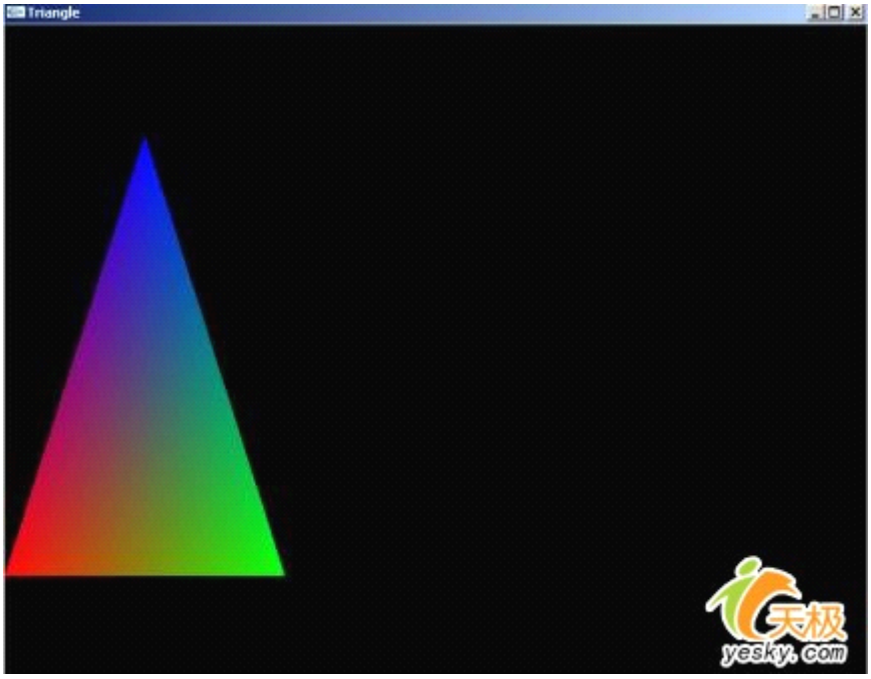
OpenGL 编程轻松入门

先编译运行一个简单的例子，这样我们可以有一个直观的印象

先编译运行一个简单的例子，这样我们可以有一个直观的印象。从这个例子我们可以看到OpenGL 可以做什么，当然这个例子只做了很简单的一件事--绘制一个彩色的三角形。除此以外，我们还可以看到典型的 OpenGL 程序结构及 openGL 的运行顺序。　　例 1：本例在黑色的背景下绘制一个彩色的三角形，如图一所示。

```
#include <stdlib.h>
#include <GL/glut.h>
void background(void)
{
  glClearColor(0.0,0.0,0.0,0.0);//设置背景颜色为黑色
}

void myDisplay(void)
{
  glClear(GL_COLOR_BUFFER_BIT);//buffer 设置为颜色可写

  glBegin(GL_TRIANGLES);//开始画三角形
  glShadeModel(GL_SMOOTH);//设置为光滑明暗模式

  glColor3f(1.0,0.0,0.0);//设置第一个顶点为红色
  glVertex2f(-1.0,-1.0);//设置第一个顶点的坐标为（-1.0，-1.0）

  glColor3f(0.0,1.0,0.0);//设置第二个顶点为绿色
  glVertex2f(0.0,-1.0);//设置第二个顶点的坐标为（0.0，-1.0）

  glColor3f(0.0,0.0,1.0);//设置第三个顶点为蓝色
  glVertex2f(-0.5,1.0);//设置第三个顶点的坐标为（-0.5，1.0）
  glEnd();//三角形结束

  glFlush();//强制 OpenGL 函数在有限时间内运行
}

void myReshape(GLsizei w,GLsizei h)
{
  glViewport(0,0,w,h);//设置视口

  glMatrixMode(GL_PROJECTION);//指明当前矩阵为 GL_PROJECTION
  glLoadIdentity();//将当前矩阵置换为单位阵

  if(w <= h)
```

```
    gluOrtho2D(-1.0,1.5,-1.5,1.5*(GLfloat)h/(GLfloat)w);//定义
二维正视投影矩阵
  else
    gluOrtho2D(-1.0,1.5*(GLfloat)w/(GLfloat)h,-1.5,1.5);
    glMatrixMode(GL_MODELVIEW);//指明当前矩阵为GL_MODELVIEW
}

int main(int argc,char ** argv)
{
  /*初始化*/
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
  glutInitWindowSize(400,400);
  glutInitWindowPosition(200,200);

  /*创建窗口*/
  glutCreateWindow("Triangle");

  /*绘制与显示*/
  background();
  glutReshapeFunc(myReshape);
  glutDisplayFunc(myDisplay);

  glutMainLoop();
  return(0);
}
```



首先创建工程，其步骤如下：　1）创建一个 Win32 Console Application。　2）链接 OpenGL libraries。在 Visual C++中先单击 Project，再单击 Settings，再找到 Link 单击，最后在 Object/library modules 的最前面加上 OpenGL32.lib GLu32.lib GLaux.lib　3）单击 Project Settings 中的 C/C++标签，将 Preprocessor definitions 中的 _CONSOLE 改为__WINDOWS。最后单击 OK。

现在你可以把下面的例子拷贝到工程中去，编译运行。你可以看到一个彩色的三角形。

我们先看看 main 函数。函数中以 glut 开头的函数都包含在 glut.h 中。GLUT 库的函数主要执行如处理多窗口绘制，处理回调驱动事件、生成层叠式弹出菜单、绘制位图字体和笔画字体，以及各种窗口管理等任务。

·glutInit 用来初始化 GLUT 库并同窗口系统对话协商。

·glutInitDisplayMode 用来确定所创建窗口的显示模式。本例中的参数 GLUT_SINGLE 指定单缓存窗口，这也是缺省模式，对应的模式为 GLUT_DOUBLE 双缓存窗口。参数 GLUT_RGB 指定颜色 RGBA 模式，这也是缺省模式，对应的模式为 GLUT_INDEX 颜色索引模式窗口。

·glutInitWindowSize 初始化窗口的大小，第一个参数为窗口的宽度，第二个参数为窗口的高度，以像素为单位。

·glutInitWindowPosition 设置初始窗口的位置，第一个参数为窗口左上角 x 的坐标，第二个参数为窗口左上角 y 的坐标，以像素为单位。屏幕的左上角的坐标为（0，0），横坐标向右逐渐增加，纵坐标向下逐渐增加。

·glutCreateWindow 创建顶层窗口，窗口的名字为扩号中的参数。

·background() 这是自己写的函数，设置背景。其实这个函数中的语句可以写在 display 函数中，但为了使功能块更加清晰，所以把背景这一部分单独提出来。

·glutReshapeFunc 注册当前窗口的形状变化回调函数。当改变窗口大小时，该窗口的形状改变回调函数将被调用。在此例中就是 myReshape 指定形状变化函数。

·glutDisplayFunc 注册当前窗口的显示回调函数。当一个窗口的图像层需要重新绘制时，GLUT 将调用该窗口的的显示回调函数。在此例中的 mydisplay 就是显示回调函数，显示回调函数不带任何参数，它负责整个图像层的绘制。我们的大部分工作将集中在这个函数中。

·glutMainLoop 进入 GLUT 事件处理循环。glutMainLoop 函数在 GLUT 程序中最多只能调用一次，它一旦被调用就不再返回，并且调用注册过的回调函数。所以这个函数必须放在注册回调函数的后面，此例中为glutReshapeFunc，glutDisplayFunc。

现在我们对 OpenGL 程序的典型的程序结构有了一个了解。首先初始化，包括对 GLUT 库的初始化和对窗口的设置及显示模式的设置。第二，创建窗口。第三，自己创作的核心部分。第四，glutMainLoop 进入 GLUT 事件处理循环。

下面，我们转到我们的创作核心。

background 这个函数很简单，只有一行语句。glClearColor 中的四个参数分别是红、绿、蓝和 alpha 值。这些值定义了窗口的颜色。这些值的范围在[0，1]之间。缺省值均为 0。你可以改变这些值，观察背景色彩的变化。

myDisplay 画了一个彩色的三角形。

·glClear 将 buffers 设置为预先设定的值。参数 GL_COLOR_BUFFER_BIT 表明现在可以向 buffer 中写入颜色值。

·glBegin 和 glEnd 是一一对应的。这两个函数限制了一组或多组图元的顶点定义。在这两个函数中间就是你所绘制的由多个顶点组成的图元。函数的参数表明了所绘制的图元的类型。本例中的 GL_TRIANGLES 表明所绘制的图形为三角形。

·glShadeModel 选择平坦或光滑渐变模式。GL_SMOOTH 为缺省值，为光滑渐变模式，GL_FLAT 为平坦渐变模式。

·glColor 设置当前颜色。后面跟的数字为参数个数。3 表明有三个参数，分别为红、绿、蓝，4 则多一个参数 alpha。紧跟数字后面的字母表示数据类型。本例中的 glColor3f 表示三个参数，数据类型为 GLfloat。

·glVertex 指定顶点。同样函数明中的数字表明参数个数。参数分别为 x，y 或 x、y、z。紧跟数字后面的字母表示

数据类型。本例中 glVertex2f 表明两个参数，数据类型为 GLfloat。窗口的中心为原心，坐标为（0，0，0）。横坐标向左为负，向右为正；纵坐标向上为正，向下为负；z 坐标向屏幕里为负，屏幕外为正，坐标系符合右手定则。

现在你将 main 函数中的 glutReshapeFunc(myReshape); 注释掉，任意改变三角形顶点的坐标你会发现窗口的最左边，最右边的 x 值分别为-1 和 1，而窗口的最上端，最下端的 y 值分别为 1 和-1。由此可见 glVertex 中坐标的值实际上是和窗口的大小成倍数的关系。好了，现在恢复原来的程序。

·glFlush 迫使 OpenGL 函数在有限时间里运行。glFlush 清空所有 buffer，使所有发出的命令能在规定的时间内运行。一定不能忘记这一条语句。只有加了这一句，前面的命令才能执行。

myReshape 改变窗口的大小。

·glViewport（Glint x，Glint y，GLsizei width，GLsizei height）设置视口。视口是一个矩形，x，y 为视口左下角的坐标，以像素为单位，缺省值为（0，0）。width 和 height 分别为视口的宽和高。OpenGl context 第一次贴到窗口上时 width 和 height 分别设置成窗口的大小。

·glMatrixMode 指明哪一个矩阵为当前矩阵。本例中 GL_PROJECTION 指明投影矩阵堆栈为随后的矩阵操作的目标。GL_MODELVIEW 指明模型视景矩阵。

·glLoadIdentity 将当前矩阵置换为单位阵。

·gluOrtho2D（GLdouble left，GLdouble right，GLdouble bottom，GLdouble top）定义二维正视投影矩阵。left，right 分别设置左右垂直切平面的坐标，bottom，top 分别设置上下垂直切平面的坐标。

现在你可能对这几个参数的意义还不是很清楚。我们现在将 myReshape 函数 glLoadIdentity()；后面所有的语句注释掉。加上 gluOrtho2D(-1.5,1.5,-1.5,1.5);改变三角型的坐标及 gluOrtho2D 中的数值，你就可以清楚的理解 gluOrtho2D 中数值的含义。

现在，你应该大致了解 OpenGL 到底是怎么一回事。通过这一节的学习我们知道怎样使用颜色及绘制几何图形及物体的最基本的函数，更多的函数你可以在网上或有关书中查到。下一节我们继续讲一下颜色的使用。

OpenGL 编程轻松入门之使用颜色

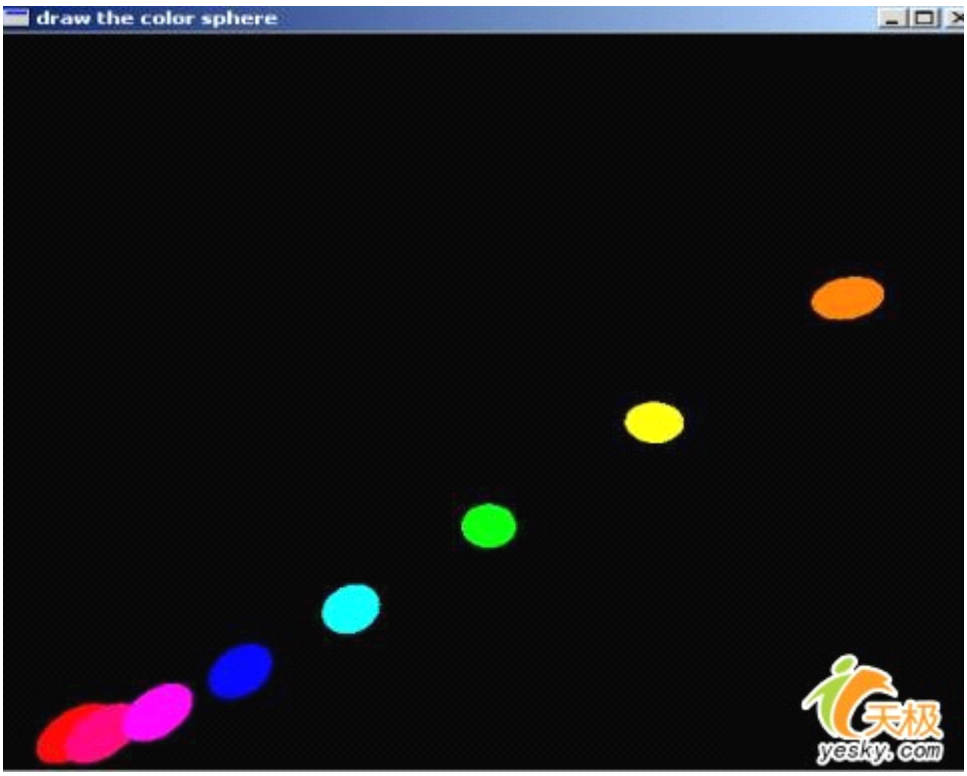
通过上一节的例子我们已经知道一些简单的使用颜色的方法。这一节我们进一步讲讲颜色的使用。

例 2：本例子使用颜色引索模式绘制 8 个不同颜色的球体，如图二所示。阅读此例时，请主要关注函数 palette 和 DrawColotFans。glIndex 设置当前颜色索引。参数为当前颜色索引。本例中 glIndexd 函数的参数 j+1 对应 palette 中 auxSetOneColor 函数中的 i+1，auxSetOneColor 函数的后三个函数制定对应的颜色，颜色值由变量 rgb[8][3]定义。

```
#include <GL/glut.h>
#include <GL/glaux.h>
void init(void)
{
glClearColor(1.0,1.0,1.0,1.0);
glClear(GL_COLOR_BUFFER_BIT);
glShadeModel(GL_SMOOTH);
}
```

```
void palette(void)
{
GLint i;
static GLfloat rgb[8][3]={{1,0,0},{1,0,0.5},{1,0,1},
{0,0,1},{0,1,1},{0,1,0},{1,1,0},{1,0.5,0}};

for(i = 0;i<8;i++)
{
auxSetOneColor(i+1,rgb[i][0],rgb[i][1],rgb[i][2]);//设置颜色
}
}

void DrawColorFans(void)
{
GLint j;

glTranslatef(-15,-15,0);
for(j = 0;j<8;j++)
{
glIndexd(j+1);//设置当前颜色索引
/*在不同位置绘制球体*/
glTranslatef(j,j-1,0);
glutSolidSphere(1,20,20);
}
}

void CALLBACK display(void)
{
palette();
DrawColorFans();
glFlush();
}

void CALLBACK reshape(GLsizei w,GLsizei h)
{
glViewport(0,0,w,h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(100,1,1,20);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(0,0,-15);
}
void main()
```

```
{
auxInitDisplayMode(AUX_SINGLE|AUX_INDEX);
auxInitPosition(100,100,500,500);
auxInitWindow("draw the color sphere");
init();
auxReshapeFunc(reshape);
auxMainLoop(display);
}
```



图二：8 个不同颜色的球体

OpenGL 编程轻松入门之坐标变换

本节中的例子仅仅是将第二节的例子作了一点点改动。将 myDisplay 函数中画三角型的那一部分提出来写成一个函数 drawTriangle。然后在 myDisplay 函数中用 drawTriangle（）；代替原来的语句。这时例 3 和例 1 完成的功能完全一样。而此时我们知道坐标的原点在窗口的中心。我们用 glTranslate 函数改变坐标的原点。同样 glTranslate 函数后的 f 和 d 表明参数的类型。其参数的含义和 glVertex 中参数的含义一样。坐标原点改变后，我们再调用一次 drawTriangle（）；可以发现三角型的位置已经发生了变化，如图三所示。

例 3：利用坐标变换在不同位置画相同的三角形（部分代码）

```
void drawTriangle(void)
```

```
{
glBegin(GL_TRIANGLES);//开始画三角形
glShadeModel(GL_SMOOTH);//设置为光滑明暗模式

glColor3f(1.0,0.0,0.0);//设置第一个顶点为红色
glVertex2f(-1.0,-1.0);//设置第一个顶点的坐标为（-1.0，-1.0）
glColor3f(0.0,1.0,0.0);//设置第二个顶点为绿色
glVertex2f(0.0,-1.0);//设置第二个顶点的坐标为（0.0，-1.0）

glColor3f(0.0,0.0,1.0);//设置第三个顶点为蓝色
glVertex2f(-0.5,1.0);//设置第三个顶点的坐标为（-0.5，1.0）
glEnd();//三角形结束

}

void myDisplay(void)
{
glClear(GL_COLOR_BUFFER_BIT);//buffer 设置为颜色可写

drawTriangle();
glTranslatef(1,0,0);//坐标变换
drawTriangle();

glFlush();//强制 OpenGL 函数在有限时间内运行
}
```

OpenGL 编程轻松入门之堆栈操作

本节中的例子其结构和例 1 一样，仅改变 myDisplay 和 myReshape。

例 4：利用堆栈绘制三个物体--一个绿色的茶壶，一个蓝色的茶壶，一个红色的立方体（部分代码）

```
void myDisplay(void)
{
glClear(GL_COLOR_BUFFER_BIT);
/*蓝色茶壶*/
glPushMatrix();
glColor3f(0.0,0.0,1.0);
glutSolidTeapot(1.5);
glPopMatrix();

/*红色立方体*/
glPushMatrix();
```

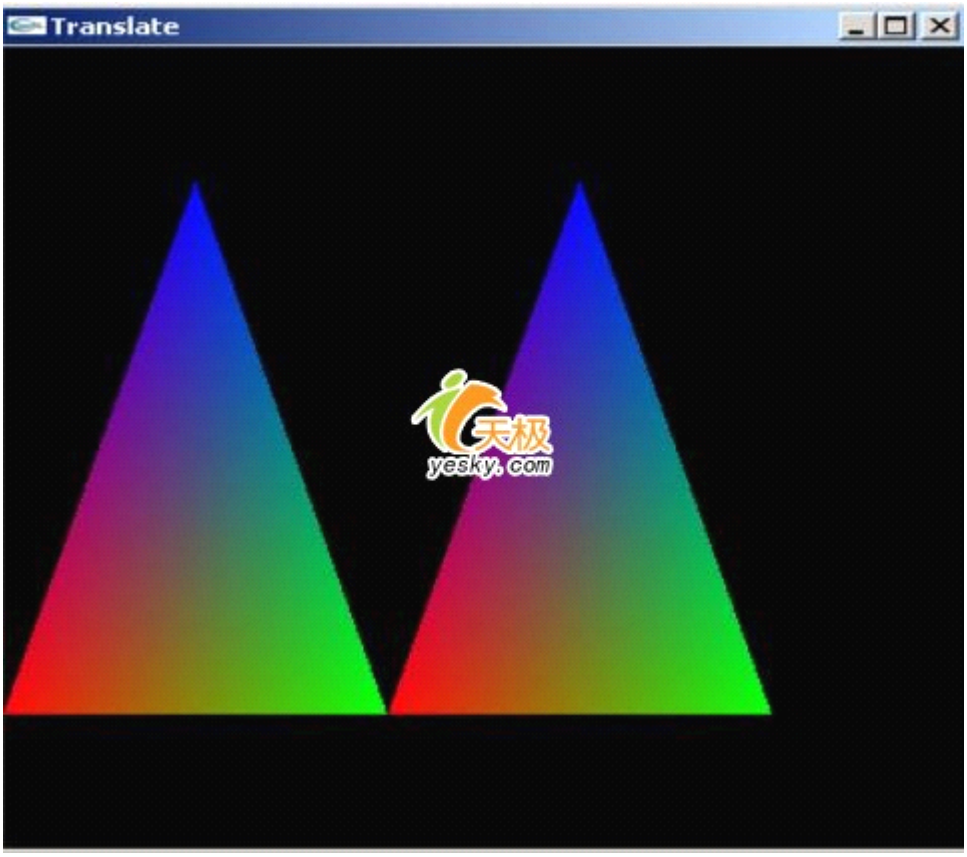
```
glTranslatef(5.0,0.0,0.0);//坐标变换
glColor3f(1.0,0.0,0.0);
glutSolidCube(1.0);
glPopMatrix();

/*绿色茶壶*/
glPushMatrix();
glTranslatef(-5.0,0.0,0.0);//坐标变换
glColor3f(0.0,1.0,0.0);
glutSolidTeapot(1.0);
glPopMatrix();

glFlush();
}

void myReshape(GLsizei w,GLsizei h)
{
glViewport(0,0,w,h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(80.0,(GLdouble)w/(GLdouble)h,1.0,20.0);//创建透视投影矩阵
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(0.0,0.0,-8.0);
}
```

运行后，我们可以看到左边为一个绿色的茶壶，中间为蓝色的茶壶，右边为红色的立方体，如图四。现在我们注释掉所有的 glPushMatrix()；glPopMatrix()；运行后我们会发现两个茶壶重叠在一起，如图五。这是因为 glPushMatrix()；glPopMatrix()；使得坐标转换的原始坐标都是最初始的坐标。而将 glPushMatrix(); glPopMatrix()注释掉后，glTranslatef(5.0,0.0,0.0); glTranslatef(-5.0,0.0,0.0);两行语句使得绿色茶壶回到了（0，0）就和蓝色的茶壶重叠在一起。

除此之外还有很多堆栈操作，需要用时可以通过查 MSDN 或网络或有关书籍。

图四：三个不重叠的物体

图五：两个茶壶重叠在一起



图三：坐标变换后的三角形

glTranslate 是对坐标进行平移，glRotate 对坐标进行旋转，glScale 实际上是对坐标的缩放。还有一些和透视有关的矩阵变换，在以后的例子中我们会接触到一些不同的坐标变换，在这里就不一一例举。

OpenGL 编程轻松入门之显示例表

利用显示列表，我们们可以减少重复的劳动。我们可以从例 5 中得到体会。

例 5：绘制六个彩色的三角形，如图六。

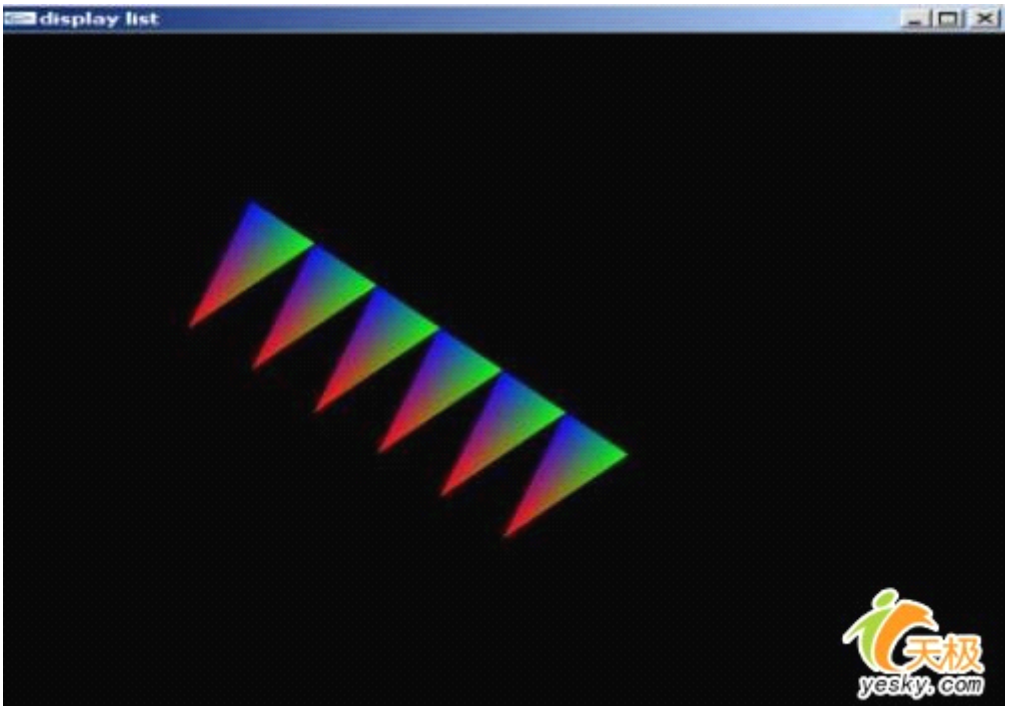
```
#include <windows.h>
#include <GL/glut.h>
GLuint listName = 1;

void myInit(void)
{
glClearColor(0.0,0.0,0.0,0.0);//设置背景为黑色

glNewList(listName,GL_COMPILE);//创建显示列表
/*画一个彩色的多边形*/
glBegin(GL_POLYGON);
glColor3f(1.0,0.0,0.0);
```

```
glVertex2f(1.0,1.0);

glColor3f(0.0,1.0,0.0);
glVertex2f(2.0,2.0);

glColor3f(0.0,0.0,1.0);
glVertex2f(1.5,2.5);

glTranslatef(0.5,-0.5,0.0);//坐标转换
glEnd();
glEndList();//结束显示列表

glShadeModel(GL_SMOOTH);
}


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

for(i = 0;i<6;i++)
glCallList(listName);
glFlush();
}

void myReshape(GLsizei w,GLsizei h)
{
glViewport(0,0,w,h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
if(w<=h)
glOrtho(-4.0,4.0,-4.0*(GLfloat)h/(GLfloat)w,4.0*(GLfloat)h/(GLfloat)w,-8.0,8.0);
else
glOrtho(-4.0,4.0*(GLfloat)w/(GLfloat)h,-4.0,4.0,-8.0,8.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(-4.0,0.0,-3.0);
}

int main(int argc,char ** argv)
{
/*初始化*/
glutInit(&argc,argv);
```

```
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
glutInitWindowSize(500,500);
glutInitWindowPosition(100,200);

/*创建窗口*/
glutCreateWindow("display list");

/*绘制与显示*/
myInit();
glutReshapeFunc(myReshape);

glutDisplayFunc(myDisplay);

/*进入 GLUT 事件处理循环*/
glutMainLoop();

return(0);
}
```

·void glNewList（GLuint list，GLenum mode）和 glEndList（void）创建或替换一个显示列表。list 为列表名称。mode 指定编译模式，本例为 GL_COMPILE。GL_COMPILE 表示仅仅编译。GL_COMPILE_AND_EXECUTE 表示当命令被编译到显示列表时执行。显示列表时一个预先存储起来已被将来执行的一组 OpenGL 命令，使用 glNewList 函数创建显示列表，并将所有需要执行的命令按照命令发出的顺序放置在显示列表中，直到调用 glEndList 函数时结束显示列表。本例中所需要执行的命令为画一个多边形。

·void glCallList（GLuint list）执行一个显示列表。参数 list 为所要执行的显示列表的名字，类型为整形。本例中在不同的位置绘制了六个完全一样的三角形。



图六：六个彩色的三角形

OpenGL 编程轻松入门之使用光照和材质

2006-05-18 11:24 作者： 黄燕 出处： 天极开发 责任编辑：方舟

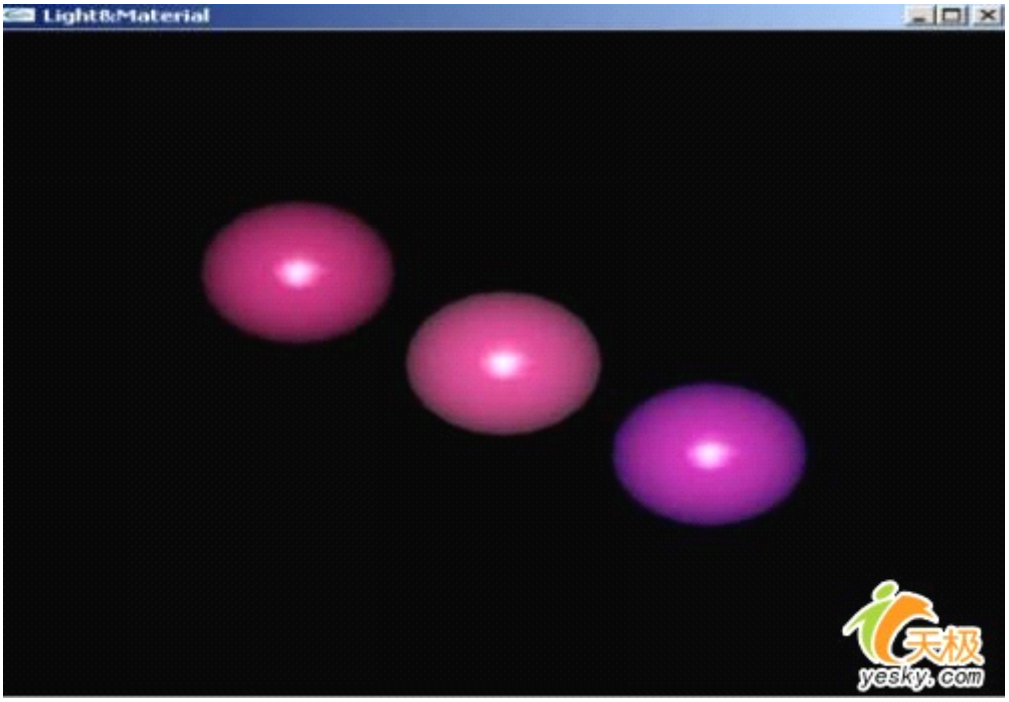
使用了光照和材质可以使物体更逼真，具有立体感。例 4 就是没有使用光照使呈现在我们眼前的物体茶壶和立方体没有立体感。

例 6：绘制三个使用不同材质的球体。

```
#include <windows.h>
#include <GL/glut.h>
GLfloat light_position[] = {0.0,3.0,6.0,0.0};
GLfloat no_mat[] = {0.0,0.0,0.0,1.0};
GLfloat mat_grey_ambient[] = {0.5,0.5,0.5,1.0};
GLfloat mat_red_ambient[] = {0.0,0.0,1.0,1.0};
GLfloat mat_diffuse[] = {0.8,0.2,0.5,1.0};
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};

void myInit(void)
{
glLightfv(GL_LIGHT2,GL_POSITION,light_position);//设置光源
```

```
glEnable(GL_DEPTH_TEST);
glDepthFunc(GL_LESS);//指定深度比较中使用的数值
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glShadeModel(GL_SMOOTH);
}

void display(void)
{
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glPushMatrix();
/*为光照模型指定材质参数*/
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);

glColorMaterial(GL_FRONT,GL_AMBIENT);//使材质色跟踪当前颜色
glEnable(GL_COLOR_MATERIAL);

/*第一个球形*/
glPushMatrix();
glColor3fv(no_mat);
glTranslatef(-2.5,1.5,0.0);
glRotatef(15.0,1.0,0.0,0.0);
glutSolidSphere(1.2,20.0,20.0);
glPopMatrix();

/*第二个球形*/
glPushMatrix();
glColor3fv(mat_grey_ambient);
glRotatef(15.0,1.0,0.0,0.0);
glutSolidSphere(1.2,20.0,20.0);
glPopMatrix();

/*第三个球形*/
glPushMatrix();
glColor3fv(mat_red_ambient);
glTranslatef(2.5,-1.5,0.0);
glRotatef(15.0,1.0,0.0,0.0);
glutSolidSphere(1.2,20.0,20.0);
glPopMatrix();
```

```
glDisable(GL_COLOR_MATERIAL);
glPopMatrix();

glFlush();
}

void myReshape(int w,int h)
{
glViewport(0,0,(GLsizei)w,(GLsizei)h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();

if(w <= h)
glOrtho(-5.5,5.5,-5.5*(GLfloat)h/(GLfloat)w,5.5*(GLfloat)h/(GLfloat)w,-5.5,5.5);
else
glOrtho(-5.5*(GLfloat)w/(GLfloat)h,5.5*(GLfloat)w/(GLfloat)h,-5.5,5.5,-5.5,5.5);

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
}

int main(int argc,char ** argv)
{
/*初始化*/
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(400,400);
glutInitWindowPosition(100,100);

/*创建窗口*/
glutCreateWindow(" Light&Material ");

/*绘制与显示*/
myInit();
glutReshapeFunc(myReshape);
glutDisplayFunc(display);

glutMainLoop();
return 0;
}
```



函数 myInit 中：

·glLight 设置光源的参数。有三个参数，第一个参数指定光照的数目，最大为 8。第二个参数为光照的单值光源参数。第三个参数为赋给第二个参数的值，本例中即为将 light_position 的值赋值给 GL_POSITION。

display 函数中：

·glMaterial 为光照模型指定材质参数。第一个参数为改变材质的面，其值必为 GL_FRONT, GL_BACK 或 GL_FRONT_AND_BACK 中的一个。第二个参数为需要改变的材质参数。第三个参数为赋给参数二的值或指向值的指针。本例中均为指针。

·glColorMateria 使材质色跟踪当前颜色。第一个参数指定哪一个面需要使材质色跟踪当前颜色。第二个参数指定哪个参数需要跟踪颜色。

OpenGL 编程轻松入门之纹理映射

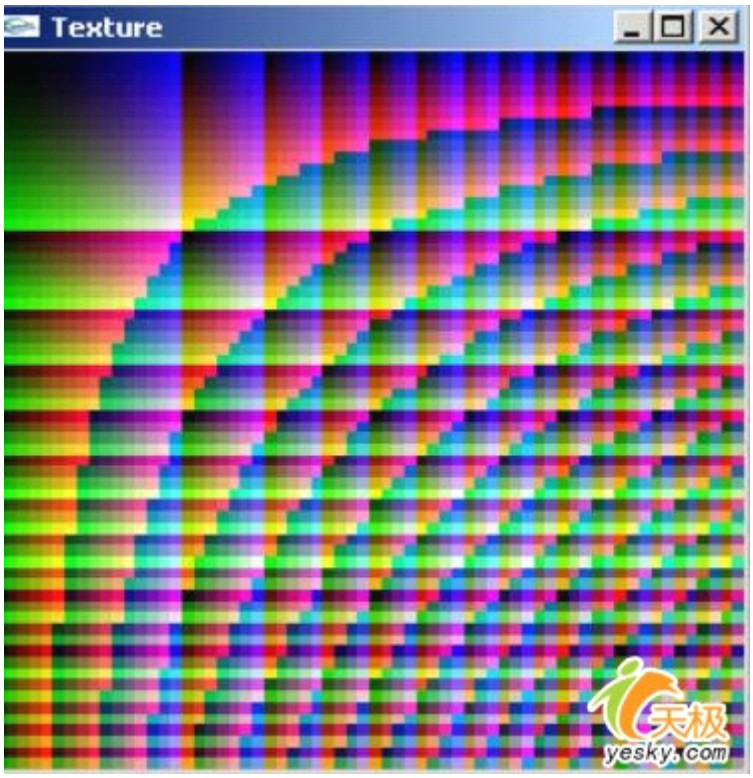
纹理我们可以简单的理解为物体表面的花纹。同样的物体我们可以加上不同的纹理。我们可以使用现成的纹理也可以自己做一个新的纹理。

例 7：绘制一个简单的二维纹理图，并将该图像映射到一个四边形上。

```
#include <stdlib.h>
#include <GL/glut.h>
#define imageWidth 64
#define imageHeight 64
GLubyte image[imageWidth][imageHeight][3];

/*绘制一个简单的二维纹理图*/
void makeImage(void)
{
int i,j,r,g,b;
```

```
/*根据点的位置设置不同的颜色*/
for(i = 0;i < imageWidth;i++)
{
for(j = 0;j <imageHeight;j++)
{
r = (i*j)%255;
g = (i*i)%255;
b = (j*j)%255;

image[i][j][0] = (GLubyte)r;
image[i][j][1] = (GLubyte)g;
image[i][j][2] = (GLubyte)b;
}
}
}

void myInit(void)
{
glClearColor(0.0,0.0,0.0,0.0);

glEnable(GL_DEPTH_TEST);
glDepthFunc(GL_LESS);
makeImage();

glPixelStorei(GL_UNPACK_ALIGNMENT,1);
/*指定二维纹理映射*/

glTexImage2D(GL_TEXTURE_2D,0,3,imageWidth,imageHeight,0,GL_RGB,GL_UNSIGNED_BYTE,&image[0][0][0]);

/*设置纹理参数*/
glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_CLAMP);
glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_CLAMP);
glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);

/*设置纹理环境参数*/
glTexEnvf(GL_TEXTURE_ENV,GL_TEXTURE_ENV_MODE,GL_DECAL);

glEnable(GL_TEXTURE_2D);
glShadeModel(GL_FLAT);
}

void myDisplay(void)
```

```
{
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
/*将纹理映射到四边形上*/
glBegin(GL_QUADS);
/*纹理的坐标和四边形顶点的对应*/
glTexCoord2f(0.0,0.0); glVertex3f(-1.0,1.0,0.0);
glTexCoord2f(0.0,1.0); glVertex3f(-1.0,-1.0,0.0);
glTexCoord2f(1.0,1.0); glVertex3f(1.0,-1.0,0.0);
glTexCoord2f(1.0,0.0); glVertex3f(1.0,1.0,0.0);
glEnd();

glFlush();
}

void myReshape(int w,int h)
{
glViewport(0,0,(GLsizei)w,(GLsizei)h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(80.0,1.0-(GLfloat)w/(GLfloat)h,1.0,30.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
}

int main(int argc,char **argv)
{
/*初始化*/
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(300,300);
glutInitWindowPosition(200,200);

/*创建窗口*/
glutCreateWindow(" Texture ");

/*绘制与显示*/
myInit();
glutReshapeFunc(myReshape);
glutDisplayFunc(myDisplay);

glutMainLoop();
return 0;
}
```



·void glTexImag2D（GLenum target，GLint level，GLint component，GLsizei width，GLsizei height，GLint border，GLenum format，GLenum type，const GLvoid *pixels）；

target 指目标纹理，必须为 GL_TEXTURE_2D 类型。

Level 细节层次数。第 0 层为基本图像级别。

components 指纹理的颜色组成数，必须为 1，2，3 或 4。本例中为 3。这个数字的改变会影响所绘纹理的颜色。

width 和 heigth 分别为纹理图像的宽和高。MSDN 上的帮组说这两个值必为 2^n+2,但我发现这两个值必为 2^n。

border 边界的宽度。

format 指定像素数据格式。可选择值为 GL_COLOR_INDEX ，GL_RED ，GL_GREEN，GL_BLUE ， GL_ALPHA ， GL_RGB ， GL_RGBA ， GL_BGR_EXT ， GL_BGR_EXT ，GL_BGRA_EXT ，GL_BGRA_EXT， GL_LUMINANCE，GL_LUMINANCE_ALPHA

type 指定像素数据类型。可选择值为 GL_UNSIGNED_BYTE, GL_BYTE, GL_BITMAP, GL_UNSIGNED_SHORT, GL_SHORT, GL_UNSIGNED_INT, GL_INT, and GL_FLOAT.

pixels 指向存储在内存中的图像数据的指针。

·glTexParameter 设置纹理参数。三个参数。第一个参数指定纹理目标，必为 GL_TEXTURE_1D 或 GL_TEXTURE_2D；第二个参数为指定单值纹理参数的名称，第三个参数为赋给第二个参数的值。

·glTexEnv 设置纹理环境参数。三个参数。第一个参数为纹理环境，必为 GL_TEXTURE_ENV。第二个参数为纹理环境参数的名称。第三个参数为单值符号常数或指向参数数组的指针。

·glTexCoord 设置当前纹理坐标。

OpenGL 编程轻松入门之特殊效果操作

2006-05-19 11:00 作者： 黄燕 出处： 天极开发 责任编辑：方舟

每个物体在不同的环境在会有不同的视觉效果。为了使效果更加真实需要增加特殊效果。

例 8：绘制三个互相重叠的矩形，这三个矩形从左到右分别为绿、红、蓝。其中红色、蓝色矩形的透明度分别为 50%的透明度，即 alpha 值为 0.5，如图九所示。

```
#include <stdlib.h>
#include <GL/glut.h>
void myInit(void)
{
glClearColor(0.2,0.8,0.8,0.0);//将背景设置为湖蓝色
glEnable(GL_BLEND);//激活 GL_BLEND
glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);//指定像素的数学算法
glEnable(GL_FLAT);
}

void myDisplay(void)
{
glClear(GL_COLOR_BUFFER_BIT);

/*绘制一个绿色的矩形  alpha = 1.0*/
glColor4f(0.0,1.0,0.0,1.0);

glBegin(GL_POLYGON);
glVertex3f(-0.75,0.5,0.0);
glVertex3f(-0.75,-0.75,0.0);
glVertex3f(0.5,-0.75,0.0);
glVertex3f(0.5,0.5,0.0);
glEnd();

/*绘制一个红色的矩形  alpha = 0.5*/
glColor4f(1.0,0.0,0.0,0.5);

glBegin(GL_POLYGON);
glVertex3f(-0.25,1.0,0.0);
glVertex3f(-0.25,-0.25,0.0);
glVertex3f(1.0,-0.25,0.0);
glVertex3f(1.0,1.0,0.0);
glEnd();

/*绘制一个蓝色的矩形  alpha = 0.5*/
glColor4f(0.0,0.0,1.0,0.5);
```

```
glBegin(GL_POLYGON);
glVertex3f(0.25,1.5,0.0);
glVertex3f(0.25,0.25,0.0);
glVertex3f(1.5,0.25,0.0);
glVertex3f(1.5,1.5,0.0);
glEnd();

glFlush();
}

void myReshape(int w,int h)
{
glViewport(0,0,w,h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();

if(w<h)
glOrtho(-1.5,1.5,-1.5*(GLfloat)h/(GLfloat)w,1.5*(GLfloat)h/(GLfloat)w,-1.5,1.5);
else
glOrtho(-1.5*(GLfloat)w/(GLfloat)h,1.5*(GLfloat)w/(GLfloat)h,-1.5,1.5,-1.5,1.5);

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(-0.4,0.0,0.0);
}

int main(int argc,char ** argv)
{
/*初始化*/
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(300,400);
glutInitWindowPosition(200,200);

/*创建窗口*/
glutCreateWindow("BLEND POLYGON");

/*绘制与显示*/
myInit();
glutReshapeFunc(myReshape);
glutDisplayFunc(myDisplay);

/*进入 GLUT 事件处理循环*/
```

```
glutMainLoop();
return(0);
}
```

图九：三个互相重叠的矩形

·glBlendFunc（GLenum sfactor，GLenum dfactor） 指定像素算法。sfactor 指定红，绿，蓝及 alpha 源混合因素是如何计算的。dfactor 指定红，绿，蓝及 alpha 目标混合因素是如何计算的。

例 9：绘制一个被雾化的圆锥体，如图十所示，为了观察不同的雾化参数，程序中加入了键盘操作。"shift+1"，" shift+2"键分别增加和减弱雾化浓度，"shift+3"设置雾化的起始点和终止点，"shift+4"和"shift+5"键改变雾化方程，"shift+6"将雾化颜色由白改为绿色，如图十一所示。

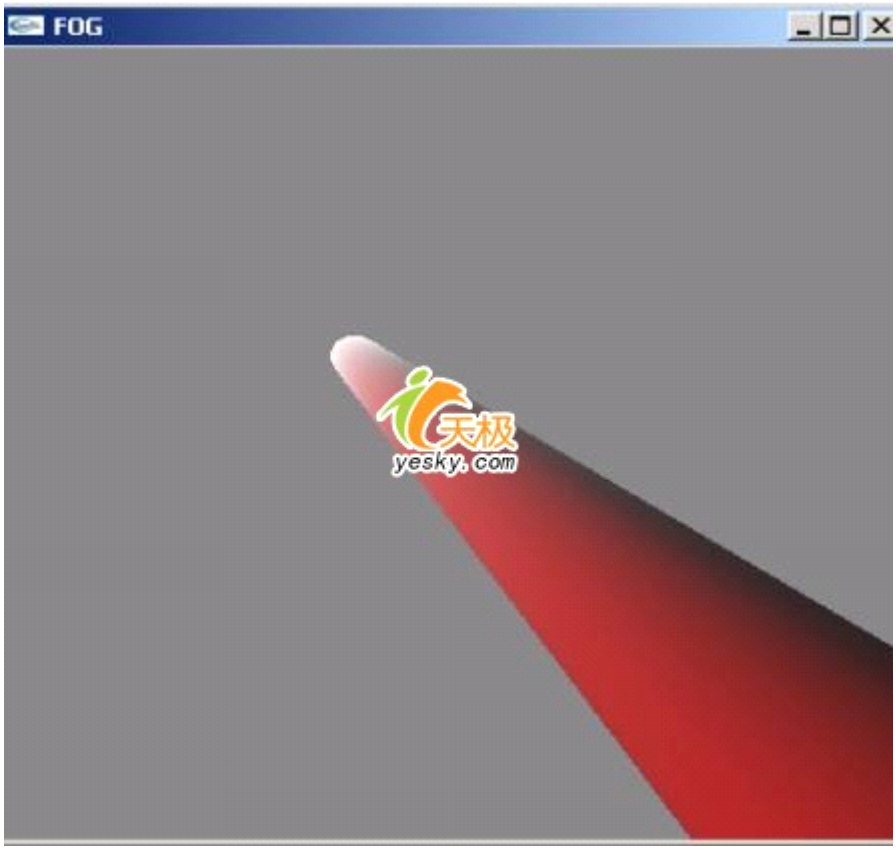
```
#include <stdlib.h>
#include <GL/glut.h>

GLfloat light_ambient[] = {0.1,0.1,0.1,0.0};
GLfloat light_diffuse[] = {1.0,1.0,1.0,0.0};
GLfloat light_specular[] = {1.0,1.0,1.0,0.0};
```

```
GLfloat light_position[] = {-10.0,0.0,5.0,0.0};

GLfloat material_ambient[] = {0.1745,0.01175,0.01175};
GLfloat material_diffuse[] = {0.61424,0.04136,0.04136};
GLfloat material_specular[] = {0.727811,0.626959,0.62659};

GLfloat fogColorWhite[] = {1.0,1.0,1.0,1.0};
GLfloat fogColorGreen[] = {0.0,1.0,0.0,1.0};
float fogDensity = 0.02;


void myInit(void)
{
/*设置背景色*/
glClearColor(0.5,0.5,0.5,1.0);

/*设置光照*/
glLightfv(GL_LIGHT0,GL_AMBIENT,light_ambient);
glLightfv(GL_LIGHT0,GL_DIFFUSE,light_diffuse);
glLightfv(GL_LIGHT0,GL_SPECULAR,light_specular);
glLightfv(GL_LIGHT0,GL_POSITION,light_position);

/*设置材质*/
glMaterialfv(GL_FRONT,GL_AMBIENT,material_ambient);
glMaterialfv(GL_FRONT,GL_DIFFUSE,material_diffuse);
glMaterialfv(GL_FRONT,GL_SPECULAR,material_specular);
glMaterialf(GL_FRONT,GL_SHININESS,0.6*128);

glShadeModel(GL_SMOOTH);

glEnable(GL_DEPTH_TEST);
glDepthFunc(GL_LESS);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glEnable(GL_AUTO_NORMAL);
glEnable(GL_NORMALIZE);
glFrontFace(GL_CW);

/*设置雾化*/
glEnable(GL_FOG);
glFogi(GL_FOG_MODE,GL_LINEAR);
glFogfv(GL_FOG_COLOR,fogColorWhite);
glFogf(GL_FOG_DENSITY,fogDensity);
glFogf(GL_FOG_START,0.0);
```

```
glFogf(GL_FOG_END,15.0);

glHint(GL_FOG_HINT,GL_DONT_CARE);
}

/*根据不同的键值设置不同的雾化效果*/
static void myKey(unsigned char key,int x,int y)
{
switch(key)
{
case 33: //shift+1
fogDensity *= 1.10;
glFogi(GL_FOG_MODE,GL_EXP);
glFogf(GL_FOG_DENSITY,fogDensity);
glutPostRedisplay();
break;

case 64: //shift+2
fogDensity /= 1.10;
glFogi(GL_FOG_MODE,GL_EXP);
glFogf(GL_FOG_DENSITY,fogDensity);
glutPostRedisplay();
break;

case 35: //shift+3
glFogi(GL_FOG_START,0.0);
glFogi(GL_FOG_END,8.0);
glutPostRedisplay();
break;

case 36: //shift+4
glFogi(GL_FOG_MODE,GL_EXP2);
glutPostRedisplay();
break;

case 37: //shift+5
glFogi(GL_FOG_MODE,GL_LINEAR);
glutPostRedisplay();
break;

case 94: //shift+6
glFogfv(GL_FOG_COLOR,fogColorGreen);
glutPostRedisplay();
break;
```

```
case 27: //Esc
exit(0);

default:
break;
}
}

/*绘制圆锥体*/
void myDisplay(void)
{
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

glPushMatrix();
glTranslatef(-4.0,4.0,0.0);
glRotatef(30.0,1.0,1.0,0.0);
glutSolidCone(1.0,50.0,20.0,20.0);
glPopMatrix();
glutSwapBuffers();
}

void myReshape(int w,int h)
{
glViewport(0,0,(GLsizei)w,(GLsizei)h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(100.0,1.0,1.0,20.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(0.0,0.0,-15.5);
}

int main(int argc,char ** argv)
{
/*初始化*/
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
glutInitWindowSize(300,300);
glutInitWindowPosition(100,100);

/*创建窗口*/
glutCreateWindow(" FOG ");
```

```
/*绘制与显示*/
myInit();
glutKeyboardFunc(myKey);
glutReshapeFunc(myReshape);
glutDisplayFunc(myDisplay);

/*进入 GLUT 事件处理循环*/
glutMainLoop();
return 0;
}
```



图十：被雾化的圆锥体

·glFog 指定雾化参数。后面不同的字母表明参数的数据类型。f 表示 float，i 表示 integer，v 表示 vector，也就是表明一个指针。

GL_FOG_MODE 是一个单值整数或浮点数，该数值指定了用来计算雾化融合因子 f 的方程。

GL_DENSITY 是一个单值整数或浮点数，该值指定雾化浓度。

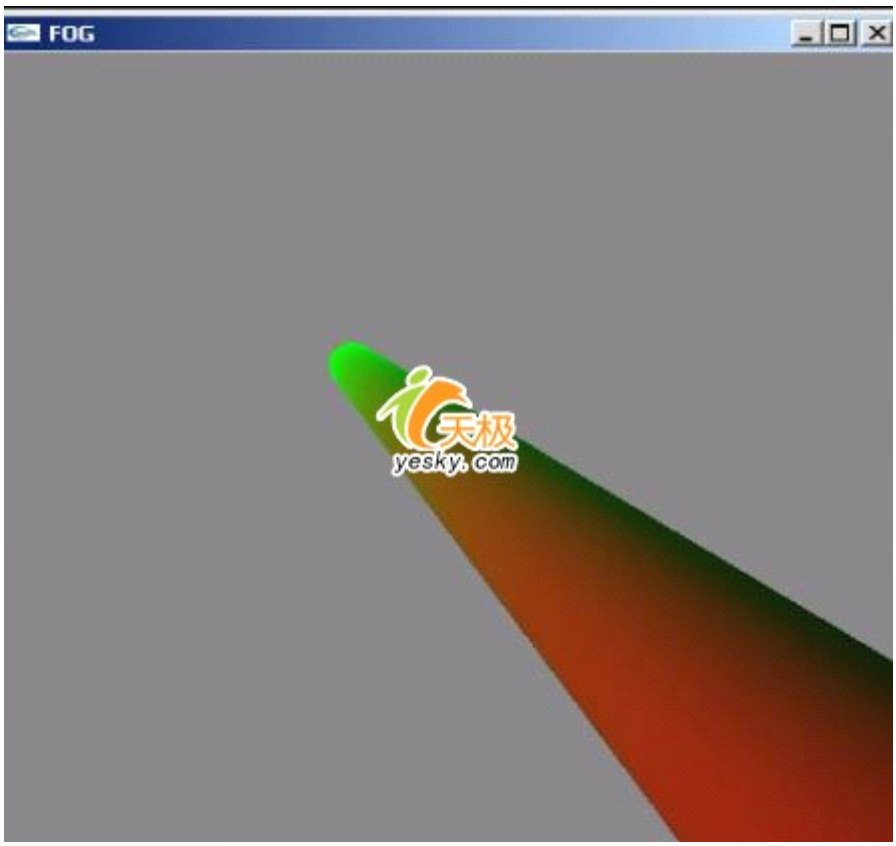
GL_FOG_START 是一个单值整数或浮点数，该值指定雾化的起始值。

GL_FOG_END 是一个单值整数或浮点数，该值指定雾化的终止值。

GL_FOG_INDEX 是一个单值整数或浮点数，该值指定雾化索引值 if，缺省的雾化索引值为 0.0。

GL_FOG_COLOR 包含 4 个整数值或浮点数值，这些数值指定的是雾化颜色 Cf，整数值进行线性映射。

·glHint 指定实现的线索。本例中 GL_FOG_HINT 指定雾化计算精度。GL_DONT_CARE 指对选项不做考虑。



图十一：雾化为绿色的圆锥体

本例中还涉及到键盘操作，glutKeyboardFun。此函数注册当前窗口的键盘回调函数。因为这不是本节的主要内容，我们只简单提一下。此函数的原形为 void glutKeyboardFunc（void（*func）（unsigned char key， int x， int y））所以我们自己写的回调函数必须按照原形上规定的写。

OpenGL 编程轻松入门之曲面和曲线

前面我们讲了如何绘制平面的图形，这一节我们学习如何绘制曲线和曲面。

前面我们讲了如何绘制平面的图形，这一节我们学习如何绘制曲线和曲面。

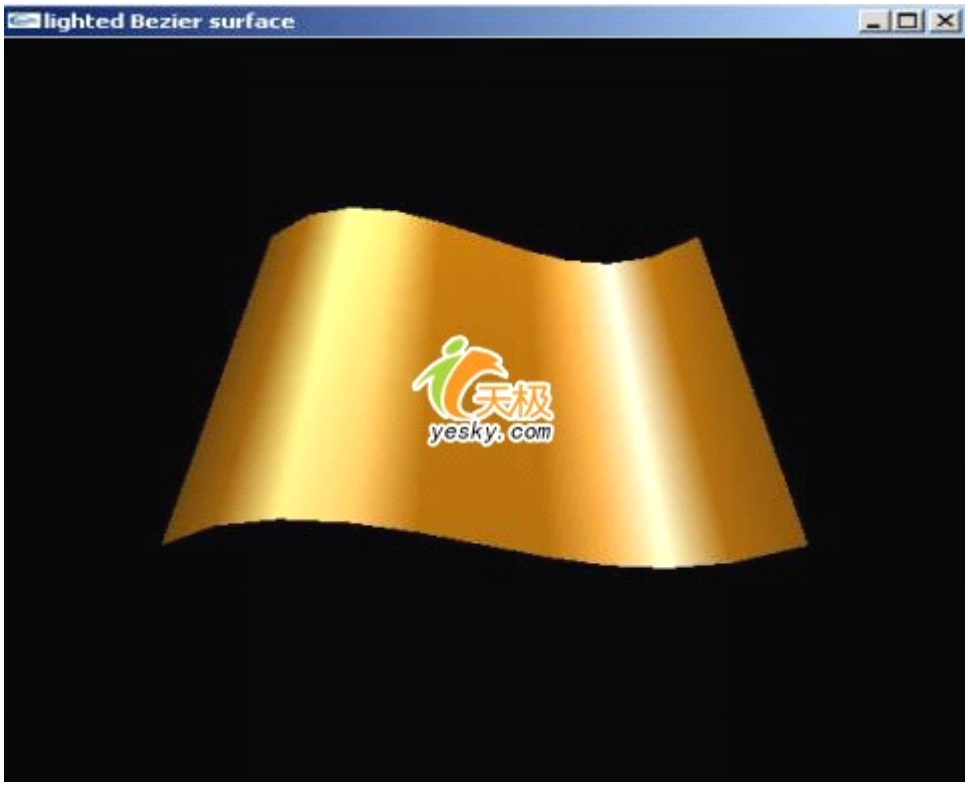
例 10：绘制一个曲面，如图十二所示。本程序使用二维求值器绘制一个曲面。本例中也有一些特殊效果的操作。

```
#include <windows.h>
#include <GL/GLAUX.h>
#include <GL/glut.h>
#include <math.h>
```

```
GLfloat                    ctrlpoints[5][5][3]                    =
{{{-2,0,0},{-1,1,0},{0,0,0},{1,-1,0},{2,0,0}},
{{-2,0,-1},{-1,1,-1},{0,0,-1},{1,-1,-1},{2,0,-1}},
{{-2,0,-2},{-1,1,-2},{0,0,-2},{1,-1,-2},{2,0,-2}},
{{-2,0,-3},{-1,1,-3},{0,0,-3},{1,-1,-3},{2,0,-3}},
{{-2,0,-4},{-1,1,-4},{0,0,-4},{1,-1,-4},{2,0,-4}}};

GLfloat mat_ambient[] = {0.1,0.1,0.1,1.0};
GLfloat mat_diffuse[] = {1.0,0.6,0.0,1.0};
GLfloat mat_specular[] = {1.0,1.0,1.0,1.0};

GLfloat light_ambient[] = {0.1,0.1,0.1,1.0};
GLfloat light_diffuse[] = {1.0,1.0,1.0,0.0};
GLfloat light_specular[] = {1.0,1.0,1.0,0.0};
GLfloat light_position[] = {2.0,23.0,-4.0,1.0};

void myInit(void)
{
  glClearColor(0.0,0.0,0.0,0.0);//设置背景色

  /*为光照模型指定材质参数*/
  glMaterialfv(GL_FRONT,GL_AMBIENT,mat_ambient);
  glMaterialfv(GL_FRONT,GL_DIFFUSE,mat_diffuse);
  glMaterialfv(GL_FRONT,GL_SPECULAR,mat_specular);
  glMaterialf(GL_FRONT,GL_SHININESS,60.0);

  /*设置光源参数*/
  glLightfv(GL_LIGHT0,GL_AMBIENT,light_ambient);
  glLightfv(GL_LIGHT0,GL_DIFFUSE,light_diffuse);
  glLightfv(GL_LIGHT0,GL_SPECULAR,light_specular);
  glLightfv(GL_LIGHT0,GL_POSITION,light_position);

  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);

  /*enable depth comparisons and update the depth buffer*/
  glEnable(GL_DEPTH_TEST);
  /*设置特殊效果*/
  glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
  glHint(GL_LINE_SMOOTH_HINT,GL_DONT_CARE);
  glEnable(GL_BLEND);

  glEnable(GL_AUTO_NORMAL);
  glEnable(GL_NORMALIZE);
```

```
  glFrontFace(GL_CW);
  glShadeModel(GL_SMOOTH);
  glEnable(GL_LINE_SMOOTH);

}

void myDisplay(void)
{
  glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
  glColor3f(0.0,0.0,0.0);
  glTranslatef(0.0,-1.0,0.0);
  glRotatef(50.0,1.0,0.0,0.0);
  glPushMatrix();
  /*绘制曲面*/
  glEnable(GL_MAP2_VERTEX_3);

glMap2f(GL_MAP2_VERTEX_3,0,1,3,5,0,1,15,5,&ctrlpoints[0][0][0]);

  glMapGrid2f(10.0,0.0,1.0,10.0,0.0,1.0);
  glEvalMesh2(GL_FILL,0,10.0,0,10.0);
  glPopMatrix();
  glutSwapBuffers();
}

void myReshape(GLsizei w,GLsizei h)
{
  glViewport(0,0,w,h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(60.0,(GLfloat)w/(GLfloat)h,1.0,100.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glTranslatef(0.0,0.0,-5.0);
}

int main(int argc,char ** argv)
{
  /*初始化*/
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
  glutInitWindowSize(400,400);
  glutInitWindowPosition(200,200);

  /*创建窗口*/
```

```
  glutCreateWindow("lighted Bezier surface");

  /*绘制与显示*/
  myInit();
  glutReshapeFunc(myReshape);
  glutDisplayFunc(myDisplay);

  /*进入 GLUT 事件处理循环*/
  glutMainLoop();
  return(0);
}
```



图十二：曲面

myInit（）中的几个有关特殊效果的操作。

·glBlendFunc（GLenum sfactor，GLenum dfactor） 指定像素算法。sfactor 指定红，绿，蓝及 alpha 源混合因素是如何计算的。dfactor 指定红，绿，蓝及 alpha 目标混合因素是如何计算的。

·glHint（GLenum target，GLenum mode）指定操作线索。

Target 为需要控制的符号常量。mode 为所希望的行为符号常数。本例中 GL_LINE_SMOOTH_HINT 指定反走样线段的采样质量。GL_DONT_CARE 指对选项不做考虑。

myDisplay（）中的曲面操作： ·void glMap2f（GLenum target，GLfloat u1,GLfloat u2,Glint ustride, Glint uorder, GLfloat v1, GLfloat v2, Glint vstride, Glint vorder, const GLfloat *points）;定义 2 维求值器。

target 指定求值器生成的数值类型。本例中的 GL_MAP2_VERTEX_3 指明每一个控制点为 x、y、z 表示的三个浮点值。 u1，u2 指定线性映射。

ustride 指定控制点 Rij 的起始点和控制点 R（i+1）j 的的起始点之间单精度或双精度浮点值的个数。这里 i 和 j 分

别是 u 和 v 控制点索引，它允许控制点装入任意的数据结构中。唯一的限制是对于特定控制点的数值必须存在连续的内存单元。

uorder 控制点数组在 u 轴方向上的维数。

v1，v2 指定线性映射 v

vstride 指定控制点 Rij 的起始点和控制点 Ri（j+ 1）的起始点之间单精度或双精度浮点值的个数。这里 i 和 j 分别是 u 和 v 控制点索引，它允许控制点装入任意的数据结构中。唯一的限制是对于特定控制点的数值必须存在连续的内存单元。

vorder 控制点数组在 v 轴方向上的维数。

points 一个指向控制点数组的指针。

·glMapGrid 定义一维或二维网格。void glMapGrid2f（Glint un, GLfloat u1, GLfloat u2,Glint vn, GLfloat v1,GLfloat v2）；

un 在网格[u1，u2]中的分段数目。

u1，u2 指定整数网格范围 i= 0；i= un 的映射。

vn 在网格[v1，v2]中的分段数目。

v1，v2 指定整数网格范围 j = 0；j= vn 的映射。

·glEvalMesh 计算一维或二维点或线网格。本例中为 2 维。void glEvalMesh2（GLenum mode，Glint i1，Glint i2，Glint j1，Glint j2）；

mode 指定是否计算二维点、线或多边形的网格。

i1，i2 分别为网格定义域变量 i 的第一个和最后一个整数值。

j1，j2 分别为网格定义域变量 j 的第一个和最后一个整数值。

glMapGrid 和 glEvalMesh 用来生成并求取一系列等间隔的网格点，glEvalMesh 逐步计算一维或二维网格，他的定义范围由 glMap 指定。mode 决定最终计算的顶点是绘制为点、线还是充实的多边形。具体的映射关系及有关图形方面的知识，你可以很方便的在 MSDN、互联网及有关书籍中查到，本文就不详述这方面的内容。

OpenGL 编程轻松入门之 NURBS 曲线和曲面

上一节讲了一般的曲线与曲面的绘制，本节讲 NURBS 曲线和曲面的绘制。

例 11：此例绘制两个相同形状的 NURBS 曲面，不同之处是一个为线框式，一个是由实多边形组成。运行后可以看到其中的区别，如图十三所示。

```
#include <windows.h>
#include <GL/glut.h>
GLUnurbsObj *theNurb1;
GLUnurbsObj *theNurb2;

GLfloat ctrlpoints[5][5][3] = {{{-3,0.5,0},{-1,1.5,0},{-2,2,0},{1,-1,0},{-5,0,0}},
{{-3,0.5,-1},{-1,1.5,-1},{-2,2,-1},{1,-1,-1},{-5,0,-1}},
{{-3,0.5,-2},{-1,1.5,-2},{-2,2,-2},{1,-1,-2},{-5,0,-2}},
{{-3,0.5,-3},{-1,1.5,-3},{-2,2,-3},{1,-1,-3},{-5,0,-3}},
{{-3,0.5,-4},{-1,1.5,-4},{-2,2,-4},{1,-1,-4},{-5,0,-4}}};//控制点

GLfloat mat_diffuse[] = {1.0,0.5,0.1,1.0};
GLfloat mat_specular[] = {1.0,1.0,1.0,1.0};
GLfloat mat_shininess[] = {100.0};
GLfloat light_position[] = {0.0,-10.0,0.0,1.0};
```

```
void myInit(void)
{
glClearColor(1.0,1.0,1.0,0.0);//设置背景色

/*为光照模型指定材质参数*/
glMaterialfv(GL_FRONT,GL_DIFFUSE,mat_diffuse);
glMaterialfv(GL_FRONT,GL_SPECULAR,mat_specular);
glMaterialfv(GL_FRONT,GL_SHININESS,mat_shininess);

glLightfv(GL_FRONT,GL_POSITION,light_position);//设置光源参数
glLightModeli(GL_LIGHT_MODEL_TWO_SIDE,GL_TRUE);//设置光照模型参数

/*激活光照*/
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);

glDepthFunc(GL_LEQUAL);
glEnable(GL_DEPTH_TEST);
glEnable(GL_LEQUAL);
glEnable(GL_AUTO_NORMAL);
glEnable(GL_NORMALIZE);

/*设置特殊效果*/
glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
glHint(GL_LINE_SMOOTH_HINT,GL_DONT_CARE);
glEnable(GL_BLEND);

glFrontFace(GL_CW);
glShadeModel(GL_SMOOTH);
glEnable(GL_LINE_SMOOTH);

theNurb1 = gluNewNurbsRenderer();//创建 NURBS 对象 theNurb1
gluNurbsProperty(theNurb1,GLU_SAMPLING_TOLERANCE,25.0);
gluNurbsProperty(theNurb1,GLU_DISPLAY_MODE,GLU_OUTLINE_POLYGON);

theNurb2 = gluNewNurbsRenderer();//创建 NURBS 对象 theNurb2
gluNurbsProperty(theNurb2,GLU_SAMPLING_TOLERANCE,25.0);
gluNurbsProperty(theNurb2,GLU_DISPLAY_MODE,GLU_FILL);
}

int spin = 0;

/*接收键盘指令*/
```

```
static void myKey(unsigned char key,int x,int y)
{
switch(key)
{
case'd':
spin = spin + 1;
glRotatef(spin,1.0,1.0,0.0);
glutPostRedisplay();
break;
case 27:
exit(0);
default:
break;
}
}

/*绘制曲面*/
void myDisplay(void)
{
GLfloat knots[10] = {0.0,0.0,0.0,0.0,0.0,1.0,1.0,1.0,1.0,1.0};

glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glRotatef(50.0,1.0,1.0,0.0);

/*第一个曲面*/
glPushMatrix();
glTranslatef(1.0,0.0,0.0);
gluBeginSurface(theNurb1);
/*定义曲面形状*/
gluNurbsSurface(theNurb1,10,knots,10,knots,5*3,3,&ctrlpoints[0][0][0],5,5,GL_MAP2_VERTEX_3);
gluEndSurface(theNurb1);
glPopMatrix();

/*第二个曲面*/
glPushMatrix();
glTranslatef(7.0,0.0,0.0);
gluBeginSurface(theNurb2);
/*定义曲面形状*/
gluNurbsSurface(theNurb2,10,knots,10,knots,5*3,3,&ctrlpoints[0][0][0],5,5,GL_MAP2_VERTEX_3);
gluEndSurface(theNurb2);
glPopMatrix();

glutSwapBuffers();
}
```

```
void myReshape(GLsizei w,GLsizei h)
{
glViewport(0,0,w,h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(50.0,(GLfloat)w/(GLfloat)h,1.0,15.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(0.0,0.0,-9.0);
}

int main(int argc,char ** argv)
{
/*初始化*/
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
glutInitWindowSize(600,400);
glutInitWindowPosition(200,200);

/*创建窗口*/
glutCreateWindow("NURBS surface");

/*绘制与显示*/
myInit();
glutKeyboardFunc(myKey);
glutReshapeFunc(myReshape);
glutDisplayFunc(myDisplay);

/*进入 GLUT 事件处理循环*/
glutMainLoop();
return(0);
}
```

·GLUnurbsObj* glNewNurbsRenderer（）创建一个 NURBS 对象，并返回一个指向该对象的指针。如果没有足够的内存分配给该对象，则返回值为 0。

·void gluNurbsProperty(GLUnurbsObj* nobj, GLenum property, GLfloat value)设置 NURBS 属性。

nobj 指向 NURBS 对象的指针。

property 需设置的属性。

value 设置指定属性的值。

·glBeginSurface 及 gluEndSurface 两个函数一起限定一个 NURBS 面的定义。返回值均为 void，参数均为 GLUnurbsObj* nobj，为指向 NURBS 对象的指针。

·void gluNurbsSurface(GLUnurbsObj *nobj, Glint knot_count, GLfloat tknot_count, GLfloat *tknot, Glint s_stride, Glint t_stride, GLfloat *ctlarry, GLint sorder, GLint torder,GLenum type)
定义 NURBS 曲面形状。

nobj 指向 NURBS 对象的指针。

sknot_count 参数化 u 方向上的节点数。

sknot 参数化 u 方向上的非递减节点值。

tknot_count 参数化 v 方向上的节点数。

tknot 参数化 v 方向上的非递减节点值。

s_stride 在 ctlarry 中参数化 u 方向上相邻控制点的偏移量。

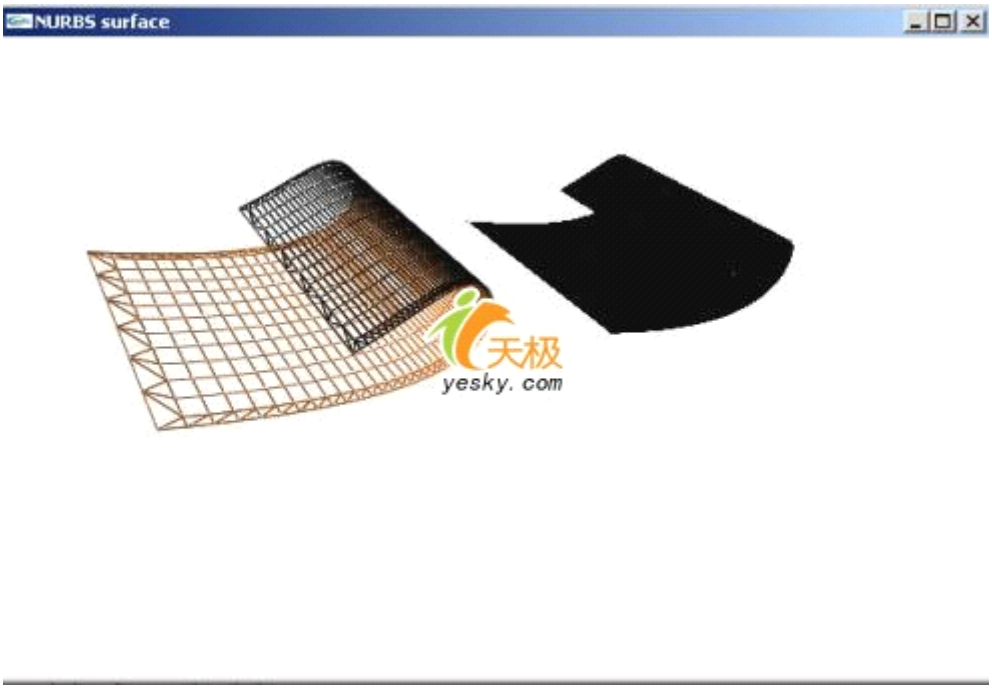
t_stride 在 ctlarry 中参数化 v 方向上相邻控制点的偏移量。

ctlarryNURBS 的控制点数组。

sorder 参数化 u 方向上 NURBS 的阶数，阶数比维数大 1。

torder 参数化 v 方向上 NURBS 的阶数，阶数比维数大 1。

type 曲面类型。



图十三：NURBS 曲面

例 12：绘制一个彩色的曲线，曲线闭合成圆。在曲线的边缘绘制 8 个点，如图十四所示。

```
#include <windows.h>
#include <GL/glut.h>

GLUnurbsObj *theNurb;

GLfloat ctrlpoints[12][3] = {{4,0,0},{2.828,2.828,0},{0,4,0},{-2.828,2.828,0},
{-4,0,0},{-2.828,-2.828,0},{0,-4,0},{2.828,-2.828,0},
```

```
{4,0,0},{2.828,2.828,0},{0,4,0},{2.828,2.828,0}};//控制点

GLfloat color[12][3]={{1.0,0.0,0.0},{1.0,1.0,0.0},{0.0,1.0,0.0},{-1.0,1.0,0.0},
{-1.0,0.0,0.0},{-1.0,-1.0,0.0},{0.0,-1.0,0.0},{1.0,-1.0,0.0},
{1.0,0.0,0.0},{1.0,1.0,0.0},{0.0,1.0,0.0},{1.0,1.0,0.0}};

GLfloat knots[15] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};

void myInit(void)
{
glClearColor(1.0,1.0,1.0,0.0);//设置背景色
theNurb = gluNewNurbsRenderer();//创建 NURBS 对象 theNurb
gluNurbsProperty(theNurb,GLU_SAMPLING_TOLERANCE,10);
}

/*绘制曲线*/
void myDisplay(void)
{
int i;

glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glColor3f(0.0,0.0,0.0);

glLineWidth(3.0);

/*绘制曲线*/
gluBeginCurve(theNurb);
gluNurbsCurve(theNurb,15,knots,3,&ctrlpoints[0][0],3,GL_MAP1_VERTEX_3);
gluNurbsCurve(theNurb,15,knots,3,&ctrlpoints[0][0],3,GL_MAP1_COLOR_4);
gluEndCurve(theNurb);

/*绘制点*/
glColor3f(1.0,0.0,0.0);
glPointSize(5.0);
glBegin(GL_POINTS);
for(i = 0;i < 8;i++)
glVertex2fv(&ctrlpoints[i][0]);
glEnd();

glutSwapBuffers();
}

void myReshape(GLsizei w,GLsizei h)
{
```

```
glViewport(0,0,w,h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();

if(w <=h)
glOrtho(-10.0,10.0,-10.0*(GLfloat)h/(GLfloat)w,10.0*(GLfloat)h/(GLfloat)w,-10.0,10.0);
else
glOrtho(-10.0*(GLfloat)w/(GLfloat)h,10.0*(GLfloat)w/(GLfloat)h,-10.0,10.0,-10.0,10.0);

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(0.0,0.0,-9.0);
}

int main(int argc,char ** argv)
{
/*初始化*/
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
glutInitWindowSize(600,400);
glutInitWindowPosition(200,200);

/*创建窗口*/
glutCreateWindow("NURBS curve");

/*绘制与显示*/
myInit();

glutReshapeFunc(myReshape);
glutDisplayFunc(myDisplay);

glutMainLoop();
return(0);
}
```

·gluBeginCurve，gluEndCurve 限定 NURBS 曲面。返回值均为 void，参数均为 GLUnurbsObj* nobj，为指向 NURBS 对象的指针。

·void gluNurbsCurve（GLUnurbsObj *nobj， GLint nknots， GLfloat *knot， Glint stride，GLfloat *ctlarray， GLint order，GLenum type）定义曲线形状。

nobj 指向 NURBS 对象的指针。

nknots 节点数，节点数等于控制点数加上阶数。

knot nknots 数组非递减节点值。

stride 相邻控制点的偏移量。

Ctlarry 指向 NURBS 的控制点数组的指针。

order NURBS 曲线的阶数，阶数比维数大 1。

type 曲面类型。

图十四：NURBS 曲线

OpenGL 编程轻松入门之二次几何体

2006-05-22 10:10 作者： 黄燕 出处： 天极开发 责任编辑：方舟

这一章我们讲一下二次几何物体的绘制。二次几何物体的绘制有几种不同的方式，在本例中可以看出不同的绘制方式的不同效果，如图十五所示。

例 13：本例使用 GLU 库函数绘制了四个几何物体，分别为圆柱体、球体、圆盘和部分圆盘。

```
#include <windows.h>
#include <GL/glut.h>
#include <math.h>

/*声明四个二次曲面物体*/
GLUquadricObj *quadObj1;
GLUquadricObj *quadObj2;
GLUquadricObj *quadObj3;
GLUquadricObj *quadObj4;

static float light_ambient[] = {0.1,0.1,0.1,1.0};
static float light_diffuse[] = {0.5,1.0,1.0,1.0};
static float light_position[] = {90.0,90.0,150.0,0.0};
```

```
static float front_mat_shininess[] = {60.0};
static float front_mat_specular[] = {0.2,0.2,0.2,1.0};
static float front_mat_diffuse[] = {0.5,0.5,0.28,1.0};

static float back_mat_shininess[] = {60.0};
static float back_mat_specular[] = {0.5,0.5,0.2,1.0};
static float back_mat_diffuse[] = {1.0,0.9,0.2,1.0};

static float Imodel_ambient[] = {1.0,1.0,1.0,1.0};
static float Imodel_twoside[] = {GL_TRUE};
static float Imodel_oneside[] = {GL_FALSE};

void myInit(void)
{
/*设置背景色*/
glClearColor(1.0,1.0,1.0,1.0);

glEnable(GL_DEPTH_TEST);
glDepthFunc(GL_LEQUAL);

/*设置光照*/
glLightfv(GL_LIGHT0,GL_AMBIENT,light_ambient);
glLightfv(GL_LIGHT0,GL_DIFFUSE,light_diffuse);
glLightfv(GL_LIGHT0,GL_POSITION,light_position);

/*设置材质*/
glMaterialfv(GL_FRONT,GL_DIFFUSE,front_mat_diffuse);
glMaterialfv(GL_FRONT,GL_SPECULAR,front_mat_specular);
glMaterialfv(GL_FRONT,GL_SHININESS,front_mat_shininess);

glMaterialfv(GL_BACK,GL_DIFFUSE,back_mat_diffuse);
glMaterialfv(GL_BACK,GL_SPECULAR,back_mat_specular);
glMaterialfv(GL_BACK,GL_SHININESS,back_mat_shininess);

/*设置光照模型参数*/
glLightModelfv(GL_LIGHT_MODEL_AMBIENT,Imodel_ambient);
glLightModelfv(GL_LIGHT_MODEL_TWO_SIDE,Imodel_twoside);

/*激活关照*/
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glShadeModel(GL_SMOOTH);
}
```

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

/*创建四个二次曲面物体*/
quadObj1 = gluNewQuadric();
quadObj2 = gluNewQuadric();
quadObj3 = gluNewQuadric();
quadObj4 = gluNewQuadric();

/*绘制一个圆柱体*/
glPushMatrix();
gluQuadricDrawStyle(quadObj1,GLU_FILL);
gluQuadricNormals(quadObj1,GL_FLAT);
gluQuadricOrientation(quadObj1,GLU_INSIDE);
gluQuadricTexture(quadObj1,GL_TRUE);

glColor3f(1.0,1.0,0.0);
glRotatef(30,1.0,0.0,0.0);
glRotatef(40,0.0,1.0,0.0);
gluCylinder(quadObj1,2.0,2.0,9.0,20.0,8.0);
glPopMatrix();

/*绘制一个球体*/
glPushMatrix();
gluQuadricDrawStyle(quadObj2,GLU_SILHOUETTE);
glTranslatef(-5.0,-1.0,0.0);
gluSphere(quadObj2,3.0,20.0,20.0);
glPopMatrix();

/*绘制一个圆盘*/
glPushMatrix();
gluQuadricDrawStyle(quadObj3,GLU_LINE);
glTranslatef(-2.0,4.0,0.0);
gluDisk(quadObj3,2.0,5.0,15.0,10.0);
glPopMatrix();

/*绘制一个部分圆盘*/
glPushMatrix();
gluQuadricDrawStyle(quadObj4,GLU_POINT);
glTranslatef(-3.0,-7.0,0.0);
gluPartialDisk(quadObj4,2.0,5.0,15.0,10.0,10.0,100.0);
glPopMatrix();
```

```
/*删除四个二次曲面物体对象*/
gluDeleteQuadric(quadObj1);
gluDeleteQuadric(quadObj2);
gluDeleteQuadric(quadObj3);
gluDeleteQuadric(quadObj4);

glFlush();
}

void myReshape(int w,int h)
{
glViewport(0,0,(GLsizei)w,(GLsizei)h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(45.0,(GLfloat)w/(GLfloat)h,1.0,50.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(0.0,0.0,-25.0);
}

int main(int argc,char ** argv)
{
/*初始化*/
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
glutInitWindowSize(400,400);
glutInitWindowPosition(100,100);

/*创建窗口*/
glutCreateWindow(" DRAW QUADRIC OBJECTS ");

/*绘制与显示*/
myInit();
glutReshapeFunc(myReshape);
glutDisplayFunc(myDisplay);

glutMainLoop();
return 0;
}
```

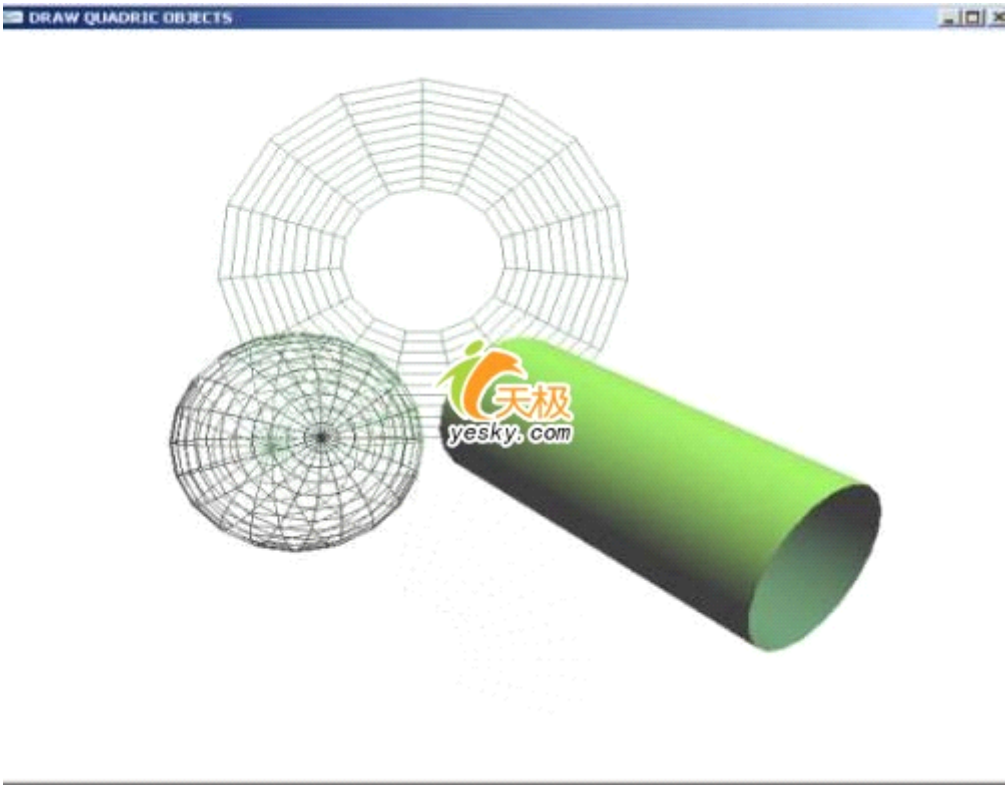
·gluNewQuadric 创建一个二次对象。这个函数创建并返回一个指向新的二次对象的指针。当调用二次描述和控制函数是指向这个对象。如果返回值为 0 则表明没有足够的空间分配给该对象。

·glQuadricDrawStyle 函数指定二次对象的绘制方式。本例中圆柱体的绘制方式为 GLU_FILL。含义为用多边形原绘制本二次对象，多边形的绘制方式为逆时针。球体的绘制

方式为 GL_SILHOUETTE,即除边界外用一系列线来绘制二次对象。圆盘的绘制方式为 GL_LINE,即用一系列线来绘制二次对象。部分圆盘的绘制方式为 GL_POINT,即用一系列点来绘制二次对象。

·glQuadricNormals，指定二次对象使用的法向量类型。

·glQuadricOrientation，指定二次对象的内面或外面方位。GLU_OUTSIDE 为缺省值，表明使用指向内面的法相量绘出二次对象,GLU_INSIDE 表明使用指向外面的法相量绘出二次对象。



图十五：二次几何体

·glQuadricTexture 指定二次对象是否使用纹理。GL_FALSE 为缺省值。

·void gluCylinder（GLUquadricObj *qobj，GLdouble baseRadius，GLdouble topRadius，GLdouble height，Glint slices，Glint stacks）绘制一个圆柱体。

qobj 指明是哪个二次对象。

baseRadius 圆柱体在 z=0 时的半径。

topRadius 圆柱体在 z=height 时的半径。

height 圆柱体的高。

slices 围绕着 z 轴分片的个数。

stacks 顺着 z 轴分片的个数。stacks 和 slices 垂直。

·void gluSphere（GLUquadricObj *qobj，GLdouble radius，Glint slices，Glint stacks）绘制一个球形。

qobj 指明是哪个二次对象。

radius 球体半径。

slices 围绕着 z 轴分片的个数。

stacks 顺着 z 轴分片的个数。

·void gluDisk（GLUquadricObj *qobj，GLdouble innerRadius，GLdouble outerRadius，Glint slices，Glint loops）绘制一个圆盘。

qobj 指明是哪个二次对象。

innerRadius 圆盘的内部半径，可能为 0。

outerRadius 圆盘的外部半径。

slices 围绕着 z 轴分片的个数。

loops 圆盘同心圆个数。

·void gluPartialDisk（GLUquadricObj *qobj，GLdouble innerRadius，GLdouble outerRadius，Glint slices，Glint loops，GLdouble startAngle，GLdouble sweepAngle）绘制一个圆盘的一部分。

startAngle 起始角度，单位为度。

sweepAngle 扫描角，单位为度。

OpenGL 编程轻松入门之动画制作

到目前为止我们所做的图形全部都是静止的。而 OpenGL 的是一个可以制作大型 3D 图形、动画的工具。下面我们做一个可以旋转的立方体。

例：一个旋转的立方体

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

GLfloat X = 10.0f;
GLfloat Y = 1.0f;
GLfloat Z = 5.0f;

void myDisplay(void)
{
GLfloat diffuse[] = {0.7,0.7,0.0,1.0};
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glColor3f(1.0,1.0,1.0);
/*绘制立方体*/
glPushMatrix();
glRotatef(X,1.0,0.0,0.0);
glRotatef(Y,0.0,1.0,0.0);
glRotatef(Z,0.0,0.0,1.0);
glMaterialfv(GL_FRONT,GL_DIFFUSE,diffuse);
glutSolidCube(1.0);
glPopMatrix();
glFlush();
glutSwapBuffers();//交换当前窗口使用层的缓存
}
```

```
void myReshape(int w,int h)
{
glViewport(0,0,w,h);

glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(-1.5,1.5,-1.5,1.5);
if(w <= h)
glOrtho(-2.25,2.25,-2.25*h/w,2.25*h/w,-10.0,10.0);
else
glOrtho(-2.25*w/h,2.25*w/h,-2.25,2.25,-10.0,10.0);

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
}

void myAnimate(void)
{
X += 1.0;
Y += 1.0;
Z += 1.0；

glutPostRedisplay();//标记当前窗口需要重新绘制

}

int main(int argc,char ** argv)
{
/*初始化*/
glutInit(&argc,argv);

glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
glutInitWindowSize(400,400);
glutInitWindowPosition(200,200);

/*创建窗口*/
glutCreateWindow("color");

/*绘制与显示*/
glutReshapeFunc(myReshape);

glutDisplayFunc(myDisplay);
glutIdleFunc(myAnimate);//设置全局空闲回调函数
```

```
glutMainLoop();

return(0);
}
```

myDisplay 函数中有一个 glutSwapBuffers()函数。此函数交换当前窗口使用层的缓存，它将后台缓存中的内容交换到前台缓存中，该函数执行的结果直到显示器垂直回行扫描后才看得到。必须使用双缓存结构，否则此函数不起任何作用。

myAanimate 函数中 glutPostRedisplay()函数标记当前窗口需要重新绘制。在 glutMainLoop 函数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绘该窗口的图像层。

在 main 函数中 glutInitDisplayMode 中为 GLUT_DOUBLE，而我们以前的很多例子为 GLUT_SINGLE。main 函数中还调用了 glutIdleFunc，此函数设置全局空闲回调函数。，从而使 GLUT 程序可以执行后台任务或连续动画。