex3f(-1.0f, 1.0f, 0.0f); // 左上
    glVertex3f( 1.0f, 1.0f, 0.0f); // 右上
    glVertex3f( 1.0f,-1.0f, 0.0f); // 左下
    glVertex3f(-1.0f,-1.0f, 0.0f); // 右下
    glEnd(); // 正方形绘制结束
    return TRUE; // 继续运行
}
//换掉窗口模式下的标题内容
if (keys[VK_F1]) // F1 键按下了么?
{
    keys[VK_F1]=FALSE; // 若是，使对应的 Key 数组
    中的值为 FALSE
    KillGLWindow(); // 销毁当前的窗口
    fullscreen=!fullscreen; // 切换 全屏 / 窗口 模式
    // 重建 OpenGL 窗口(修改)
}

```

```

        if (!CreateGLWindow("第一个多边形程序",640,480,16,fullscreen))
        {
            return 0; // 如果窗口未能创建，程序退出
        }
    }

    ● 添加颜色

    int DrawGLScene(GLvoid) // 此过程中包括所有的绘制代码
    {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // 清除屏幕及深度缓存

        glLoadIdentity(); // 重置模型观察矩阵
        glTranslatef(-1.5f,0.0f,-6.0f); // 左移 1.5 单位，并移入屏幕 6.0
        glBegin(GL_TRIANGLES); // 绘制三角形
        glColor3f(1.0f,0.0f,0.0f); // 设置当前色为红色
        glVertex3f( 0.0f, 1.0f, 0.0f); // 上顶点
        glColor3f(0.0f,1.0f,0.0f); // 设置当前色为绿色
        glVertex3f(-1.0f,-1.0f, 0.0f); // 左下
        glColor3f(0.0f,0.0f,1.0f); // 设置当前色为蓝色
        glVertex3f( 1.0f,-1.0f, 0.0f); // 右下
        glEnd(); // 三角形绘制结束
        glTranslatef(3.0f,0.0f,0.0f); // 右移 3 单位
        glColor3f(0.5f,0.5f,1.0f); // 一次性将当前色设置为蓝色
        glBegin(GL_QUADS); // 绘制正方形
        glVertex3f(-1.0f, 1.0f, 0.0f); // 左上
        glVertex3f( 1.0f, 1.0f, 0.0f); // 右上
        glVertex3f( 1.0f,-1.0f, 0.0f); // 右下
        glVertex3f(-1.0f,-1.0f, 0.0f); // 左下
        glEnd(); // 正方形绘制结束
        return TRUE; // 继续运行
    }

    // 重建 OpenGL 窗口
    if (!CreateGLWindow("颜色实例",640,480,16,fullscreen))

```

【思考题】改改代码中的红绿蓝分量值，看看最后 y 有什么样的结果。

## 实验二 交互与动画的应用（4 学时）

## 一、实验目的

- 1、掌握键盘回调函数；
- 2、掌握定时器回调函数。

## 二、实验要求

在给出的定时器回调函数例子基础上，修改程序，达到以键盘控制矩形移动的目的。

## 三、实验环境

- 1、windows 操作系统；
- 2、OpenGL+VC++6.0。

## 四、实验内容

通过键盘的按下，移动矩形。

## 五、实验步骤

1. 启动 VC++6.0 软件；
2. 创建一个新的 Win32 程序，链接 OpenGL 库文件，在 Project-> Settings,然后单击 LINK 标签。

在“Object/LibraryModules”选项中的开始处（在 kernel32.lib 前）增加 OpenGL32.lib, GLu32.lib 和 GLaux.lib 后，单击 OK 按钮；

3. 开始编写程序。

- 给出一个自动移动的矩形例子，在窗口内绘制一个移动的矩形，其中需要利用 GLUT 库中的函数：

glutTimerFunc(unsigned int msec, (\*func) (int value), int value);

具体程序如下：

```
#include <windows.h>
#include <gl/glut.h>
#include <gl/gl.h>
#include <gl/glu.h>
```

```
// 参数指定正方形的位置和大小
```

```
GLfloat x1 = 100.0f;
GLfloat y1 = 150.0f;
GLsizei rsize = 50;
```

```
// 正方形运动变化的步长
```

```
GLfloat xstep = 1.0f;
GLfloat ystep = 1.0f;
```

```
// 窗口的大小
```

```
GLfloat windowWidth;
GLfloat windowHeight;
```

```
void RenderScene(void)
{
```

```
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0f, 0.0f, 0.0f);
    glRectf(x1, y1, x1+rsize, y1+rsize);

    //清空命令缓冲区并交换帧缓存
    glutSwapBuffers();
}

void ChangeSize(GLsizei w, GLsizei h)
{
    if(h == 0)    h = 1;

    glViewport(0, 0, w, h);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    if (w <= h)
    {
        windowHeight = 250.0f*h/w;
        windowWidth = 250.0f;
    }
    else
    {
        windowWidth = 250.0f*w/h;
        windowHeight = 250.0f;
    }
    glOrtho(0.0f, windowWidth, 0.0f, windowHeight, 1.0f, -1.0f);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void TimerFunction(int value)
{
    // 处理到达窗口边界的正方形，使之反弹
    if(x1 > windowWidth-rsize || x1 < 0)    xstep = -xstep;
    if(y1 > windowHeight-rsize || y1 < 0)    ystep = -ystep;
    if(x1 > windowWidth-rsize)    x1 = windowWidth-rsize-1;
    if(y1 > windowHeight-rsize)    y1 = windowHeight-rsize-1;

    // 根据步长修改正方形的位置
    x1 += xstep;
    y1 += ystep;
```

```
// 用新坐标重新绘图
glutPostRedisplay();
glutTimerFunc(33, TimerFunction, 1);
}

void SetupRC(void)
{
    //设置窗口清除色为蓝色
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
}

int main(void)
{
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutCreateWindow("Bounce");
    glutDisplayFunc(RenderScene);
    glutReshapeFunc(ChangeSize);
    glutTimerFunc(33, TimerFunction, 1);

    SetupRC();
    glutMainLoop();
}
```

## 实验三 三维编程基础（4 学时）

### 一、实验目的

- 1、熟悉 OpenGL 三维绘图操作；
- 2、理解三维物体的平行投影方式和透视投影方式；

### 二、实验要求

- 1、了解 OpenGL 的投影变换操作。

### 三、实验环境

- 1、windows 操作系统；
- 2、OpenGL+VC++6.0。

### 四、实验内容

- 1、使用 Visual C++ 6.0 和 OpenGL 进行简单的三维绘图，并设置不同的投影方式。

### 五、实验步骤

- 1、启动 VC++6.0 软件；



2、创建一个新的 Win32 程序，链接 OpenGL 库文件，在 Project-> Settings,然后单击 LINK 标签。在 “Object/LibraryModules” 选项中的开始处（在 kernel32.lib 前）增加 OpenGL32.lib, GLu32.lib 和 GLaux.lib 后，单击 OK 按钮；

3、请将下面的程序写入源文件 Test.c；

```
#include <gl/glut.h>
static int angle = 45; //定义旋转角度
void reshape(GLsizei w, GLsizei h)
{
    // 视口变换
    glViewport(0,0,w,h); //使用glViewport来定义视口。其中前两个参数定义了视口的左下脚(0,0表示最左下方)
    //后两个参数分别是宽度和高度。
    // 投影变换
    glMatrixMode(GL_PROJECTION); //OpenGL支持两种类型的投影变换，即透视投影和正投影。
    //投影也是使用矩阵来实现的。如果需要操作投影矩阵，需要以
    //GL_PROJECTION为参数调用glMatrixMode函数。
    // glLoadIdentity(); //重置模型观察矩阵

    // glFrustum (-1.0, 1.0, -1.0, 1.0, 1.0, 10.0);
    //使用glFrustum函数可以将当前的可视空间设置为透视投影空间。
    //用它之前先要用glMatrixMode()说明当前矩阵方式是投影GL_PROJECTION。
    //也可以使用更常用的gluPerspective函数。
    //空间可以是“正投影”的（使用glOrtho或gluOrtho2D），
    //也可以是“透视投影”的（使用glFrustum或gluPerspective）。
    glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 6.0);
    //正射投影，同glFrustum，六个参数分别为：左右垂直切面坐标，底上水平切面坐标
    //近深度切面和远深度切面的距离，可为负值。
}

void myDisplay()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // 模型变换
    glMatrixMode(GL_MODELVIEW);
    glTranslatef(0.0f,0.0f,0.0f); // 物体在三个坐标上的位移值。
    glRotatef(angle,0.5f,0.5f,0.0f); // 物体绕(0,0,0)到(x,y,z)的直线以逆时针旋转，参数angle表示旋转的角度。
    glScalef(1.0, 1.0, 1.0); // 三个参数分别是X、Y、Z轴向的比例变换因子。

    //金字塔底边
    glBegin(GL_QUADS);
    glVertex3f(0.5f,-0.5f,0.5f);
```

```
glVertex3f(-0.5f,-0.5f,0.5f);
glVertex3f(0.5f,-0.5f,-0.5f);
glVertex3f(-0.5f,-0.5f,-0.5f);
glEnd();

glBegin(GL_TRIANGLES);
// 三角形前侧面
glColor3f(1.0f,0.0f,0.0f);
glVertex3f( 0.0f, 0.5f, 0.0f);
glColor3f(0.0f,0.5f,0.0f);
glVertex3f(-0.5f,-0.5f, 0.5f);
glColor3f(0.0f,0.0f,0.5f);
glVertex3f( 0.5f,-0.5f, 0.5f);

// 三角形右侧面
glColor3f(0.0f,1.0f,0.0f);
glVertex3f( 0.0f, 0.5f, 0.0f);
glColor3f(0.0f,0.0f,0.5f);
glVertex3f( 0.5f,-0.5f, 0.5f);
glColor3f(0.0f,0.5f,0.0f);
glVertex3f( 0.5f,-0.5f, -0.5f);

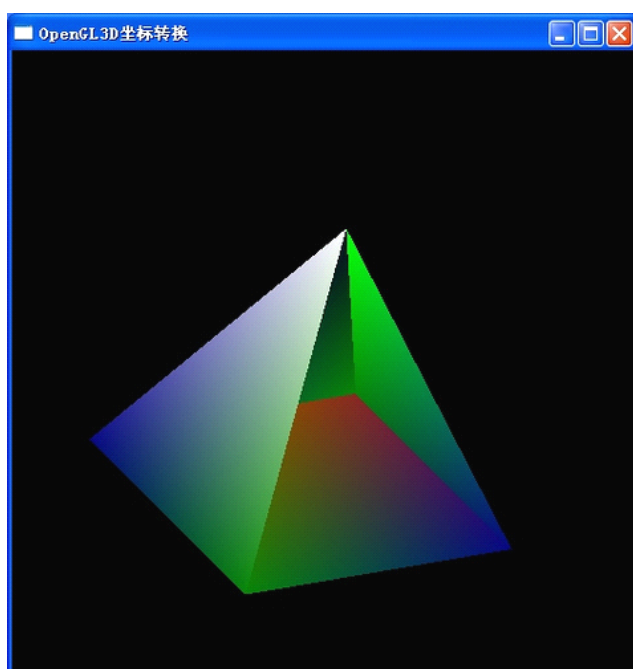
// 三角形后侧面
glColor3f(0.0f,0.0f,0.1f);
glVertex3f( 0.0f, 0.5f, 0.0f);
glColor3f(0.0f,0.5f,0.0f);
glVertex3f( 0.5f,-0.5f, -0.5f);
glColor3f(0.0f,0.0f,0.5f);
glVertex3f(-0.5f,-0.5f, -0.5f);

// 三角形左侧面
glColor3f(1.0f,1.0f,1.0f);
glVertex3f( 0.0f, 0.5f, 0.0f);
glColor3f(0.0f,0.0f,0.5f);
glVertex3f(-0.5f,-0.5f,-0.5f);
glColor3f(0.0f,0.5f,0.0f);
glVertex3f(-0.5f,-0.5f, 0.5f);
glEnd();

glFlush();
}

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

```
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
    glutInitWindowPosition(0,0);
    glutInitWindowSize(500,500);
    glutCreateWindow("OpenGL3D坐标转换");
    glutReshapeFunc(reshape);
    glutDisplayFunc(&myDisplay);
    glutMainLoop();
    return 0;
}
```



- 1、修改投影方式，观察不同的投影变换效果，适当修改绘图函数 myDisplay 中的内容，自由发挥并绘制三维图形；
- 2、完成实验报告并提交程序源代码。

## 实验四 几何变换的应用（2 学时）

### 一、实验目的

通过几何体的位置变换，熟练掌握几何图形的平移、旋转和缩放等变换操作。

### 二、实验要求

熟练掌握各种变换方法函数。

### 三、实验环境

- 1、windows 操作系统；

2、OpenGL+VC++6.0。

#### 四、实验内容

一个三角形进行的平移、旋转和缩放等变换，其输出如图 4-1 所示。

#### 五、实验步骤

1. 启动 VC++6.0 软件；

2. 创建一个新的 Win32 程序，链接 OpenGL 库文件，在 Project-> Settings,然后单击 LINK 标签。

在“Object/LibraryModules”选项中的开始处（在 kernel32.lib 前）增加 OpenGL32.lib, GLu32.lib 和 GLaux.lib 后，单击 OK 按钮；

3. 开始编写程序。

##### ● 几何变换

```
#include <windows.h>
```

```
#include <gl/gl.h>
```

```
#include <gl/glut.h>
```

```
void SetupRC(void)
```

```
{
```

```
    // 设置窗口背景颜色为白色
```

```
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
```

```
}
```

```
//绘制三角形
```

```
void DrawTriangle(void)
```

```
{
```

```
    glBegin(GL_TRIANGLES);
```

```
        glVertex2f(0.0f, 0.0f);
```

```
        glVertex2f(40.0f, 0.0f);
```

```
        glVertex2f(20.0f, 40.0f);
```

```
    glEnd();
```

```
}
```

```
void ChangeSize(int w, int h)
```

```
{
```

```
    if(h == 0)h = 1;
```

```
    glViewport(0, 0, w, h);
```

```
    glMatrixMode(GL_PROJECTION);
```

```
    glLoadIdentity();
```

```
    if (w <= h)
```

```
        glOrtho (-100.0f, 100.0f, -100.0f*h/w, 100.0f*h/w, -100.0f, 100.0f);
```

```
    else
```

```
glOrtho (-100.0f*w/h, 100.0f*w/h, -100.0f, 100.0f, -100.0f, 100.0f);

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
}

void RenderScene(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    // 设置当前操作矩阵为模型视图矩阵，并复位为单位矩阵
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    //绘制黑色的坐标轴
    glColor3f(0.0f, 0.0f, 0.0f);
    glBegin(GL_LINES);
        glVertex2f(-100.0f, 0.0f);
        glVertex2f(100.0f, 0.0f);
        glVertex2f(0.0f, -100.0f);
        glVertex2f(0.0f, 100.0f);
    glEnd();

    //绘制出第一个红色的三角形
    glColor3f(1.0f, 0.0f, 0.0f);
    DrawTriangle();

    //绘制出逆时针旋转 200 度角的绿色三角形
    glRotatef(200.0f,0.0f,0.0f,1.0f);
    glColor3f(0.0f, 1.0f, 0.0f);
    DrawTriangle();

    //绘制出沿 x 轴负方向平移 40 单位的黄色三角形
    glLoadIdentity();
    glTranslatef(-60.0f,0.0f,0.0f);
    glColor3f(1.0f, 1.0f, 0.0f);
    DrawTriangle();

    //绘制出缩放因子为 1.0, 2.0, 1.0 的蓝色三角形
    glTranslatef(100.0f, 10.0f, 0.0f);
    glScalef(1.0f, 2.0f, 1.0f);
    glColor3f(0.0f, 0.0f, 1.0f);
    DrawTriangle();
}
```

```

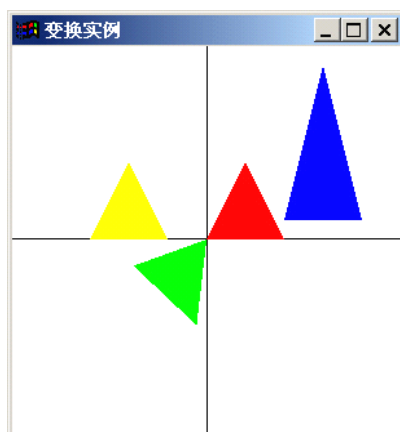
        glutSwapBuffers();
    }

    int main(int argc, char* argv[])
    {
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
        glutCreateWindow("变换实例");
        glutReshapeFunc(ChangeSize);
        glutDisplayFunc(RenderScene);

        SetupRC();
        glutMainLoop();

        return 0;
    }

```



如图 4-1

## 实验五 光照与材质的应用（4 学时）

### 一、实验目的

- 1、掌握在场景中应用简单的光源，使场景看起来更逼真。
- 2、掌握材质的使用。

### 二、实验要求

- 1、注意两种光源的区别，掌握在程序中如何设置光源及光源位置，如何启动光源。

### 三、实验环境

- 1、windows 操作系统；
- 2、OpenGL+VC++6.0。

### 四、实验内容

给一个正方体添加材质和光源，效果如 5-1 所示，有阴影效果。

## 五、实验步骤

1、启动 VC++6.0 软件;



图 5-1

2、创建一个新的 Win32 程序，链接 OpenGL 库文件，在 Project-> Settings,然后单击 LINK 标签。在“Object/LibraryModules”选项中的开始处（在 kernel32.lib 前）增加 OpenGL32.lib, GLu32.lib 和 GLaux.lib 后，单击 OK 按钮;

3、开始编写程序。

按照实验一给出的 OpenGL 程序模板修改。

```

#include <GL/glut.h>

static double speed= 25.0;
void init() //初始化背景颜色，光照，材质等
{
    glClearColor(0.9,0.9,0.8,1.0); //初始背景色
    /***** 光照处理 *****/
    GLfloat light_ambient[] = { 0.0, 0.0, 0.0, 1.0 };
    GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat light_specular[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat light_position0[] = { 3.0, 1.0, 1.0, 1.0 }; //定义光位置得齐次坐标(x,y,z,w),如果w=1.0,
    为定位光源（也叫点光源），
    //如果w=0，为定向光源（无限光源），定向光源为无穷远点，因而产生光为
    //平行光。
    glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient ); //环境光
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse ); //漫射光
    glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular); //镜面反射
    glLightfv(GL_LIGHT0, GL_POSITION, light_position0); //光照位置
    /***** 材质处理 *****/
    GLfloat mat_ambient[] = { 0.0, 0.2, 1.0, 1.0 };
    GLfloat mat_diffuse[] = { 0.8, 0.5, 0.2, 1.0 };
    GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat mat_shininess[] = { 100.0 }; //材质RGBA镜面指数，数值在0~128范围内

    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
    glEnable(GL_LIGHTING); //启动光照
    glEnable(GL_LIGHT0); //使第一盏灯有效
    glEnable(GL_DEPTH_TEST); //测试深度缓存
    /***** 其他可选项 *****/
    // glDepthFunc(GL_LESS); //函数指定比较函数，用来比较每个引入像素的z值和深度缓存中
    给定的z值，只有当
    //激活深度检验时才执行此比较。
    // glEnable(GL_CULL_FACE); //剔除多边形表面:在三维空间中，一个多边形虽然有两个面，
    但我们无法看见背
    //面的那些多边形，而一些多边形虽然是正面的，但被其他多边形所遮挡。如果将
    //无法看见的多边形和可见的多边形同等对待，无疑会降低我们处理图形的效率。
    //在这种时候，可以将不必要的面剔除。
    // glCullFace(GL_FRONT); //glCullFace的参数可以是GL_FRONT，GL_BACK或者
    GL_FRONT_AND_BACK，分别表示
    //剔除正面、剔除反面、剔除正反两面的多边形。
    // glEnable(GL_COLOR_MATERIAL); //材质颜色追踪当前颜色

```



```

}

static void Reshape(int width, int height)
{
    const float ar = (float) width / (float) height;
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION); //对投影矩阵堆栈应用随后的操作。
    glLoadIdentity();
    glFrustum(-ar, ar, -1.0, 1.0, 2.0, 10.0);
    glMatrixMode(GL_MODELVIEW); //对模型矩阵堆栈应用随后的操作。
    //glMatrixMode(GL_TEXTURE); //对纹理矩阵应用随后的操作。
    glLoadIdentity();
}

static void key(unsigned char key, int x, int y)
{
    switch (key)
    {
        {
        case 27 :
        case 'q':
            exit(0);
            break; //按ESC键(ASCII码为27)和q键为退出
        case '+': //按'+'和'-'分别为增加和降低转速
            if(speed>1)
                speed=speed-1.0;
            break;
        case '-':
            speed=speed+1.0;
            break;
        }
        //glutPostRedisplay();
    }

static void idle(void)
{
    glutPostRedisplay(); //标记当前窗口图象层需要重新绘制，在glutMainLoop()的事件处理循环
    的下一个
    //反复中，将调用该窗口的显示回调函数重新绘制该图层。
}

static void display(void)
{
    const double angle = glutGet(GLUT_ELAPSED_TIME)/speed; //调用glutInit函数后运行的时
    间，或者是第一次调用glutGet(GLUT_ELAPSED_TIME)

```

```

//后运行的时间。
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glTranslated(0.0,0.0,-4.0);
glRotated(angle,1,1,0);
glutSolidTeapot(1.0);
glPopMatrix();
//glFlush();
glutSwapBuffers(); //函数交换当前窗口的使用层的缓存，它将后台缓存中的内容交换到前台
缓存中
//如果不是使用双缓存结构的，则glutSwapBuffers()不起任何作用。
}

int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitWindowSize(800,600);
    glutInitWindowPosition(0,0);
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
    //指定RGBA模式窗口，双缓存，深度缓存
    glutCreateWindow("壶");
    init();
    glutReshapeFunc(Reshape);
    glutDisplayFunc(display);
    glutKeyboardFunc(key);
    glutIdleFunc(idle);
    glutMainLoop();
    return 0;
}

```

修改光照和材质的参数，观察场景中三维图形的变化，理解光照和材质参数的意义。适当修改绘图函数 myDisplay 中的内容，自由发挥并绘制其它的三维图形；完成实验报告并提交程序源代码。

## 实验六 纹理的应用（4 学时）

### 一、实验目的

- 1、熟悉纹理位图的载入，并转化为纹理；
- 2、熟悉纹理绑定算法及原理。

## 二、实验要求

- 1、根据实验要求认真填写实验报告，记录所有的实验用例；
- 2、在实验报告中要给出具体的操作要求和过程，并针对各种情况做出具体的分析和讨论。

## 三、实验环境

- 1、windows 操作系统；
- 2、OpenGL+VC++6.0。

## 四、实验内容

把纹理映射到正方体的六个面，效果如图 7-1。



图 7-1

## 五、实验步骤

- 1、启动 VC++6.0 软件；
- 2、创建一个新的 Win32 程序，链接 OpenGL 库文件，在 Project-> Settings,然后单击 LINK 标签。在“Object/LibraryModules”选项中的开始处（在 kernel32.lib 前）增加 OpenGL32.lib, GLu32.lib 和 GLaux.lib 后，单击 OK 按钮；
- 3、开始编写程序。

按照实验一给出的 OpenGL 程序模板修改。

- 载入位图，并转换成纹理，使用函数

```
int LoadGLTextures()           // 载入位图(调用上面的代码)并转换成纹理
LoadBMP("*.bmp")              // 载入位图
int InitGL(GLvoid)             // 此处开始对 OpenGL 进行所有设置
glEnable(GL_TEXTURE_2D);       // 启用纹理映射
int DrawGLScene(GLvoid)        // 从这里开始进行所有的绘制
glBindTexture(GL_TEXTURE_2D, texture[0]); // 绑定纹理
```

- 为了将纹理正确的映射到四边形上，您必须将纹理的右上角映射到四边形的右上角，纹

理的左上角映射到四边形的左上角，纹理的右下角映射到四边形的右下角，纹理的左下角映射到四边形的左下角。

// 前面

```
glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f);    // 纹理和四边形的左下
glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f, 1.0f);    // 纹理和四边形的右下
glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f,  1.0f, 1.0f);    // 纹理和四边形的右上
glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f,  1.0f, 1.0f);    // 纹理和四边形的左上
```

【思考题】试试给立方体的六个面贴上不同的纹理。

## 实验七 绘制 **Bezier** 曲线（2 学时）

### 一、实验目的

- 1、理解 Bezier 曲线的生成方式；
- 2、掌握二次 Bezier 曲线和三次 Bezier 曲线的生成算法。

### 二、实验要求

- 1、使用 C++ 语言实现 Bezier 曲线的绘制。

### 三、实验环境

- 1、windows 操作系统；
- 2、OpenGL+VC++6.0。

### 四、实验内容

- 1、使用 Visual C++ 6.0 和 OpenGL 进行二次 Bezier 曲线和三次 Bezier 曲线的绘制。

### 五、实验步骤

- 1、启动 VC++6.0 软件；
- 2、创建一个新的 Win32 程序，链接 OpenGL 库文件，在 Project-> Settings, 然后单击 LINK 标签。在 “Object/LibraryModules” 选项中的开始处（在 kernel32.lib 前）增加 OpenGL32.lib, GLu32.lib 和 GLaux.lib 后，单击 OK 按钮；
- 3、请将下面的程序写入源文件 Test.c，下面的程序完成二次 Bezier 曲线的绘制；

```
#include <math.h>
#include <gl/glut.h>

int SCREEN_HEIGHT = 480; // 屏幕高度
// 跟踪鼠标点击次数，达到3次后绘制Bezier曲线
int NUMPOINTS = 0;

// 点
class Point {
public:
    float x, y;
    void setxy(float x2, float y2) { x = x2; y = y2; }
    const Point & operator=(const Point &rPoint) {
        x = rPoint.x;
        y = rPoint.y;

        return *this;
    }
};

Point abc[3];

void myInit() {
    glClearColor(0.0,0.0,0.0,0.0);
    glColor3f(1.0,0.0,0.0);
    glPointSize(4.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0,640.0,0.0,480.0);
}

void drawDot(int x, int y) {
    glBegin(GL_POINTS);
    glVertex2i(x,y);
    glEnd();
    glFlush();
}

void drawLine(Point p1, Point p2) {
    glBegin(GL_LINES);
    glVertex2f(p1.x, p1.y);
    glVertex2f(p2.x, p2.y);
}
```

```

    glEnd();
    glFlush();
}

// 计算下一个Bezier曲线上的点
Point drawBezier(Point A, Point B, Point C, double t) {
    Point P;
    P.x = pow((1 - t), 2) * A.x + 2 * t * (1 - t) * B.x + pow(t, 2) * C.x;
    P.y = pow((1 - t), 2) * A.y + 2 * t * (1 - t) * B.y + pow(t, 2) * C.y;
    return P;
}

void myMouse(int button, int state, int x, int y) {
    //左键按下
    if(button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
        //存储鼠标点击的点
        abc[NUMPOINTS].setxy((float)x, (float)(SCREEN_HEIGHT - y));
        NUMPOINTS++;

        // 绘制红点
        drawDot(x, SCREEN_HEIGHT - y);

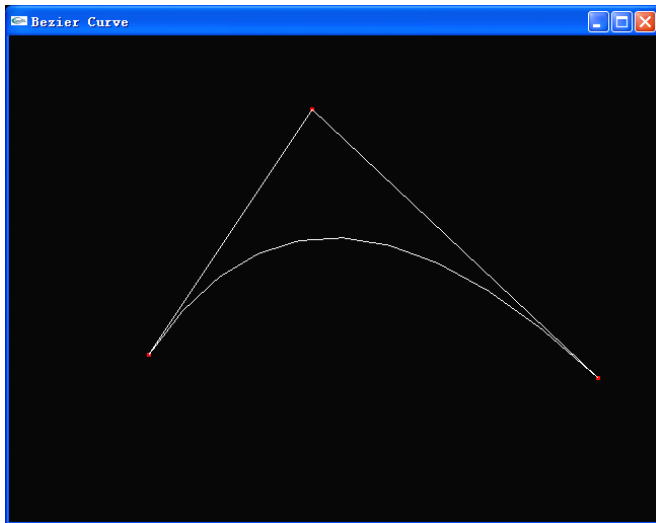
        // 绘制Bezier曲线
        if(NUMPOINTS == 3) {
            glColor3f(1.0, 1.0, 1.0);
            // 绘制控制多边形
            drawLine(abc[0], abc[1]);
            drawLine(abc[1], abc[2]);
            Point POld = abc[0];
            //绘制Bezier曲线段，控制t的增量可以控制曲线精度
            for(double t = 0.0; t <= 1.0; t += 0.1) {
                Point P = drawBezier(abc[0], abc[1], abc[2], t);
                drawLine(POld, P);
                POld = P;
            }
            glColor3f(1.0, 0.0, 0.0);
            NUMPOINTS = 0;
        }
    }
}

void myDisplay() {
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
}

```

```
}

int main(int argc, char *argv[]) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(640,480);
    glutInitWindowPosition(100,150);
    glutCreateWindow("Bezier Curve");
    glutMouseFunc(myMouse);
    glutDisplayFunc(myDisplay);
    myInit();
    glutMainLoop();
    return 0;
}
```



【思考题】扩展程序代码，可以支持 3 次 Bezier 曲线的绘制。