第二天 OpenGL 基础渲染

学习内容

- 使用GLBatch 帮助类传递几何图形
- 执行深度测试、背面消除
- 绘制透明或者混合几何图形
- 绘制抗锯齿点、线和多边形

一、点连接

第一次学习任何计算机系统中绘制任何类型的2D图形时，大多数可能从绘制像素开始。像素是计算机屏幕上显示的最小元素。以下是最简单额计算机图形：在计算机屏幕绘制一个点，并将它设置一个特定的颜色。在这个简单的基础上慢慢学会创建线、多边形、圆和其他性质和图形。

但是！使用OpenGL在计算机屏幕上进行绘图则完全不同。
我们不关物理屏幕坐标和像素，关注的是视景体中的位置坐标。将这些点、线和三角形从创建3D空间投影到计算机屏幕上的2D图形则是着色器程序和光栅化硬件所要完成的工作。

1.1 点和线

第一节课我们也讲过OpenGL 的几何图元。今天我们将从更底层更基础的角度来详细学习OpenGL 图元渲染。
点，是最简单的图像。每个特定的顶点在屏幕上都仅仅是一个单独的点。默认的情况下，点的大小是一个像素的大小。

修改点大小的方法：

```
//1.最简单也是最常用的 4.0f,表示点的大小
   glPointSize(4.0f);

//2.设置点的大小范围和点与点之间的间隔
GLfloat sizes[2] = {2.0f,4.0f};
GLfloat step = 1.0f;
```

```
//获取点大小范围和最小步长
glGetFloatv(GL_POINT_SIZE_RANGE,sizes);
glGetFloatv(GL_POINT_GRAULARITY,&step);

//3.通过使用程序点大小模式来设置点大小
glEnable(GL_PROGRAM_POINT_SIZE);

//这种模式下允许我们通过编程在顶点着色器或几何着色器中设置点大小。着色器内建变量：gl_PointSize，并且可以在着色器源码直接写
 gl_PointSize = 5.0;
```

1.2 线

比点更进一步的就是独立线段了。一个线段就是2个顶点之间绘制的。
默认情况下，线段的宽度是一个像素。改变线段唯一的方式通过：

```
//1.设置独立线段宽度为 2.5f;
glLineWidth(2.5f);
```

1.3 线带

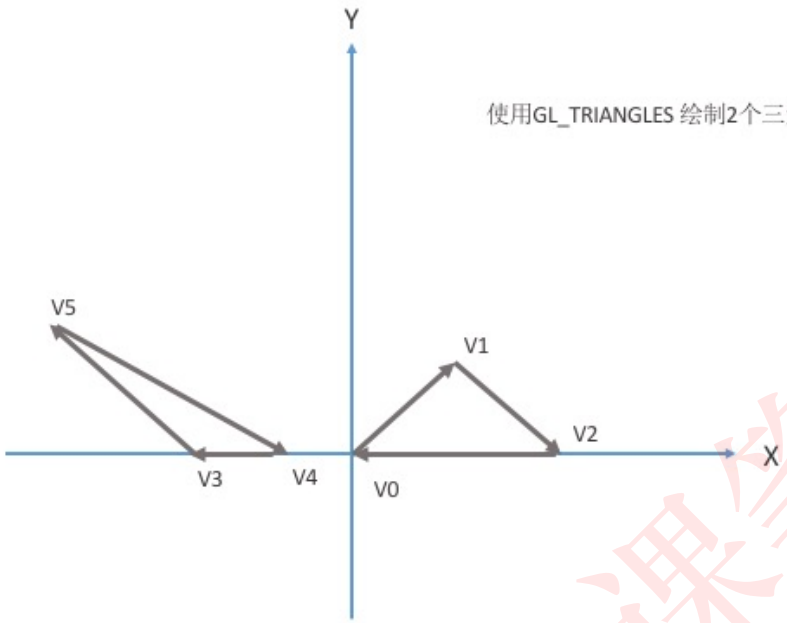
线段连续从一个顶点到下一个顶点绘制的线段，以形成一个真正链接的点的线段。
（为了把图形连接起来，每个连接的顶点会被选定2次。一次作为线段的终点、一次作为下一条线段的起点）

1.4 线环

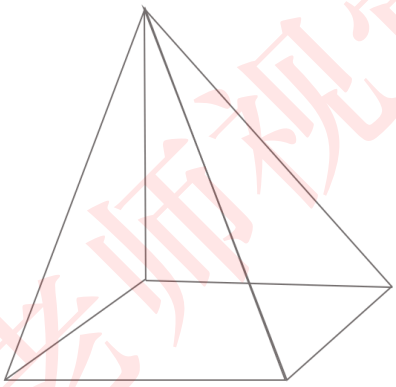
线环是线带的一种简单拓展。在线带的基础上额外增加一条将线带闭合的。

1.5 绘制三角形

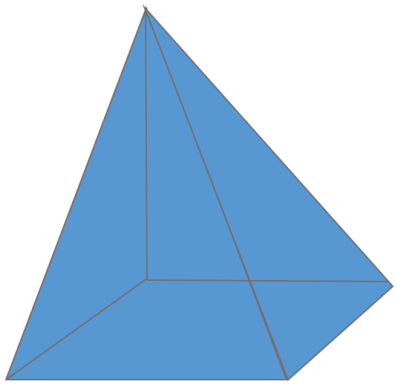
最简单的实体多边形就是三角形，它只有3个边。光栅化硬件最欢迎三角形。并且现在OpenGL已经是OpenGL中支持的唯一一种多边形。每3个顶点定义一个新的三角形。



在今天的课程中，我们不仅绘制一个三角形，而是绘制4个三角形组成金字塔形的三角形。我们可以使用方向键来旋转金字塔，从不同角度进行观察。但是这个金字塔木有底面，所以我们可以看到它的底部。



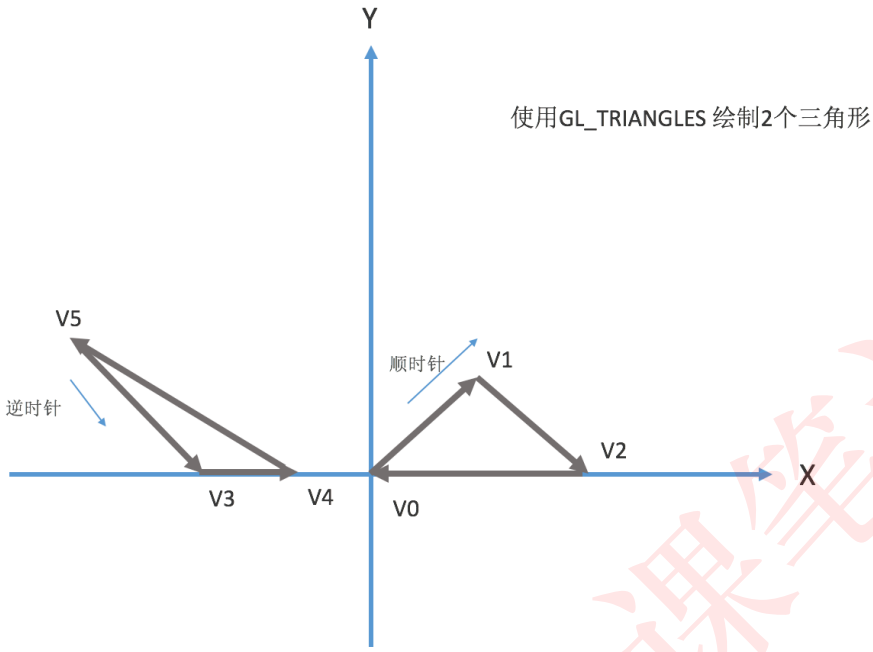
四边金字塔



四边金字塔

1.6 环绕

将顺时针方向绘制的三角形用逆时针的方式绘制。
在绘制第一个三角形时，线条是按照从V0-V1，再到V2。最后再回到V0的一个闭合三角形。 这个是沿着顶点顺时针方向。
这种顺序与方向结合来指定顶点的方式称为环绕。



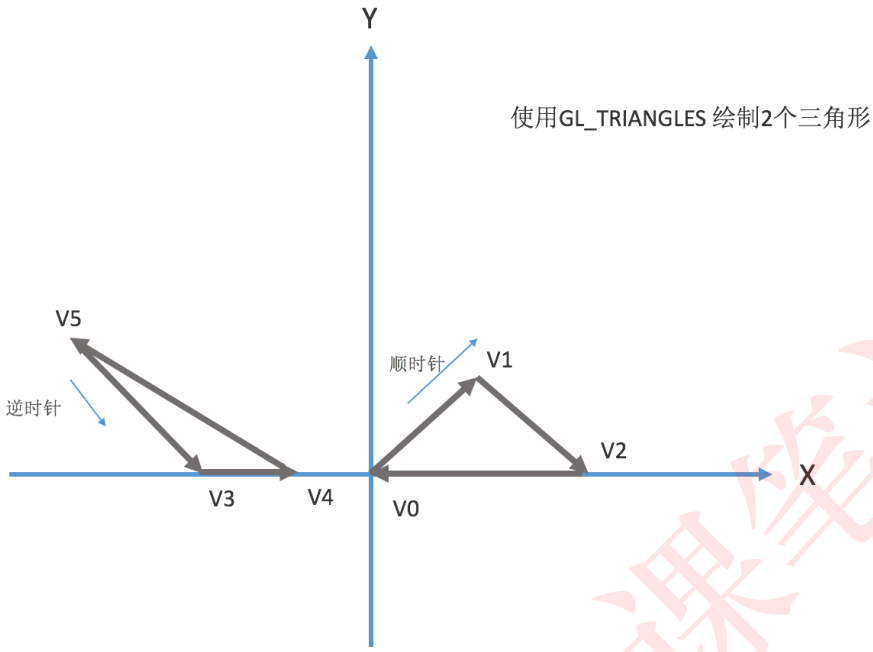
上图的2个三角形的缠绕方向完全相反。

- 在默认的情况下，OpenGL认为具有逆时针方向环绕的多边形是正面的。而而右侧的顺时针方向三角形是三角形的背面。

> **为什么会认为这个问题会很重要了?**
> 因为我们常常希望为一个多边形的正面和背面分别设置不同的物理特征。我们可以完全隐藏一个多边形的背面，或者给它设置一种不同的颜色和反射属性。纹理图像在背面三角形中也是相反的。在一个场景中，使所有的多边形保持环绕方向的一致，并使用正面多边形来绘制所有实心物体的表面是非常重要的。

```
//定义前向和背向的多变形:
glFrontFace(mode)
参数: GL_CW | GL_CCW
GL_CCW: 表示传入的mode会选择逆时针为前向
GL_CW: 表示顺时针为前向。
默认: GL_CCW。逆向时针为前向。
```

1.7 三角地带



对于很多表面和形状来说，我们可能需要绘制几个相连的三角形。我们可以使用 `GL_TRIANGLE_STRIP` 图元绘制一串相连的三角形。从而节省大量的时间。

使用三角带而不是分别指定每个三角形，这样做的优点：

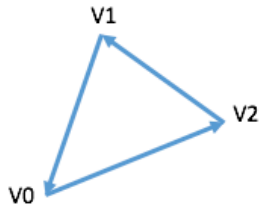
1.用前3个顶点指定第1个三角形之后，对于接下来的每一个三角形，只需要再指定1个顶点。需要绘制大量的三角形时，采用这种方法可以节省大量的程序代码和数据存储空间。

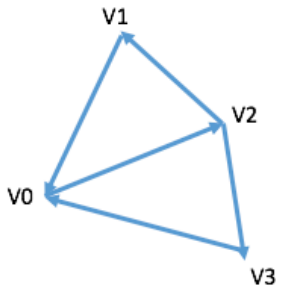
2.提供运算性能和节省带宽。更少的顶点意味着数据从内存传输到图形卡的速度更快，并且顶点着色器需要处理的次数也更少了。

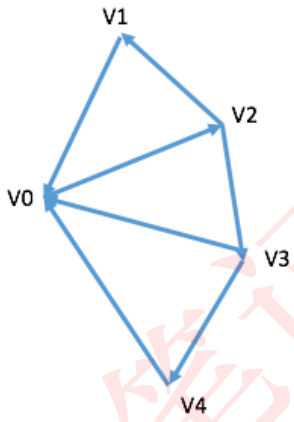
1.8 三角形扇

除了三角形带之外，还可以使用 `GL_TRIANGLE_FAN` 创建一组围绕一个中心点的相连三角形。通过4个顶点所产生的包括3个三角形的三角形扇。

第一个顶点v0构建了扇形的原点，用前3个顶点指定了最初的三角形之后，后续的每个顶点都和原点（V0）以及之前紧挨着它的那个顶点(Vn-1)形成接下来的三角形。







二、简单的批次容器

GLTools 库中包含额一个简单的容器类，叫做GBatch。这个类可以作为7种图元的简单批次容器使用。而且它知道在使用GL_ShaderManager 支持的任意存储着色器时如何对图元进行渲染。

```
void GLBatch::Begain(GLeunm primitive,GLuint nVerts,GLuint nTe
xttureUnints = 0);

参数1：图元
参数2：顶点数
参数3：一组或者2组纹理坐标（可选）

//复制表面法线
void GLBatch::CopyNormalDataf(GLfloat *vNorms);

//复制颜色
void GLBatch::CopyColorData4f(GLfloat *vColors);

//复制纹理坐标
void GLBatch::CopyTexCoordData2f(GLFloat *vTextCoords,GLuint u
iTextureLayer);
```

```
//结束绘制
void GLBatch::End(void);
```

三、实例

```
/*
 课程名称：OpenGL 视觉班--第二次课
 案例名称：OpenGL demo1
 实现功能：
    点击屏幕，将固定位置上的顶点数据以6种不同形态展示！
 */


#include "GLTools.h"
#include "GLMatrixStack.h"
#include "GLFrame.h"
#include "GLFrustum.h"
#include "GLBatch.h"
#include "GLGeometryTransform.h"

#include <math.h>
#ifdef __APPLE__
#include <glut/glut.h>
#else
#define FREEGLUT_STATIC
#include <GL/glut.h>
#endif

/*
 GLMatrixStack 变化管线使用矩阵堆栈

 GLMatrixStack 构造函数允许指定堆栈的最大深度、默认的堆栈深度为64.这个矩阵堆在初始化时已经在堆栈中包含了单位矩阵。
 GLMatrixStack::GLMatrixStack(int iStackDepth = 64);
```

```
 //通过调用顶部载入这个单位矩阵
 void GLMatrixStack::LoadIndentiy(void);

 //在堆栈顶部载入任何矩阵
 void GLMatrixStack::LoadMatrix(const M3DMatrix44f m);
 */
// 各种需要的类
GLShaderManager     shaderManager;
GLMatrixStack       modelViewMatrix;
GLMatrixStack       projectionMatrix;
GLFrame             cameraFrame;
GLFrame             objectFrame;
//投影矩阵
GLFrustum           viewFrustum;

//容器类（7种不同的图元对应7种容器对象）
GLBatch             pointBatch;
GLBatch             lineBatch;
GLBatch             lineStripBatch;
GLBatch             lineLoopBatch;
GLBatch             triangleBatch;
GLBatch             triangleStripBatch;
GLBatch             triangleFanBatch;

//几何变换的管道
GLGeometryTransform transformPipeline;
M3DMatrix44f        shadowMatrix;


GLfloat vGreen[] = { 0.0f, 1.0f, 0.0f, 1.0f };
GLfloat vBlack[] = { 0.0f, 0.0f, 0.0f, 1.0f };


// 跟踪效果步骤
```

```
int nStep = 0;



// 此函数在呈现上下文中进行任何必要的初始化。.
// 这是第一次做任何与opengl相关的任务。
void SetupRC()
{
    // 灰色的背景
    glClearColor(0.7f, 0.7f, 0.7f, 1.0f );

    shaderManager.InitializeStockShaders();

    glEnable(GL_DEPTH_TEST);

    //设置变换管线以使用两个矩阵堆栈
    transformPipeline.SetMatrixStacks(modelViewMatrix, project
ionMatrix);

    cameraFrame.MoveForward(-15.0f);

    /*
     常见函数:
     void GLBatch::Begin(GLenum primitive,GLuint nVerts,GLuint
 nTextureUnits = 0);
      参数1: 表示使用的图元
      参数2: 顶点数
      参数3: 纹理坐标（可选）

     //负责顶点坐标
     void GLBatch::CopyVertexData3f(GLFloat *vNorms);

     //结束，表示已经完成数据复制工作
     void GLBatch::End(void);
```

```
 */
//定义一些点，类似佛罗里达州的形状。
GLfloat vCoast[24][3] = {
    {2.80, 1.20, 0.0 }, {2.0,  1.20, 0.0 },
    {2.0,  1.08, 0.0 },  {2.0,  1.08, 0.0 },
    {0.0,  0.80, 0.0 },  {-.32, 0.40, 0.0 },
    {-.48, 0.2, 0.0 },   {-.40, 0.0, 0.0 },
    {-.60, -.40, 0.0 },  {-.80, -.80, 0.0 },
    {-.80, -1.4, 0.0 },  {-.40, -1.60, 0.0 },
    {0.0, -1.20, 0.0 },  { .2, -.80, 0.0 },
    {.48, -.40, 0.0 },   {.52, -.20, 0.0 },
    {.48,  .20, 0.0 },   {.80,  .40, 0.0 },
    {1.20, .80, 0.0 },   {1.60, .60, 0.0 },
    {2.0, .60, 0.0 },    {2.2, .80, 0.0 },
    {2.40, 1.0, 0.0 },   {2.80, 1.0, 0.0 }};

//用点的形式--表示佛罗里达州的形状
pointBatch.Begin(GL_POINTS, 24);
pointBatch.CopyVertexData3f(vCoast);
pointBatch.End();

//通过线的形式--表示佛罗里达州的形状
lineBatch.Begin(GL_LINES, 24);
lineBatch.CopyVertexData3f(vCoast);
lineBatch.End();

//通过线段的形式--表示佛罗里达州的形状
lineStripBatch.Begin(GL_LINE_STRIP, 24);
lineStripBatch.CopyVertexData3f(vCoast);
lineStripBatch.End();

//通过线环的形式--表示佛罗里达州的形状
lineLoopBatch.Begin(GL_LINE_LOOP, 24);
lineLoopBatch.CopyVertexData3f(vCoast);
```

```
    lineLoopBatch.End();

    //通过三角形创建金字塔
    GLfloat vPyramid[12][3] = {
        -2.0f, 0.0f, -2.0f,
        2.0f, 0.0f, -2.0f,
        0.0f, 4.0f, 0.0f,

        2.0f, 0.0f, -2.0f,
        2.0f, 0.0f, 2.0f,
        0.0f, 4.0f, 0.0f,

        2.0f, 0.0f, 2.0f,
        -2.0f, 0.0f, 2.0f,
        0.0f, 4.0f, 0.0f,

        -2.0f, 0.0f, 2.0f,
        -2.0f, 0.0f, -2.0f,
        0.0f, 4.0f, 0.0f};

    //GL_TRIANGLES 每3个顶点定义一个新的三角形
    triangleBatch.Begin(GL_TRIANGLES, 12);
    triangleBatch.CopyVertexData3f(vPyramid);
    triangleBatch.End();


    // 三角形扇形--六边形
    GLfloat vPoints[100][3];    //超过我们需要的数组
    int nVerts = 0;
    //半径
    GLfloat r = 3.0f;

    //原点(x,y,z) = (0,0,0);
    vPoints[nVerts][0] = 0.0f;
    vPoints[nVerts][1] = 0.0f;
```

```
    vPoints[nVerts][2] = 0.0f;

    //M3D_2PI 就是2Pi 的意思，就一个圆的意思。 绘制圆形

    for(GLfloat angle = 0; angle < M3D_2PI; angle += M3D_2PI / 6.0f) {

        //数组下标自增（每自增1次就表示一个顶点）
        nVerts++;
        /*
         弧长=半径*角度,这里的角度是弧度制,不是平时的角度制
         既然知道了cos值,那么角度=arccos,求一个反三角函数就行了
         */
        //x点坐标 cos(angle) * 半径
        vPoints[nVerts][0] = float(cos(angle)) * r;
        //y点坐标 sin(angle) * 半径
        vPoints[nVerts][1] = float(sin(angle)) * r;
        //z点的坐标
        vPoints[nVerts][2] = -0.5f;
    }

    // 结束扇形 前面一共绘制7个顶点（包括圆心）
    printf("三角形扇形顶点数:%d\n",nVerts);
    //添加闭合的终点
    //课程添加演示：屏蔽177-180行代码，并把绘制节点改为7.则三角形扇形是无法闭合的。
    nVerts++;
    vPoints[nVerts][0] = r;
    vPoints[nVerts][1] = 0;
    vPoints[nVerts][2] = 0.0f;

    // 加载!
    //GL_TRIANGLE_FAN 以一个圆心为中心呈扇形排列，共用相邻顶点的一组三角形
    triangleFanBatch.Begin(GL_TRIANGLE_FAN, 8);
```

```
    triangleFanBatch.CopyVertexData3f(vPoints);
    triangleFanBatch.End();

    //三角形条带，一个小环或圆柱段
    //顶点下标
    int iCounter = 0;
    //半径
    GLfloat radius = 3.0f;
    //从0度~360度，以0.3弧度为步长
    for(GLfloat angle = 0.0f; angle <= (2.0f*M3D_PI); angle += 0.3f)
    {
        //或许圆形的顶点的X,Y
        GLfloat x = radius * sin(angle);
        GLfloat y = radius * cos(angle);

        //绘制2个三角形（他们的x,y顶点一样，只是z点不一样）
        vPoints[iCounter][0] = x;
        vPoints[iCounter][1] = y;
        vPoints[iCounter][2] = -0.5;
        iCounter++;

        vPoints[iCounter][0] = x;
        vPoints[iCounter][1] = y;
        vPoints[iCounter][2] = 0.5;
        iCounter++;
    }

    // 关闭循环
    printf("三角形带的顶点数：%d\n",iCounter);
    //结束循环，在循环位置生成2个三角形
    vPoints[iCounter][0] = vPoints[0][0];
    vPoints[iCounter][1] = vPoints[0][1];
    vPoints[iCounter][2] = -0.5;
    iCounter++;
```

```
    vPoints[iCounter][0] = vPoints[1][0];
    vPoints[iCounter][1] = vPoints[1][1];
    vPoints[iCounter][2] = 0.5;
    iCounter++;

    // GL_TRIANGLE_STRIP 共用一个条带（strip）上的顶点的一组三角形
    triangleStripBatch.Begin(GL_TRIANGLE_STRIP, iCounter);
    triangleStripBatch.CopyVertexData3f(vPoints);
    triangleStripBatch.End();
}



void DrawWireFramedBatch(GLBatch* pBatch)
{
    /* GLShaderManager 中的Uniform 值——平面着色器
     参数1：平面着色器
     参数2：运行为几何图形变换指定一个 4 * 4变换矩阵
          --transformPipeline 变换管线（指定了2个矩阵堆栈）
     参数3：颜色值
    */
    shaderManager.UseStockShader(GLT_SHADER_FLAT, transformPip
eline.GetModelViewProjectionMatrix(), vGreen);
    pBatch->Draw();

    /*
        glEnable(GLenum mode); 用于启用各种功能。功能由参数决定
        参数列表：http://blog.csdn.net/augusdi/article/details/2
3747081
        注意：glEnable() 不能写在glBegin() 和 glEnd()中间
        GL_POLYGON_OFFSET_LINE  根据函数glPolygonOffset的设置，启
用线的深度偏移
        GL_LINE_SMOOTH          执行后，过虑线点的锯齿
        GL_BLEND                启用颜色混合。例如实现半透明效果
```

```
        GL_DEPTH_TEST           启用深度测试 根据坐标的远近自动隐藏被遮住的图形（材料


        glDisable(GLenum mode); 用于关闭指定的功能 功能由参数决定

     */

    //画黑色边框
    glPolygonOffset(-1.0f, -1.0f);// 偏移深度，在同一位置要绘制填充和边线，会产生z冲突，所以要偏移
    glEnable(GL_POLYGON_OFFSET_LINE);

    // 画反锯齿，让黑边好看些
    glEnable(GL_LINE_SMOOTH);
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

    //绘制线框几何黑色版 三种模式，实心，边框，点，可以作用在正面，背面，或者两面
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    glLineWidth(2.5f);

    /* GLShaderManager 中的Uniform 值——平面着色器
     参数1：平面着色器
     参数2：运行为几何图形变换指定一个 4 * 4变换矩阵
         --transformPipeline.GetModelViewProjectionMatrix() 获取的
          GetMatrix函数就可以获得矩阵堆栈顶部的值
     参数3：颜色值（黑色）
     */
    shaderManager.UseStockShader(GLT_SHADER_FLAT, transformPipeline.GetModelViewProjectionMatrix(), vBlack);
    pBatch->Draw();
```

```
    // 复原原本的设置
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    glDisable(GL_POLYGON_OFFSET_LINE);
    glLineWidth(1.0f);
    glDisable(GL_BLEND);
    glDisable(GL_LINE_SMOOTH);
}



// 召唤场景
void RenderScene(void)
{
    // Clear the window with current clearing color
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STE
NCIL_BUFFER_BIT);

    //压栈
    modelViewMatrix.PushMatrix();
    M3DMatrix44f mCamera;
    cameraFrame.GetCameraMatrix(mCamera);

    //矩阵乘以矩阵堆栈的顶部矩阵，相乘的结果随后简存储在堆栈的顶部
    modelViewMatrix.MultMatrix(mCamera);

    M3DMatrix44f mObjectFrame;
    //只要使用 GetMatrix 函数就可以获取矩阵堆栈顶部的值，这个函数可以进
行2次重载。用来使用GLShaderManager 的使用。或者是获取顶部矩阵的顶点副本
数据
    objectFrame.GetMatrix(mObjectFrame);

    //矩阵乘以矩阵堆栈的顶部矩阵，相乘的结果随后简存储在堆栈的顶部
    modelViewMatrix.MultMatrix(mObjectFrame);

    /* GLShaderManager 中的Uniform 值——平面着色器
```

```
    参数1：平面着色器
    参数2：运行为几何图形变换指定一个 4 * 4变换矩阵
    --transformPipeline.GetModelViewProjectionMatrix() 获取的
    GetMatrix函数就可以获得矩阵堆栈顶部的值
    参数3：颜色值（黑色）
    */
    shaderManager.UseStockShader(GLT_SHADER_FLAT, transformPip
eline.GetModelViewProjectionMatrix(), vBlack);

    switch(nStep) {
        case 0:
            //设置点的大小
            glPointSize(4.0f);
            pointBatch.Draw();
            glPointSize(1.0f);
            break;
        case 1:
            //设置线的宽度
            glLineWidth(2.0f);
            lineBatch.Draw();
            glLineWidth(1.0f);
            break;
        case 2:
            glLineWidth(2.0f);
            lineStripBatch.Draw();
            glLineWidth(1.0f);
            break;
        case 3:
            glLineWidth(2.0f);
            lineLoopBatch.Draw();
            glLineWidth(1.0f);
            break;
        case 4:
            DrawWireFramedBatch(&triangleBatch);
            break;
```

```
        case 5:
            DrawWireFramedBatch(&triangleStripBatch);
            break;
        case 6:
            DrawWireFramedBatch(&triangleFanBatch);
            break;
    }

    //还原到以前的模型视图矩阵（单位矩阵）
    modelViewMatrix.PopMatrix();

    // 进行缓冲区交换
    glutSwapBuffers();
}


//特殊键位处理（上、下、左、右移动）
void SpecialKeys(int key, int x, int y)
{
    if(key == GLUT_KEY_UP)
        objectFrame.RotateWorld(m3dDegToRad(-5.0f), 1.0f, 0.0f
, 0.0f);

    if(key == GLUT_KEY_DOWN)
        objectFrame.RotateWorld(m3dDegToRad(5.0f), 1.0f, 0.0f,
 0.0f);

    if(key == GLUT_KEY_LEFT)
        objectFrame.RotateWorld(m3dDegToRad(-5.0f), 0.0f, 1.0f
, 0.0f);

    if(key == GLUT_KEY_RIGHT)
        objectFrame.RotateWorld(m3dDegToRad(5.0f), 0.0f, 1.0f,
 0.0f);
```

```
    glutPostRedisplay();
}



//根据空格次数。切换不同的“窗口名称”
void KeyPressFunc(unsigned char key, int x, int y)
{
    if(key == 32)
    {
        nStep++;

        if(nStep > 6)
            nStep = 0;
    }

    switch(nStep)
    {
        case 0:
            glutSetWindowTitle("GL_POINTS");
            break;
        case 1:
            glutSetWindowTitle("GL_LINES");
            break;
        case 2:
            glutSetWindowTitle("GL_LINE_STRIP");
            break;
        case 3:
            glutSetWindowTitle("GL_LINE_LOOP");
            break;
        case 4:
            glutSetWindowTitle("GL_TRIANGLES");
            break;
        case 5:
```

```
            glutSetWindowTitle("GL_TRIANGLE_STRIP");
            break;
        case 6:
            glutSetWindowTitle("GL_TRIANGLE_FAN");
            break;
    }

    glutPostRedisplay();
}


// 窗口已更改大小，或刚刚创建。无论哪种情况，我们都需要
// 使用窗口维度设置视口和投影矩阵.
void ChangeSize(int w, int h)
{
    glViewport(0, 0, w, h);
    //创建投影矩阵，并将它载入投影矩阵堆栈中
    viewFrustum.SetPerspective(35.0f, float(w) / float(h), 1.0
f, 500.0f);
    projectionMatrix.LoadMatrix(viewFrustum.GetProjectionMatri
x());

    //调用顶部载入单元矩阵
    modelViewMatrix.LoadIdentity();
}


int main(int argc, char* argv[])
{
    gltSetWorkingDirectory(argv[0]);
    glutInit(&argc, argv);

    //申请一个颜色缓存区、深度缓存区、双缓存区、模板缓存区
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH |
 GLUT_STENCIL);
```

```
    //设置window 的尺寸
    glutInitWindowSize(800, 600);

    //创建window的名称
    glutCreateWindow("GL_POINTS");

    //注册回调函数（改变尺寸）
    glutReshapeFunc(ChangeSize);

    //点击空格时，调用的函数
    glutKeyboardFunc(KeyPressFunc);

    //特殊键位函数（上下左右）
    glutSpecialFunc(SpecialKeys);

    //显示函数
    glutDisplayFunc(RenderScene);

    //判断一下是否能初始化glew库，确保项目能正常使用OpenGL 框架
    GLenum err = glewInit();
    if (GLEW_OK != err) {
        fprintf(stderr, "GLEW Error: %s\n", glewGetErrorString
(err));
        return 1;
    }

    //绘制
    SetupRC();

    //runloop运行循环
    glutMainLoop();
    return 0;
}
```