**双缓冲只需要交换两块内存的指针而已，因此效率极高**

#include "stdafx.h"

#include <iostream>

#include <gl/glut.h>

using namespace std;

const GLsizei WIDTH = 480;

const GLsizei HEIGHT = 800;

const GLfloat WIDTH\_HEIGHT\_RATE = (GLfloat)WIDTH / (GLfloat)HEIGHT;

void ChangeSize(GLsizei width, GLsizei height);

void RenderScene();

void setRC();

int \_tmain(int argc, char\* argv[])

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGBA | GLUT\_DOUBLE);

glutCreateWindow("gltest");

glutInitWindowSize(WIDTH, HEIGHT);

setRC();

glutDisplayFunc(RenderScene);

glutReshapeFunc(ChangeSize);

glutMainLoop();

system("pause");

return 0;

}

void setRC()

{

glClearColor(0.0f, 0.0f, 1.0f, 1.0f);

glColor3f(1.0f, 0.0f, 0.0f);

}

void RenderScene()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glRectf(-25.0f, 25.0f, 25.0f, -25.0f);

glutSwapBuffers();

cout << "renderScene" << endl;

}

void ChangeSize(GLsizei width, GLsizei height)

{

if(height == 0)

height == 1;

GLfloat rate = (GLfloat)width / (WIDTH\_HEIGHT\_RATE \* height);

//设置视口

glViewport(0, 0, width, height);

//设置裁剪区域

glMatrixMode(GL\_PROJECTION);//设置当前矩阵为投影矩类型¨ª

glLoadIdentity();//设置单位矩阵

if(width < height \* WIDTH\_HEIGHT\_RATE)//变窄-

{

glOrtho(-100.0f, 100.0f, (-100.0f / WIDTH\_HEIGHT\_RATE) / rate, (100.0f / WIDTH\_HEIGHT\_RATE) / rate, 1.0f, -1.0f);

}

else

{

glOrtho(-100.0f \* WIDTH\_HEIGHT\_RATE \* rate, 100.0f \* WIDTH\_HEIGHT\_RATE \* rate, -100.0f, 100.0f, 1.0f, -1.0f);

}

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

cout << "changeSize" << endl;

}

注意：在上面的程序中，如果没有添加glutSwapBuffers,那么就会什么都画不出，因为对于双缓冲来说图像总是先在内存中画出然后在屏幕上显示

**关于模型变换（平移、旋转、放缩）**

沿着X、Y与Z轴移动，但是在glTranslatef(x,y,z)中，当移动时不是相对屏幕中心移动，而是相对于当前所在的屏幕位置，其作用就是讲你绘制的点坐标的远在在当前原点的基础上平移一个(x,y,z)向量

例如下面的例子：

const GLsizei WIDTH = 480;

const GLsizei HEIGHT = 800;

const GLfloat WIDTH\_HEIGHT\_RATE = (GLfloat)WIDTH / (GLfloat)HEIGHT;

void ChangeSize(GLsizei width, GLsizei height);

void RenderScene();

void setRC();

void DrawTriangles();

void DrawSecondTriangles();

void DrawThirdTriangles();

void DrawOriginPoint();

int \_tmain(int argc, char\* argv[])

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGBA | GLUT\_DOUBLE);

glutCreateWindow("gltest");

glutInitWindowSize(WIDTH, HEIGHT);

setRC();

glutDisplayFunc(RenderScene);

glutReshapeFunc(ChangeSize);

glutMainLoop();

system("pause");

return 0;

}

void setRC()

{

glClearColor(0.0f, 0.0f, 1.0f, 1.0f);

glColor3f(1.0f, 0.0f, 0.0f);

}

void RenderScene()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

glTranslatef(50.0f, 0.0f, 0.0f);

DrawOriginPoint();

DrawTriangles();

DrawSecondTriangles();

DrawThirdTriangles();

glutSwapBuffers();

cout << "renderScene" << endl;

}

void DrawTriangles()

{

glBegin(GL\_TRIANGLES);

glVertex3f(0.0f, 0.0f, 0.0f);

glVertex3f(0.0f, 25.0f, 0.0f);

glVertex3f(25.0f, 0.0f, 0.0f);

glEnd();

}

void DrawThirdTriangles()

{

glBegin(GL\_TRIANGLES);

glVertex3f(0.0f, -50.0f, 0.0f);

glVertex3f(0.0f, -25.0f, 0.0f);

glVertex3f(25.0f, -50.0f, 0.0f);

glEnd();

}

void DrawSecondTriangles()

{

glPushMatrix();

glLoadIdentity();

glBegin(GL\_TRIANGLES);

glVertex3f(0.0f, 0.0f, 0.0f);

glVertex3f(0.0f, 25.0f, 0.0f);

glVertex3f(25.0f, 0.0f, 0.0f);

glEnd();

glPopMatrix();

}

void DrawOriginPoint()

{

glEnable(GL\_POINT\_SMOOTH);

glPointSize(5.0f);

glBegin(GL\_POINTS);

glVertex3f(0.0f, 0.0f, 0.0f);

glEnd();

}

void ChangeSize(GLsizei width, GLsizei height)

{

if(height == 0)

height == 1;

GLfloat rate = (GLfloat)width / (WIDTH\_HEIGHT\_RATE \* height);

//设置视口

glViewport(0, 0, width, height);

//设置裁剪区域

glMatrixMode(GL\_PROJECTION);//设置当前矩阵为投影矩阵类型

glLoadIdentity();//设置单位矩阵

if(width < height \* WIDTH\_HEIGHT\_RATE)//变窄-

{

glOrtho(-100.0f, 100.0f, (-100.0f / WIDTH\_HEIGHT\_RATE) / rate, (100.0f / WIDTH\_HEIGHT\_RATE) / rate, 1.0f, -1.0f);

}

else

{

glOrtho(-100.0f \* WIDTH\_HEIGHT\_RATE \* rate, 100.0f \* WIDTH\_HEIGHT\_RATE \* rate, -100.0f, 100.0f, 1.0f, -1.0f);

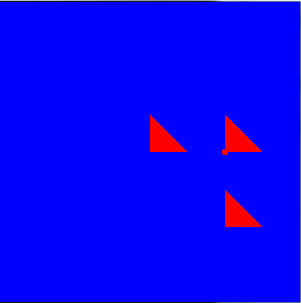
}

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

cout << "changeSize" << endl;

}



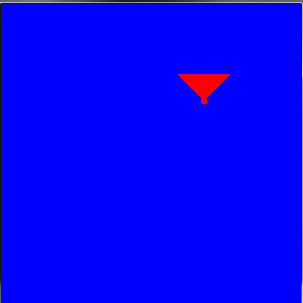
在上面的例子中，需要注意的是glPointSize需要放在glBegin外面，同时可以看到glLoadIdentity函数只是改变了当前栈顶的矩阵，并没有将栈底的矩阵改变（参照第三个矩形的绘制，并没有在原点绘制，而是相对于glTranslatef函数绘制的），同时要注意到在glBegin前面的模型操作都是通过计算变成了一个矩阵放在栈顶，如果需要保存某一个模型操作则需要自己手动进行glPushMatrix操作

**移动与旋转的顺序问题：**

先旋转在移动：

glRotatef(45, 0, 0, 1);

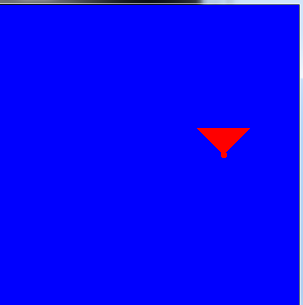
glTranslatef(50.0f, 0.0f, 0.0f);



先移动在旋转：

glTranslatef(50.0f, 0.0f, 0.0f);

glRotatef(45, 0, 0, 1);



**OpenGL中的光照**

OpenGL在处理光照时采用一种近视：把光照系统分为三部分，分别是光源、材质和光照环境。光源就是光的来源，可以是太阳或电灯等，材质是指接受光照的各种物体的表面，由于物体如何反射光线只由物体表面决定（OpenGL没有考虑光的折射），材料特点就决定了物体反射光线的特点，光照环境是指一些额外的参数，它们将影响最终的光照画面

通常，为了实现光照效果，需要在代码中为每个顶点指定其法线向量，指定法线向量的方式与指定颜色的方式有雷同之处，在指定颜色时，只需要指定每个顶点的颜色，OpenGL就可以自行计算顶点之间的其他点的颜色，并且颜色一旦被指定，除非再指定新的颜色，否则以后指定的所有顶点都将以这一向量作为自己的颜色，在指定法线时，只需要指定每个顶点的法线向量，OpenGL会自行计算顶点之间的其它法线向量，使用glColor\*可以指定颜色，使用glNormal\*可以指定法线向量

**注意：**使用glTranslate\*函数或者glRotate\*函数可以改变物体的外观，但法线向量并不会随之改变。然而，使用glScale\*函数，对每一坐标轴进行不同程度的缩放，很有可能导致法线向量的不正确，虽然OpenGL提供了一些措施来修正这一问题，但由此也带来了各种开销。因此，在使用了法线向量的场合，应尽量避免使用glScale\*函数。即使使用，也最好保证各坐标轴进行等比例缩放。

源类型是：点光源、无穷远光源、方向光源和环境光。  
点光源：光线从光源点向四面八方发散，发光的恒星（如太阳）、发光的灯泡一般使用该光源模型模拟，是最简单的光源。  
无穷远光源：所有的光线都平行的从一个方向过来，当发光体（如太阳）离渲染的场景很远可以认为是无穷远时，一般使用该光源模型进行模拟。  
方向光源：光线沿着一个方向在特定角度范围内逐渐发散开。现实世界中的车灯，手电筒一般使用该光源模型进行模拟。  
环境光源：光线从各个地方以各个角度投射到场景中所有物体表面，找不到光源的确切位置。现实世界中不存在这样的光源，一般使用该光源模型来模拟点光源、无穷远光源、方向光源在物体表面经过许多次反射后的情况，环境光源照亮所有物体的所有面。

在OpenGL中漫反射部分的光线与镜面反射部分的光线是分开计算的，然后将分开计算的交过进行叠加

材质的属性包括：镜面反射颜色，漫反射颜色，环境光颜色，光洁度，自发光颜色

在OpenGL中可以使用下面的代码来设置材质属性：  
GLfloat planet\_ambient[] = { 0.01 , 0.01 , 0.01 , 1.0 };  
GLfloat planet\_diffuse[] = { 0.7 , 0.7 , 0.7 , 1.0 }  
;  
glMaterialfv(GL\_FRONT , GL\_AMBIENT ,planet\_ambient);  
glMaterialfv(GL\_FRONT , GL\_DIFFUSE ,planet\_diffuse);

openGL确定某个顶点颜色可以分为两种情况：开启光照渲染和关闭关照渲染。当关闭光照渲染时，顶点的颜色由openGL状态机绘制该物体时的颜色确定；当开启光照渲染时，绘制该物体时openGL状态机的颜色将对该物体上顶点的颜色没有任何影响，此时物体顶点的颜色由物体材质、光照叠加效果以及物体表面的纹理贴图（如果有纹理贴图的话）共同决定