实验四&实验五——三维变换实验实验报告

一、 综述

本次实验目标为编程实现简单的三维建模，并通过交互控件对三维模型进行变换、投影、光照处理、明暗过渡、消隐等操作。

三维变换主要关于三维建模、三维空间基本变换、空间消隐和裁剪、三维投影变换以及光照计算五个方面。

三维空间基本变换变换的方法是在二维方法的基础上考虑了 Z 坐标而得到的。可以通过指定一个表示对象在三个坐标方向移动距离的三维变换向量来对对象进行平移变换。类似地，也可以利用三个坐标上的缩放因子来缩放对象。然而，三维旋转的扩展则不是那么简单。当我们讨论 XY 平面上的二维旋转时，只需考虑沿着垂直于 XY 平面的坐标轴进行旋转；而在三维空间中，可能选择空间的任意方向作为旋转轴方向。因此，现在的每一几何变换操作是一个从左边去乘坐标向量的 4x4 矩阵。和二维中一样，任意变换序列通过依序合并单个变换矩阵而得的一个矩阵表示。变换序列中每一后继矩阵从左边去和以前的变换矩阵合并。

要画出确定的、立体感很强的三维图形，就必须将那些被不透明的面（或物体）所遮挡的线段（或面）移去，这就是隐藏线或隐藏面的消隐处理。

在照相机模型中，由于照相机视角大小是有限的，因此需要三维裁剪。此外，将真实世界中的实体模型经过视见变换变换到视坐标系，再经过映射转换为屏幕坐标系。因此要显示三维空间的变换结果，就需要经过视见变换等一系列操作，将变换结果输出在屏幕上。本实验就需要利用这种技术，实现屏幕的输出绘制。

三维投影变换的目的是将三维形体转换为二维视图，然后可以进行输出绘制等操作。

简单光照模型仅考虑光源直接照射在景物表面所产生的光照效果，并且景物表面通常被假定为不透明，且具有均匀反射率。由此而来，虽然在计算处理上变得简单，但是其缺点也显而易见：虽然不同的物体具有不同的亮度，但是同一物体表面的亮度被看成是一个恒定的值，没有明暗的过渡，导致真实感不强。由于简单光照模型假定物体不透明，那么物体表面呈现的颜色仅由其反射光决定。反射光由两部分组成，一是环境反射，二是漫反射与镜面反射。环境反射假定入射光均匀地从周围环境入射至景物表面并等量地向各个方向反射出去，而漫反射分量和镜面反射分量则表示特定光源照射在景物表面上产生的反射光。

本次实验的处理流程为：先构建三维场景，实现三维建模，经过三维空间基本变换后对变换后的形体进行消隐和裁剪操作，然后经过光照计算，在空间投影变换与映射后输出到屏幕上显示。

三维变换实验目的有：了解掌握三维图形变换与绘制处理的基本流程，加深对三维变换原理的理解，能够通过编程实现简单的三维变换操作。同时，对三维空间建模、三维变换矩阵计算、三维空间变换、空间投影变换、照相机模型与视见变换有更加深入的理解，掌握三维光照模型、三维空间裁剪、投影绘制、消隐与明暗过渡的原理，并在实际中加以运用。

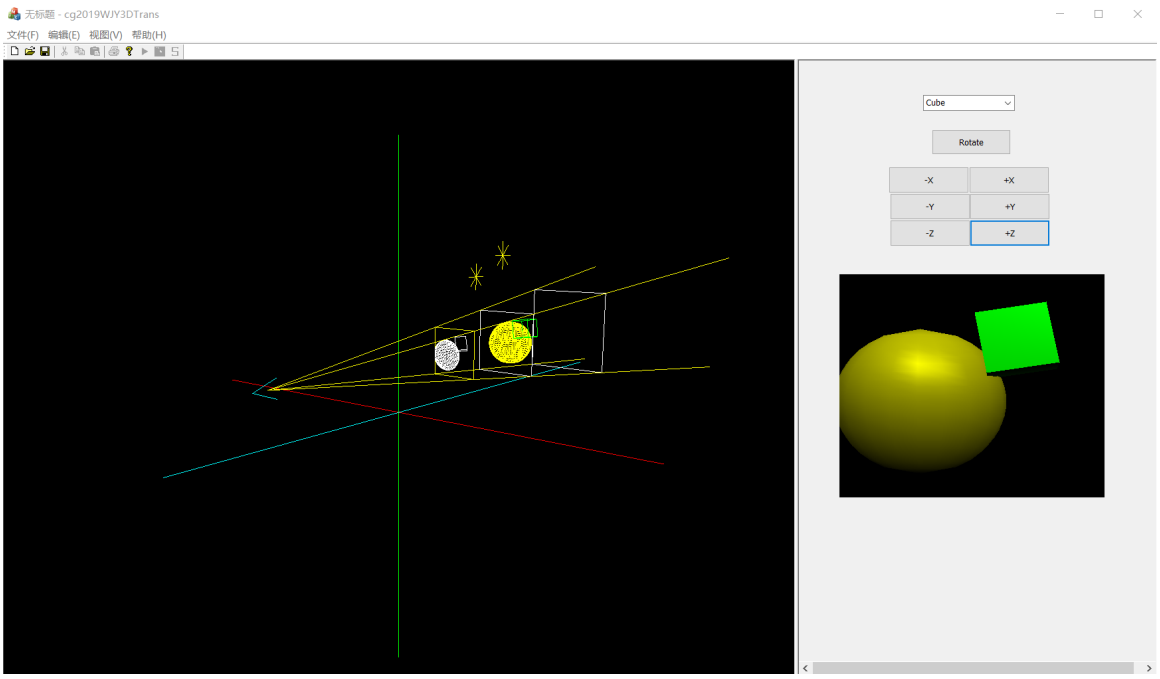
本次实验内容为：在实验四中实现三维图形基本变换、空间变换、投影变换和绘制处理；在实验五中实现视见变换（照相机模型）、消隐、裁剪、简单光照计算和 Ground 明暗过渡等算法。

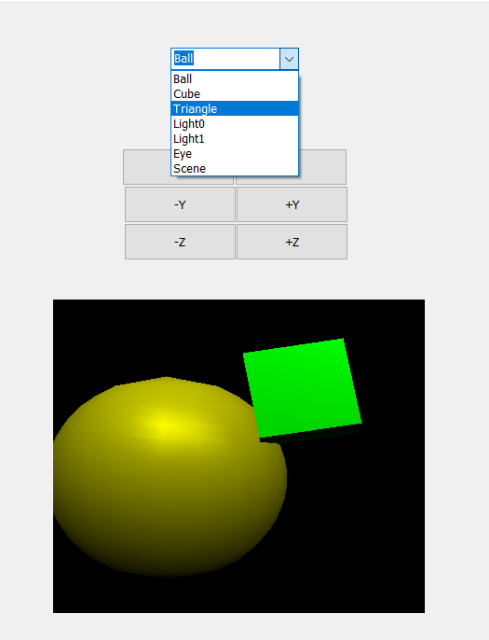
本次实验设计方案为：通过使用 Visual Studio2015 集成开发环境，利用 OpenGl 库绘制三维场景，并将需要变换的对象加入此场景。通过三维变换原理，计算变换矩阵，实现物体的三维空间变换；根据消隐、光照、明暗过渡等算法实现三维模型的更多操作。

本次实验的技术路线为：使用 Visual Studio 2015 集成开发环境，利用 MFC 框架搭建以图形界面交互的应用程序，借助 OpenGl 库函数进行场景绘制，编程语言为 C++。

二、 程序交互界面设计/编辑

1. 程序交互界面设计





如上图所示，程序交互界面主要由三部分组成：视图显示窗口、用户交互窗口和工具栏扩展选项（动画按钮、光照按钮、明暗过渡按钮）

1．视图显示窗口，由 CCg3DTransView 类管理，场景采用 OpenGL 库函数绘制。

2．用户交互窗口，由 CgTransSelect 类管理，包括：

空间变换对象选择—ComboBox 控件，选择 Ball、Cube、Trangle、eye、light0、light1、scene 七个操作对象

空间变换方式控制—Button 控件，控制变换方式：move、rotate

空间变换方向选择—Button 控件，控制变换的方向：-X、+X、-Y、+Y、-Z、+Z

3．动画按钮、光照按钮、明暗过渡按钮对应 CCg3DTransView 类，在该类中创建如下几个控件函数：

```
void Ccg20193DTransView::OnAutoplay() //动画效果
void Ccg20193DTransView::OnOnlight() //添加光照
void Ccg20193DTransView::OnOnshade() //明暗过渡
```

2．程序交互界面编辑

处理流程：

1．插入一个新的 Dialog，进入编辑页面。

2．选择 Check Box，并添加，修改相关属性。Sort 属性改为 false，添加成员 Ball、Cubic、Triangle、Eye、Light0、Light1、Scene，命名为 IDC_TRANSOBJ

3．选择 button，并添加，修改相关属性，将 Caption 改为 Move， ID 改为

DC_TRANSMODE。

4．选择 button 控件，修改相关属性，将 Caption 更改为-X，名称改为 IDC_XLEFT，其他控件同理。

5．选择 Picture Control，拖入编辑区域，调整成适当大小以作为窗口，ID 设置为 IDC_VIEWPORT。

6．为以上添加的控件添加消息映射函数，并根据相关处理逻辑补充消息映射函数，从而实现相关功能。

2．项目资源视图中选择 Toolbar 项目，编辑动画与添加光照、添加明暗过渡交互控制

处理流程：

1.选择控制框绘制所需的按钮图形，将 ID 改为 ID_AUTOPLAY。其余同理，ID 分别为 ID_ONLIGHT，ID_ONSHADE，分别对应动画按钮、光照按钮和明暗过渡按钮。

2．为以上添加的控件添加消息映射函数，并根据相关处理逻辑补充消息映射函数，从而实现相关功能。

三、 程序框架结构

三维变换程序 cg3DTrans 由六部分组成，分别为： CCg3DTransApp、CMainFrame、CCg3DTransDoc、CCg3DTransView、CDrawScene、CCgTransControl

a.模块功能概述：

1．CCg3DTransApp：

派生于 CWinApp，封装了 windows 应用程序控制和属性（有别于 dos 控制台程序等程序类型），是程序的主进程控制类和入口。和一些相关控件的初始化。

2．CMainFrame：

是程序的主窗体。包含菜单、工具栏、状态栏等

3．CCg3DTransDoc：

主要用于定义三维空间中的物体、裁剪立方体的相关参数，并且实现绘制

4．CCg3DTransView：

实现通过调用 Drawscence 类的成员函数，实现三维空间物体的动态变换、光照效果和明暗过渡

5．CDrawScene：

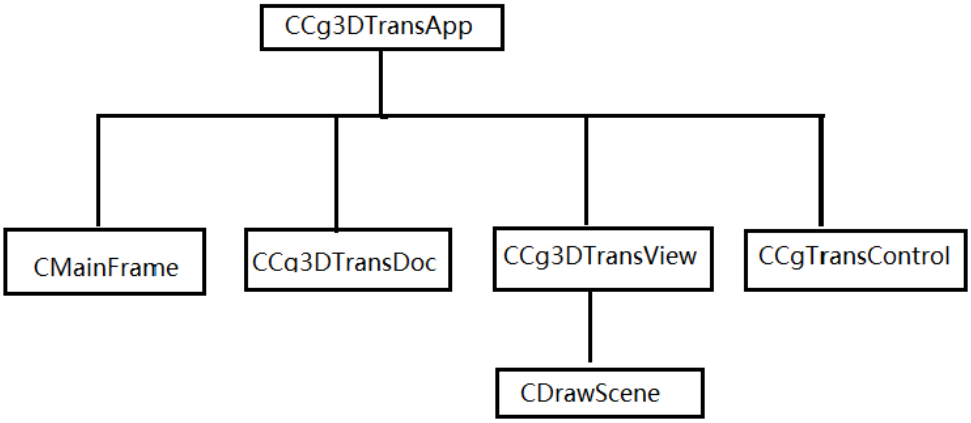
主要是绘制三维场景绘制的部分，包括裁剪，光线照射，三维变换

6. CCgTransControl：

控制三维空间内物体对象的变换、场景变换以及视角变换，实现交互功能：添加动画、添加光照、实现明暗过渡

b.模块调用关系：

如下图所示



四、 空间模型数据结构、变换参数设计说明

1. 空间模型数据结构

TypeDefine.h 定义描述空间模型数据结构。

空间形体数据结构 **Object_t**，最多 1200 个四边形组成。

```
typedef struct object {
        int    polyCount;            // Object polygons number
        bool   objectVisible;        // Whether object visible
        Poly_t objectSpace[1200];    // Object polygons
        float  transMatrix[4][3];    // Object transforms matrix
        Gpoint_t objCenter;          // Object center point
} Object_t, *Object_p;
```

空间四边形数据结构 **Poly_t**

```
    typedef struct poly {
        int      clipCount;
        int      polyCount;
        float    polyLight[10];
        bool     polyVisible;        // Polygon whther visible
```

```
        Gpoint_t polyObject[5];      // Original Object
        Gpoint_t transObject[5];     // Transform Original Object
        Gpoint_t clipObject[10];     // Clip Transform Object
        Gpoint_t projectObject[10];  // Project Clip Object
        Ipoint_t zBufferObject[10];  // Project Object Trans to Project
    Plane for Z-Buffer processing
        Gpoint_t viewDrawObject[5];  // View Draw Object
        Gpoint_t viewTransObject[5]; // View Transform Object
        Gpoint_t viewProjectObject[5]; // View Transform Project Object
    } Poly_t, *Poly_p;
```

空间点数据结构 **Gpoint_t**

```
typedefstructGpoint {
   float x,y,z;                          //坐标 x,y,z
} Gpoint_t, *Gpoint_p;
```

2. 空间变换参数。

CCg3DTransDoc.h 定义空间模型、**OpenGL** 空间场景变换参数和空间变换交互控制选择参数。

（**1**） 空间模型

```
// Trans Object Select No.
#define BALL        0
#define CUBE        1
#define TRIANGLE    2
#define LIGHT0      3
#define LIGHT1      4
#define EYE         5
#define SCENE       6
#define SPACEOBJECTS 16
Object_t m_spaceObjects[SPACEOBJECTS]; // Space Object Buffer
Float ballRadius;
Float zBack, zFront;              // Clip Box Front and Back Plane.
```

```
    Float eyeX,eyeY,eyeZ;              // Eye Position
    float winLpaneM, winRpaneM, winBpaneM, winTpaneM;     // window on XOY
project plane
```

（2） OpenGL 空间场景变换参数

```
    float m_translateVector[3];           // 空间场景平移变换参数
    float m_xAngle,m_yAngle,m_zAngle;     // 空间场景绕 XYZ 轴旋转变换参数
```

（3） 空间变换交互控制选择参数

```
    // Transform mode, direction and select
    char  m_transMode,m_transDir,m_transSelect;
```

3．空间变换参数初始化

CCg3DTransDoc.cpp 构造函数初始化空间变换参数。

```
    CCg3DTransDoc::CCg3DTransDoc()
    {
       // TODO: add one-time construction code here
       m_xAngle =  15.0f;              // 空间场景绕 X 轴旋转 15 度
       m_yAngle = -40.0f;              // 空间场景绕 Y 轴旋转负 40 度
       m_zAngle =   0.0f;
       m_translateVector[0] =  0.0f;   // 空间场景平移至（0.0f，-0.3f，-5.0f）
       m_translateVector[1] = -0.3f;
       m_translateVector[2] = -5.0f;
       m_transDir = 0;        // 0: -X  1: +X  2: -Y  3:+Y  4:-Z  5:+Z
       m_transMode = 0;       // 0: translate  1: rotate
       m_transSelect = 0;     // 0: Ball 1: Cube 2: Triangle 3:Eye
                              //  4: Light0 5: Light1  6: Scene
       // Window On XOY plane
       winLx = 1.0f;  winLy = 1.0f;
       winRx = 2.0f;  winRy = 2.0f;
       // Eye position
       eyeX = 1.5f;
       eyeY = 1.5f;
```

```
    eyeZ = 5.0f;
    // Clip box front and back plane
    zBack = -5.0f;
    zFront = -2.0f;
    // Space ball radius
    ballRadius = 0.5f;
    // Initial Space Object Buffer
    for  (inti = 0; i< SPACEOBJECTS; i++) {
        m_spaceObjects[i].polyCount = 0;
         m_spaceObjects[i].objectVisible = FALSE;
    }
    pCreateBall();
    pCreateCube();
    pCreateTriangle();
    pCreateClipBox();
}
```

五、 空间场景模型构建

空间场景模型由五部分组成：球、立方体、人眼、灯和裁剪约束体。

CCg3DTransDoc.cpp 成员函数：pCreateBallWithQuard ()，pCreateCube()，pCreateTriangle()，pCreateClipBox()

1. 空间球面模型（已实现三角形分割）

a．空间模型生成原理：

通过球心坐标和半径参数，计算出球面的位置，依照角度参数，通过 For 循环语句给球面绘制经纬线。在每一个由经纬线构成的四边形中通过循环遍历，将顶点数据加入三角形数据结构中保存，即可实现三角形剖分

b.实现方法：

```
void Ccg20193DTransDoc::pCreateBallWithQuard()
{
      int i, j, polyCount;
      float u, winCx, winCy, winCz;
```

```
Gpoint_t objectPoint[64][64];
float dh, angle, radius, height;
float BallCx, BallCy, BallCz;
// Calculate the ball's center point
winCx = (winLx + winRx) / 2.0f;
winCy = (winLy + winRy) / 2.0f;
winCz = 0.0f;
BallCz = (zFront + zBack) / 2.0f + 1.0f;
u = (BallCz - eyeZ) / (winCz - eyeZ);
BallCx = u * (winCx - eyeX) + eyeX + 0.1f;
BallCy = u * (winCy - eyeY) + eyeY;            //(1.6,1.5,-2.5)
radius = 0.0001f;
height = ballRadius;                           //0.5
dh = -ballRadius / 10;                         //-0.05
angle = 3.1415926f / 10;
for (i = 0; i <= 20; i++) {
   if (i == 20) radius = 0.0001f;
   for (j = 0; j <= 20; j++) {
      objectPoint[i][j].y = height + BallCy;
      objectPoint[i][j].x = radius * sinf(j * angle) + BallCx;
      objectPoint[i][j].z = radius * cosf(j * angle) + BallCz;
   }
   height = height + dh;
   radius = sqrtf(ballRadius * ballRadius - height * height);
   if (fabs(radius) < 0.0001f)  radius = 0.0001f;
}
polyCount = 0;
Object_p ballObject = &m_spaceObjects[BALL];
for (i = 0; i < 20; i++) {
   for (j = 0; j < 20; j++) {
```

```
            ballObject->objectSpace[polyCount].polyCount = 4;
            ballObject->objectSpace[polyCount].polyObject[0].x =
objectPoint[i][j].x;
            ballObject->objectSpace[polyCount].polyObject[0].y =
objectPoint[i][j].y;
            ballObject->objectSpace[polyCount].polyObject[0].z =
objectPoint[i][j].z;
            ballObject->objectSpace[polyCount].polyObject[1].x =
objectPoint[i + 1][j + 1].x;
            ballObject->objectSpace[polyCount].polyObject[1].y =
objectPoint[i + 1][j + 1].y;
            ballObject->objectSpace[polyCount].polyObject[1].z =
objectPoint[i + 1][j + 1].z;
            ballObject->objectSpace[polyCount].polyObject[2].x =
objectPoint[i][j + 1].x;
            ballObject->objectSpace[polyCount].polyObject[2].y =
objectPoint[i][j + 1].y;
            ballObject->objectSpace[polyCount].polyObject[2].z =
objectPoint[i][j + 1].z;
            ballObject->objectSpace[polyCount].polyObject[3].x =
objectPoint[i][j].x;
            ballObject->objectSpace[polyCount].polyObject[3].y =
objectPoint[i][j].y;
            ballObject->objectSpace[polyCount].polyObject[3].z =
objectPoint[i][j].z;
            polyCount++;
            ballObject->objectSpace[polyCount].polyCount = 4;
            ballObject->objectSpace[polyCount].polyObject[0].x =
objectPoint[i][j].x;
            ballObject->objectSpace[polyCount].polyObject[0].y =
```

```
objectPoint[i][j].y;
          ballObject->objectSpace[polyCount].polyObject[0].z =
objectPoint[i][j].z;
          ballObject->objectSpace[polyCount].polyObject[1].x =
objectPoint[i + 1][j].x;
          ballObject->objectSpace[polyCount].polyObject[1].y =
objectPoint[i + 1][j].y;
          ballObject->objectSpace[polyCount].polyObject[1].z =
objectPoint[i + 1][j].z;
          ballObject->objectSpace[polyCount].polyObject[2].x =
objectPoint[i + 1][j + 1].x;
          ballObject->objectSpace[polyCount].polyObject[2].y =
objectPoint[i + 1][j + 1].y;
          ballObject->objectSpace[polyCount].polyObject[2].z =
objectPoint[i + 1][j + 1].z;
          ballObject->objectSpace[polyCount].polyObject[3].x =
objectPoint[i][j].x;
          ballObject->objectSpace[polyCount].polyObject[3].y =
objectPoint[i][j].y;
          ballObject->objectSpace[polyCount].polyObject[3].z =
objectPoint[i][j].z;
          polyCount = polyCount + 1;
       }
    }
    ballObject->objCenter.x = BallCx;
    ballObject->objCenter.y = BallCy;
    ballObject->objCenter.z = BallCz;
    ballObject->polyCount = polyCount;
    // ----- Scene Description -----
    for (i = 0; i < 4; i++)
```

```
            for (j = 0; j < 3; j++)
               if (i == j) ballObject->transMatrix[i][j] = 1.0;
               else        ballObject->transMatrix[i][j] = 0.0;
               m_polygonNumbers += polyCount;    // Count polygon numbers.
      }
```

2. 空间立方体模型

a. 空间模型生成原理:

首先根据四边形边长和四边形中心点坐标，计算出六个面的正方形，并将他们的顶点坐标保存在数据结构中。

b. 实现方法:

```
void Ccg20193DTransDoc::pCreateCube()
{
   // Create a CUBE .....
   float u, winCx, winCy, winCz;
   float radius = 0.2f, CubeLx, CubeLy, CubeLz;
   // Calculate the cube's center point
   winCx = (winLx + winRx) / 2.0f;
   winCy = (winLy + winRy) / 2.0f;
   winCz = 0.0f;
   CubeLz = (zFront + zBack) / 2.0f + 1.0f;
   u = (CubeLz - eyeZ) / (winCz - eyeZ);
   CubeLx = u * (winCx - eyeX) + eyeX + 0.3f;
   CubeLy = u * (winCy - eyeY) + eyeY;
   Object_p cubeObject = &m_spaceObjects[CUBE];
   // Left Plane
   cubeObject->objectSpace[0].polyCount = 5;
   cubeObject->objectSpace[0].polyObject[0].x = CubeLx + 0.0f;
   cubeObject->objectSpace[0].polyObject[0].y = CubeLy + 0.0f;
   cubeObject->objectSpace[0].polyObject[0].z = CubeLz + 0.0f;
   cubeObject->objectSpace[0].polyObject[1].x = CubeLx + 0.0f;
```

```
cubeObject->objectSpace[0].polyObject[1].y = CubeLy + 0.0f;
cubeObject->objectSpace[0].polyObject[1].z = CubeLz + 2 * radius;
cubeObject->objectSpace[0].polyObject[2].x = CubeLx + 0.0f;
cubeObject->objectSpace[0].polyObject[2].y = CubeLy + 2 * radius;
cubeObject->objectSpace[0].polyObject[2].z = CubeLz + 2 * radius;
cubeObject->objectSpace[0].polyObject[3].x = CubeLx + 0.0f;
cubeObject->objectSpace[0].polyObject[3].y = CubeLy + 2 * radius;
cubeObject->objectSpace[0].polyObject[3].z = CubeLz + 0.0f;
cubeObject->objectSpace[0].polyObject[4].x = CubeLx + 0.0f;
cubeObject->objectSpace[0].polyObject[4].y = CubeLy + 0.0f;
cubeObject->objectSpace[0].polyObject[4].z = CubeLz + 0.0f;
// Right Plane
cubeObject->objectSpace[1].polyCount = 5;
cubeObject->objectSpace[1].polyObject[0].x = CubeLx + 2 * radius;
cubeObject->objectSpace[1].polyObject[0].y = CubeLy + 0.0f;
cubeObject->objectSpace[1].polyObject[0].z = CubeLz + 0.0f;
cubeObject->objectSpace[1].polyObject[1].x = CubeLx + 2 * radius;
cubeObject->objectSpace[1].polyObject[1].y = CubeLy + 2 * radius;
cubeObject->objectSpace[1].polyObject[1].z = CubeLz + 0.0f;
cubeObject->objectSpace[1].polyObject[2].x = CubeLx + 2 * radius;
cubeObject->objectSpace[1].polyObject[2].y = CubeLy + 2 * radius;
cubeObject->objectSpace[1].polyObject[2].z = CubeLz + 2 * radius;
cubeObject->objectSpace[1].polyObject[3].x = CubeLx + 2 * radius;
cubeObject->objectSpace[1].polyObject[3].y = CubeLy + 0.0f;
cubeObject->objectSpace[1].polyObject[3].z = CubeLz + 2 * radius;
cubeObject->objectSpace[1].polyObject[4].x = CubeLx + 2 * radius;
cubeObject->objectSpace[1].polyObject[4].y = CubeLy + 0.0f;
cubeObject->objectSpace[1].polyObject[4].z = CubeLz + 0.0f;
// Down Plane
cubeObject->objectSpace[2].polyCount = 5;
```

```
cubeObject->objectSpace[2].polyObject[0].x = CubeLx + 0.0f;
cubeObject->objectSpace[2].polyObject[0].y = CubeLy + 0.0f;
cubeObject->objectSpace[2].polyObject[0].z = CubeLz + 0.0f;
cubeObject->objectSpace[2].polyObject[1].x = CubeLx + 2 * radius;
cubeObject->objectSpace[2].polyObject[1].y = CubeLy + 0.0f;
cubeObject->objectSpace[2].polyObject[1].z = CubeLz + 0.0f;
cubeObject->objectSpace[2].polyObject[2].x = CubeLx + 2 * radius;
cubeObject->objectSpace[2].polyObject[2].y = CubeLy + 0.0f;
cubeObject->objectSpace[2].polyObject[2].z = CubeLz + 2 * radius;
cubeObject->objectSpace[2].polyObject[3].x = CubeLx + 0.0f;
cubeObject->objectSpace[2].polyObject[3].y = CubeLy + 0.0f;
cubeObject->objectSpace[2].polyObject[3].z = CubeLz + 2 * radius;
cubeObject->objectSpace[2].polyObject[4].x = CubeLx + 0.0f;
cubeObject->objectSpace[2].polyObject[4].y = CubeLy + 0.0f;
cubeObject->objectSpace[2].polyObject[4].z = CubeLz + 0.0f;
// Up Plane
cubeObject->objectSpace[3].polyCount = 5;
cubeObject->objectSpace[3].polyObject[0].x = CubeLx + 0.0f;
cubeObject->objectSpace[3].polyObject[0].y = CubeLy + 2 * radius;
cubeObject->objectSpace[3].polyObject[0].z = CubeLz + 0.0f;
cubeObject->objectSpace[3].polyObject[1].x = CubeLx + 0.0f;
cubeObject->objectSpace[3].polyObject[1].y = CubeLy + 2 * radius;
cubeObject->objectSpace[3].polyObject[1].z = CubeLz + 2 * radius;
cubeObject->objectSpace[3].polyObject[2].x = CubeLx + 2 * radius;
cubeObject->objectSpace[3].polyObject[2].y = CubeLy + 2 * radius;
cubeObject->objectSpace[3].polyObject[2].z = CubeLz + 2 * radius;
cubeObject->objectSpace[3].polyObject[3].x = CubeLx + 2 * radius;
cubeObject->objectSpace[3].polyObject[3].y = CubeLy + 2 * radius;
cubeObject->objectSpace[3].polyObject[3].z = CubeLz + 0.0f;
cubeObject->objectSpace[3].polyObject[4].x = CubeLx + 0.0f;
```

```
cubeObject->objectSpace[3].polyObject[4].y = CubeLy + 2 * radius;
cubeObject->objectSpace[3].polyObject[4].z = CubeLz + 0.0f;
// Front Plane
cubeObject->objectSpace[4].polyCount = 5;
cubeObject->objectSpace[4].polyObject[0].x = CubeLx + 0.0f;
cubeObject->objectSpace[4].polyObject[0].y = CubeLy + 0.0f;
cubeObject->objectSpace[4].polyObject[0].z = CubeLz + 2 * radius;
cubeObject->objectSpace[4].polyObject[1].x = CubeLx + 2 * radius;
cubeObject->objectSpace[4].polyObject[1].y = CubeLy + 0.0f;
cubeObject->objectSpace[4].polyObject[1].z = CubeLz + 2 * radius;
cubeObject->objectSpace[4].polyObject[2].x = CubeLx + 2 * radius;
cubeObject->objectSpace[4].polyObject[2].y = CubeLy + 2 * radius;
cubeObject->objectSpace[4].polyObject[2].z = CubeLz + 2 * radius;
cubeObject->objectSpace[4].polyObject[3].x = CubeLx + 0.0f;
cubeObject->objectSpace[4].polyObject[3].y = CubeLy + 2 * radius;
cubeObject->objectSpace[4].polyObject[3].z = CubeLz + 2 * radius;
cubeObject->objectSpace[4].polyObject[4].x = CubeLx + 0.0f;
cubeObject->objectSpace[4].polyObject[4].y = CubeLy + 0.0f;
cubeObject->objectSpace[4].polyObject[4].z = CubeLz + 2 * radius;
// Back Plane
cubeObject->objectSpace[5].polyCount = 5;
cubeObject->objectSpace[5].polyObject[0].x = CubeLx + 0.0f;
cubeObject->objectSpace[5].polyObject[0].y = CubeLy + 0.0f;
cubeObject->objectSpace[5].polyObject[0].z = CubeLz + 0.0f;
cubeObject->objectSpace[5].polyObject[1].x = CubeLx + 0.0f;
cubeObject->objectSpace[5].polyObject[1].y = CubeLy + 2 * radius;
cubeObject->objectSpace[5].polyObject[1].z = CubeLz + 0.0f;
cubeObject->objectSpace[5].polyObject[2].x = CubeLx + 2 * radius;
cubeObject->objectSpace[5].polyObject[2].y = CubeLy + 2 * radius;
cubeObject->objectSpace[5].polyObject[2].z = CubeLz + 0.0f;
```

```
    cubeObject->objectSpace[5].polyObject[3].x = CubeLx + 2 * radius;
    cubeObject->objectSpace[5].polyObject[3].y = CubeLy + 0.0f;
    cubeObject->objectSpace[5].polyObject[3].z = CubeLz + 0.0f;
    cubeObject->objectSpace[5].polyObject[4].x = CubeLx + 0.0f;
    cubeObject->objectSpace[5].polyObject[4].y = CubeLy + 0.0f;
    cubeObject->objectSpace[5].polyObject[4].z = CubeLz + 0.0f;
    cubeObject->polyCount = 6;
    // for(int i = 0; i < 6; i++)
    //   cubeObject->objectSpace[i].polyVisible = true;
    for (int i = 0; i < 4; i++)
       for (int j = 0; j < 3; j++)
          if (i == j) cubeObject->transMatrix[i][j] = 1.0;
          else       cubeObject->transMatrix[i][j] = 0.0;
}
```

3. 空间裁剪约束体模型

a.空间模型生成原理：

由视点和窗口可见域四个顶点的连线构成四棱放射锥，加上前后面即可构成裁剪约束体，为一四棱台几何体，根据裁剪约束体计算方程，可以计算出经过空间裁剪后的目标绘制图形，即可实现三维裁剪操作

b.实现方法：

```
void Ccg20193DTransDoc::pCreateClipBox()
{
       float u;
       clipCubeBackLB[2] = zBack;
       u = (zBack - eyeZ) / (0 - eyeZ);
       clipCubeBackLB[0] = u * (winLx - eyeX) + eyeX;
       clipCubeBackLB[1] = u * (winLy - eyeY) + eyeY;
       clipCubeBackRB[2] = zBack;
       u = (zBack - eyeZ) / (0 - eyeZ);
       clipCubeBackRB[0] = u * (winRx - eyeX) + eyeX;
```

```
        clipCubeBackRB[1] = u * (winLy - eyeY) + eyeY;
        clipCubeBackRT[2] = zBack;
        u = (zBack - eyeZ) / (0 - eyeZ);
        clipCubeBackRT[0] = u * (winRx - eyeX) + eyeX;
        clipCubeBackRT[1] = u * (winRy - eyeY) + eyeY;
        clipCubeBackLT[2] = zBack;
        u = (zBack - eyeZ) / (0 - eyeZ);
        clipCubeBackLT[0] = u * (winLx - eyeX) + eyeX;
        clipCubeBackLT[1] = u * (winRy - eyeY) + eyeY;
        clipCubeFrontLB[2] = zFront;
        u = (zFront - eyeZ) / (0 - eyeZ);
        clipCubeFrontLB[0] = u * (winLx - eyeX) + eyeX;
        clipCubeFrontLB[1] = u * (winLy - eyeY) + eyeY;
        clipCubeFrontRB[2] = zFront;
        u = (zFront - eyeZ) / (0 - eyeZ);
        clipCubeFrontRB[0] = u * (winRx - eyeX) + eyeX;
        clipCubeFrontRB[1] = u * (winLy - eyeY) + eyeY;
        clipCubeFrontRT[2] = zFront;
        u = (zFront - eyeZ) / (0 - eyeZ);
        clipCubeFrontRT[0] = u * (winRx - eyeX) + eyeX;
        clipCubeFrontRT[1] = u * (winRy - eyeY) + eyeY;
        clipCubeFrontLT[2] = zFront;
        u = (zFront - eyeZ) / (0 - eyeZ);
        clipCubeFrontLT[0] = u * (winLx - eyeX) + eyeX;
        clipCubeFrontLT[1] = u * (winRy - eyeY) + eyeY;
        winLpaneM = (eyeX - winLx) / eyeZ;
        winRpaneM = (eyeX - winRx) / eyeZ;
        winBpaneM = (eyeY - winLy) / eyeZ;
        winTpaneM = (eyeY - winRy) / eyeZ;
}
```

六、 空间变换绘制

CCg3DTransView 类和 CDrawScene 类实现空间变换绘制。

1. CCg3DTransView 类

综述：类 CCg3DTransView 实现通过调用 Drawscence 类的成员函数，实现三维空间物体的动态变换、光照效果和明暗过渡

（1） CCg3DTransView.h 中定义成员变量。

```
CClientDC  *m_pDC;            // 视区设备描述
CRectm_viewRect;              // 视区尺寸参量
```

（2） CCg3DTransView.h 中定义成员函数。

```
void DrawScene();             // 空间场景绘制函数
BOOL bSetupPixelFormat();     // 空间场景显示格式设置函数
```

（3） 重载 WM_SIZE 和 WM_CREATE 消息处理函数

```
afx_msg void OnSize(UINT nType, int cx, int cy);
afx_msgintOnCreate(LPCREATESTRUCT lpCreateStruct);
```

处理流程说明：

当应用程序通过调用成员函数 Create 或 CreateEx 请求创建 Windows 的窗口时，框架调用这个成员函数。CWnd 对象在窗口被创建以后，但是在它变为可见之前接收到对这个函数的调用。

OnCreate 是在 Create 或 CreateEx 成员函数返回之前被调用的。重载这个成员函数以执行派生类所需的初始化工作。CREATESTRUCT 结构中包含了用于创建窗口的参数的拷贝。

补充：

CView 是视图是程序设计中使用率最高的窗口对象，它是用户的主要操作界面。因为它通常以某种形式表示文档数据，所以称之为视图。一个视图对象只关联一个文档对象；一个文档对象可以关联多个视图，每个视图对象以不同形式表示文档数据。

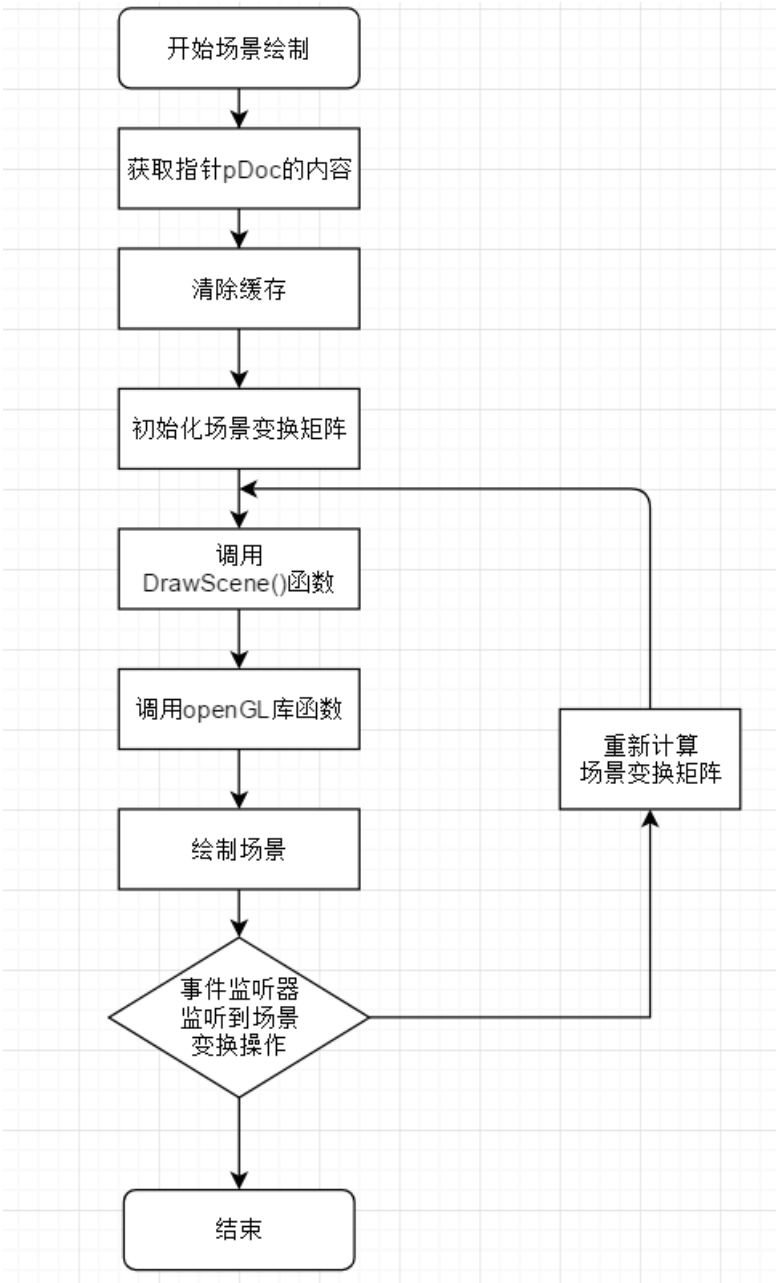
（4） 空间场景显示格式设置函数

```
BOOL CCg3DTransView::bSetupPixelFormat()
```

像素格式明确了 OpenGL 绘制平面的特性，如象素缓冲区是单缓冲还是双缓冲，数据是 RGBA 方式还是 Color Index 方式等。每个 OpenGL 显示设备一般用名为 PIXELFORMATDESCRIPTOR 的结构来表示某个的像素格式，这个结构包含 26 个属性信息。

（5） 空间场景绘制

处理流程：



简要说明：

实现函数：**void DrawScene()**；

```
void CCg3DTransView::DrawScene()
{
 static BOOL bBusy = FALSE;
 CCg3DTransDoc* pDoc = (CCg3DTransDoc*)GetDocument();
    if (bBusy)return;
```

```
bBusy = TRUE;
   // 清祯缓存
   glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   // 构建空间场景变换矩阵
   glPushMatrix();
   glTranslatef(pDoc->m_translateVector[0],
   pDoc->m_translateVector[1],
   pDoc->m_translateVector[2]);
glRotatef(pDoc->m_xAngle, 1.0f, 0.0f, 0.0f);
glRotatef(pDoc->m_yAngle, 0.0f, 1.0f, 0.0f);
glRotatef(pDoc->m_zAngle, 0.0f, 0.0f, 1.0f);
   glScalef(0.3f, 0.3f, 0.3f);   /* Make Scene Smaller Enough to Look */
     // Call CDrawScene 输出绘制空间场景坐标 XYZ 轴
   if (m_drawScene != NULL) m_drawScene->DrawScene();
    glPopMatrix();                    // 空间场景变换、绘制结束。
glFinish();
     SwapBuffers(wglGetCurrentDC());    // 交换祯缓存输出。
     bBusy = FALSE;
// pDoc->UpdateAllViews(this);
}
```

2. CDrawScene 类

CDrawScene 类说明综述：功能、组成和调用关系：

（1） CDrawScene 成员函数

//通过 OpenGL 实现场景绘制，包括：三维空间的 X、Y、Z 坐标轴，视线区域，视点，两个光源，此函数在 CDrawScene::CDrawScene()中被调用

void DrawBackground();

//绘制三维空间场景中的物体：球，立方体在函数 CDrawScene::CDrawScene()中被调用，通过 for 循环语句实现

void DrawSpaceObject();

```
    //绘制变换后的三维图形，在函数 CDrawScene::CDrawScene()中被调用
    void TransSpaceObject();
    //绘制经过视见变换后的三维空间几何体，在函数 CDrawScene::CDrawScene()中被调用
    void projectSpaceObject();
    //用于计算变换矩阵，根据不同选项：-X,+X,-Y,+Y,-Z,+Z,以及不同变换模式：translate，rotate 计算出相应的变换矩阵。通过点击交互界面的相关按钮，改变按钮的参数值，在函数 CDrawScene::CDrawScene()中被调用。
    void CaculateMatrix();
    //绕 X 轴旋转变换函数，在函数 CaculateMatrix()中被调用
    void rotateX3Dmatrix(float S, float C);
    //绕 Y 轴旋转变换函数，在函数 CaculateMatrix()中被调用
    void rotateY3Dmatrix(float S, float C);
    //绕 Z 轴旋转变换函数，在函数 CaculateMatrix()中被调用
    void rotateZ3Dmatrix(float S, float C);
    //平移变换函数，在函数 CaculateMatrix()中被调用
    void translate3dMatrix(float dx, float dy, float dz);
    //实现消隐的函数，在函数 CDrawScene::CDrawScene()中被调用
    void pRemoveBackFace();
   //以下裁剪算法所要调用的函数,在 CDrawScene::pClipSpaceObject()被调用
    void pClipSpaceObject();
    int pVisible(float x, float y, float z, int pane);
    int outPut(float x, float y, float z, int *outCount, Gpoint_p polyClip);
    int pLineCrossPane(float sx, float sy, float sz,
       float px, float py, float pz, int pane);
    int pLineInterSectPane(float sx, float sy, float sz,
       float px, float py, float pz,
```

```
                int pane, int *outCount, Gpoint_p polyClip);
            void pObjectCenter();     //计算物体的中心点，在
        pLightSpaceBall()被调用
            void pLightSpaceBall();   //判断如果是球体或者正方体则投影光
        线
          void pLightSpaceObject(); //如果不是球体或者正方体则无法投影光线
```

（2） 空间场景变换绘制说明

```
float m_translateVector[3];               // Scene Translation parameters.
float m_xAngle, m_yAngle, m_zAngle;       // Scene Rotation parameters.
BOOL m_selfRotate;
int m_transDir, m_transMode, m_transSelect;   // Transformation Direction,
Mode and Object Select Number.

void Ccg20193DTransView::DrawScene()
{
   static BOOL bBusy = FALSE;
   Ccg20193DTransDoc* pDoc = (Ccg20193DTransDoc*)GetDocument();

   if (bBusy)   return;

   bBusy = TRUE;

   glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

   glPushMatrix();
   glTranslatef(pDoc->m_translateVector[0],
      pDoc->m_translateVector[1],
      pDoc->m_translateVector[2]);
   glRotatef(pDoc->m_xAngle, 1.0f, 0.0f, 0.0f);
```

```
    glRotatef(pDoc->m_yAngle, 0.0f, 1.0f, 0.0f);
    glRotatef(pDoc->m_zAngle, 0.0f, 0.0f, 1.0f);
    glScalef(0.3f, 0.3f, 0.3f);   /* Make Scene Smaller Enough to Look */

    if (m_drawScene)  m_drawScene->DrawScene();
    //if (m_cameraView) m_cameraView->DrawScene();

    glPopMatrix();

    glFinish();
    SwapBuffers(wglGetCurrentDC());

    bBusy = FALSE;

    // Notify CCgTransControl can process the Project Result to PictureBox
'VIEWPORT'
    pDoc->UpdateAllViews(this);
}
```

（3） 空间基本变换矩阵计算

原理：

a、确定三维物体上各点的位置坐标；

b、 引入齐次坐标，求出所作变换相应的变换矩阵；

c、 将所作变换用矩阵表示，通过运算求得三维物体上各点经变换后的点坐标值；

d、 由变换后得到的二维点绘出三维物体投影后的三视图

实现方法：

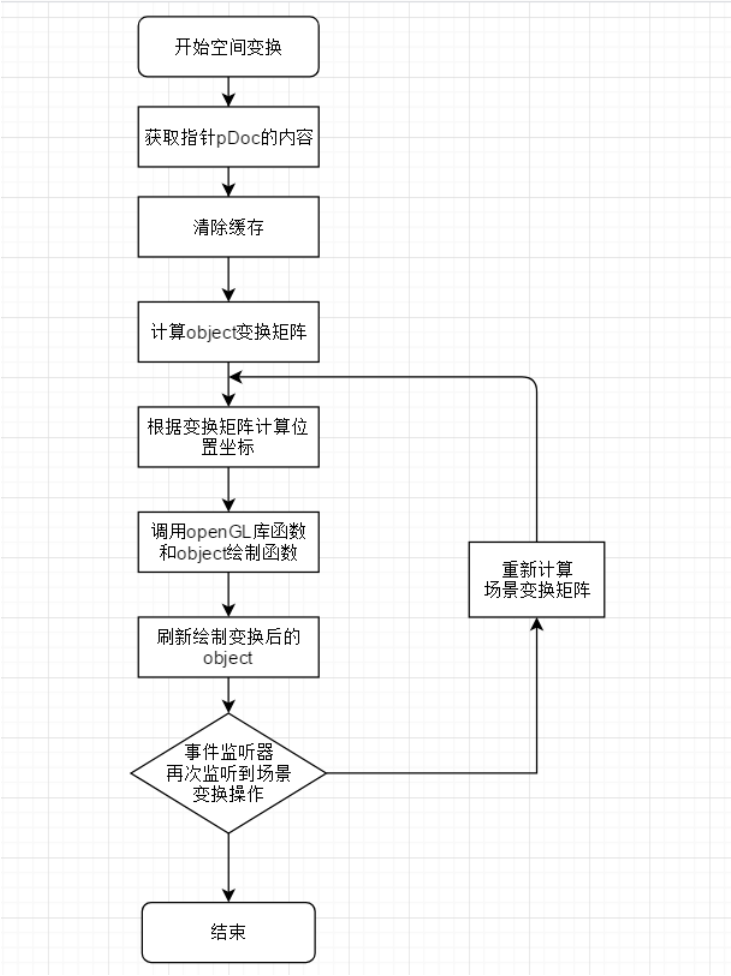
```
void CDrawScene::translate3dMatrix(float Tx, float Ty, float Tz)
{
    Ccg20193DTransDoc *pDoc = (Ccg20193DTransDoc *)m_pView->GetDocument();
    m_whoObject->transMatrix[3][0] += Tx;
    m_whoObject->transMatrix[3][1] += Ty;
```

```
    m_whoObject->transMatrix[3][2] += Tz;
}
void CDrawScene::rotateX3Dmatrix(float S, float C)
{
    Ccg20193DTransDoc *pDoc = (Ccg20193DTransDoc *)m_pView->GetDocument();
    for (int i = 0; i < 4; i++) {
        float temp;
        temp = m_whoObject->transMatrix[i][1] * C -
               m_whoObject->transMatrix[i][2] * S;
        m_whoObject->transMatrix[i][2] = m_whoObject->transMatrix[i][1] * S
                                       + m_whoObject->transMatrix[i][2] * C;
        m_whoObject->transMatrix[i][1] = temp;
    }
}
void CDrawScene::rotateY3Dmatrix(float S, float C)
{
    Ccg20193DTransDoc *pDoc = (Ccg20193DTransDoc *)m_pView->GetDocument();
    for (int i = 0; i < 4; i++) {
        float temp;
        temp = m_whoObject->transMatrix[i][0] * C +
               m_whoObject->transMatrix[i][2] * S;
         m_whoObject->transMatrix[i][2] = -m_whoObject->transMatrix[i][0] * S
                                        + m_whoObject->transMatrix[i][2] * C;
        m_whoObject->transMatrix[i][0] = temp;
    }
}
void CDrawScene::rotateZ3Dmatrix(float S, float C)
{
    Ccg20193DTransDoc *pDoc = (Ccg20193DTransDoc *)m_pView->GetDocument();
    for (int i = 0; i < 4; i++) {
```

```
        float temp;
       temp = m_whoObject->transMatrix[i][0] * C -
              m_whoObject->transMatrix[i][1] * S;
       m_whoObject->transMatrix[i][1] = m_whoObject->transMatrix[i][0] * S
                                        + m_whoObject->transMatrix[i][1] * C;
        m_whoObject->transMatrix[i][0] = temp;
    }
}
```

（4） 空间变换

处理流程：



原理和实现方法：

```
void CDrawScene::TransSpaceObject()
{
            Ccg20193DTransDoc *pDoc = (Ccg20193DTransDoc
```

```
*)m_pView->GetDocument();
    // First Trans Object according to its matrix
    for (int i = 0; i < m_whoObject->polyCount; i++) {
       for (int j = 0; j < m_whoObject->objectSpace[i].polyCount;
j++) {
          m_whoObject->objectSpace[i].transObject[j].x =
          m_whoObject->objectSpace[i].polyObject[j].x *
m_whoObject->transMatrix[0][0]+
          m_whoObject->objectSpace[i].polyObject[j].y *
m_whoObject->transMatrix[1][0]+
          m_whoObject->objectSpace[i].polyObject[j].z *
m_whoObject->transMatrix[2][0]+
          m_whoObject->transMatrix[3][0];
          m_whoObject->objectSpace[i].transObject[j].y =
          m_whoObject->objectSpace[i].polyObject[j].x *
m_whoObject->transMatrix[0][1]+
          m_whoObject->objectSpace[i].polyObject[j].y *
m_whoObject->transMatrix[1][1]+
          m_whoObject->objectSpace[i].polyObject[j].z *
m_whoObject->transMatrix[2][1]+
          m_whoObject->transMatrix[3][1];
          m_whoObject->objectSpace[i].transObject[j].z =
          m_whoObject->objectSpace[i].polyObject[j].x *
m_whoObject->transMatrix[0][2]+
          m_whoObject->objectSpace[i].polyObject[j].y *
m_whoObject->transMatrix[1][2]+
          m_whoObject->objectSpace[i].polyObject[j].z *
m_whoObject->transMatrix[2][2]+
          m_whoObject->transMatrix[3][2];
       }
```

```
                }
            }
```

（5） 空间投影变换

原理和实现方法：

```
void CDrawScene::projectSpaceObject()
{
   Ccg20193DTransDoc *pDoc = (Ccg20193DTransDoc *)m_pView->GetDocument();
   // Project object to XOY plane
   for (int i = 0; i < m_whoObject->polyCount; i++) {
      if (m_whoObject->objectSpace[i].polyVisible) {
         for (int j = 0; j < m_whoObject->objectSpace[i].clipCount; j++) {
            m_whoObject->objectSpace[i].projectObject[j].x =
               -(m_whoObject->objectSpace[i].clipObject[j].x - pDoc->eyeX)
/
               (m_whoObject->objectSpace[i].clipObject[j].z - pDoc->eyeZ)
*
               pDoc->eyeZ + pDoc->eyeX;
            m_whoObject->objectSpace[i].projectObject[j].y =
               -(m_whoObject->objectSpace[i].clipObject[j].y - pDoc->eyeY)
/
               (m_whoObject->objectSpace[i].clipObject[j].z - pDoc->eyeZ)
*
               pDoc->eyeZ + pDoc->eyeY;
            m_whoObject->objectSpace[i].projectObject[j].z = 0.0f;
         }
      }
   }
   // Then draw project result on XOY project plane.
   glColor3f(1.0f, 1.0f, 1.0f);
   for (int i = 0; i < m_whoObject->polyCount; i++) {
```

```
        if (m_whoObject->objectSpace[i].polyVisible) {
            glBegin(GL_LINE_STRIP);
            for (int j = 0; j < m_whoObject->objectSpace[i].clipCount; j++)
                glVertex3f(m_whoObject->objectSpace[i].projectObject[j].x,
                    m_whoObject->objectSpace[i].projectObject[j].y,
                    m_whoObject->objectSpace[i].projectObject[j].z);
            glEnd();
        }
    }
}
```

七、 空间变换交互与视见变换

CCgTransControl 类实现空间变换交互控制与视见变换绘制。控制三维空间内物体对象的变换、场景变换以及视角变换，实现交互功能：添加动画、添加光照、实现明暗过渡

原理和实现方法：

```
afx_msg void OnClickedXleft();          //当点击+X 按钮时，沿 X 方向向左平移
    afx_msg void OnClickedXright();         //当点击-X 按钮时，沿 X 方向向右平移
    afx_msg void OnClickedYup();        //当点击+Y 按钮时，沿 Y 方向向上平移
    afx_msg void OnClickedYdown();          //当点击-Y 按钮时，沿 Y 方向向下平移
    afx_msg void OnClickedZfront();         //当点击+Z 按钮时，沿 Z 方向向前平移
    afx_msg void OnClickedZback();          //当点击-Z 按钮时，沿 Z 方向向后平移
    afx_msg void OnClickedTransmode();      //当点击模式切换按钮时，切换变换模式
    int      m_objNumber;               // Object number
    Object_p m_whoObject;               // Object pointer
    COLORREF vcObjColor[SPACEOBJECTS];
    void pTransToZbuffer(CRect dcRect);    //视见变换
    void pDrawLineObject(CDC *pDC, CRect dcRect);
    void pDrawLightObject(CDC *pDC, CRect dcRect, float maxShade, float
minShade);                              //为物体增加光照
    int polyCount;
    BOOL m_bitmapOutput;
```

```
    float ymax[N], ymin[N];
    int ibegin, iend, scan, pdges;
    float Dx[N], Xa[N], Sc[N], Dc[N];
    int   prjPlaneRetSize;              // Z-Buffer Array Size
    float *prjPlaneRetZ;                // Z-Buffer Array Memory
    //根据 prjPlaneRet 初始化 z 缓冲 prjPlaneRetZ
    void InitPrjPlaneData(int width, int height);
    //绘制明暗过渡后的物体
    void pDrawShadeLightObject(CDC *pDC, CRect dcRect, float maxShade, float
minShade);
    //多边形填充算法
    void FillPolygon(CDC *pDC, int n, int *x, int *y, int *color, CRect dcRect);
    void Loadpolygon(int n, int *x, int *y, int *color);
    void PolyInsert(float x1, float y1, float x2, float y2, int c1, int c2);
    void UpdateXvalue();
    void XSort(int begin, int i);
    void Fillscan(CDC *pDC, CRect dcRect);
    void Include();
    //用于删除隐藏部分的 Z-buffer 方法。
    float CalculateZValue(int x, int y, CRect dcRect, int i);
    //openMP 处理加速
    BOOL m_openMP;
    _VPIXEL* prjPlaneDibImage;
    int m_imageWidth, m_imageHeight;
    BITMAPINFO* DisplayBitMapHeader;
    //使用 BitMap 绘制生成的图形
    char DisplayBitMapBuffer[sizeof(BITMAPINFO) + 16];
    void CreateDisplayBitMap(int width, int height);
    void _SetPixel(int x, int y, int red, int green, int blue);
```

八、 空间照相机模型视见变换

CameraView 类实现空间照相机模型视见变换与绘制。

原理和实现方法：

```
void CCgTransControl::pTransToZbuffer(CRect dcRect)
{
   int i, j;
   Ccg20193DTransDoc* pDoc = (Ccg20193DTransDoc*)GetDocument();

   float vxScale = (float)dcRect.right / (pDoc->winRx - pDoc->winLx);
   float vyScale = (float)dcRect.bottom / (pDoc->winRy - pDoc->winLy);
   for (i = 0; i < m_whoObject->polyCount; i++) {
      if (m_whoObject->objectSpace[i].polyVisible) {
         for (j = 0; j < m_whoObject->objectSpace[i].clipCount; j++)
{
            m_whoObject->objectSpace[i].zBufferObject[j].x =

(int)((m_whoObject->objectSpace[i].projectObject[j].x - pDoc->winLx) *
vxScale + 0.5f);
            m_whoObject->objectSpace[i].zBufferObject[j].y =
dcRect.bottom -

(int)((m_whoObject->objectSpace[i].projectObject[j].y - pDoc->winLy) *
vyScale + 0.5f);
         }
      }
   }
```

程序中的关键代码：

```
void CDrawScene::translate3dMatrix(float Tx, float Ty, float Tz)
{
   Ccg20193DTransDoc *pDoc = (Ccg20193DTransDoc *)m_pView->GetDocument();
   m_whoObject->transMatrix[3][0] += Tx;
   m_whoObject->transMatrix[3][1] += Ty;
   m_whoObject->transMatrix[3][2] += Tz;
}
void CDrawScene::rotateX3Dmatrix(float S, float C)
{
   Ccg20193DTransDoc *pDoc = (Ccg20193DTransDoc *)m_pView->GetDocument();
   for (int i = 0; i < 4; i++) {
      float temp;
      temp = m_whoObject->transMatrix[i][1] * C -
m_whoObject->transMatrix[i][2] * S;
      m_whoObject->transMatrix[i][2] = m_whoObject->transMatrix[i][1] * S +

m_whoObject->transMatrix[i][2] * C;
      m_whoObject->transMatrix[i][1] = temp;
   }
}
void CDrawScene::rotateY3Dmatrix(float S, float C)
{
   Ccg20193DTransDoc *pDoc = (Ccg20193DTransDoc *)m_pView->GetDocument();
   for (int i = 0; i < 4; i++) {
      float temp;
      temp = m_whoObject->transMatrix[i][0] * C +
m_whoObject->transMatrix[i][2] * S;
      m_whoObject->transMatrix[i][2] = -m_whoObject->transMatrix[i][0] * S
```

```
+
m_whoObject->transMatrix[i][2] * C;
      m_whoObject->transMatrix[i][0] = temp;
   }
}
void CDrawScene::rotateZ3Dmatrix(float S, float C)
{
   Ccg20193DTransDoc *pDoc = (Ccg20193DTransDoc *)m_pView->GetDocument();
   for (int i = 0; i < 4; i++) {
      float temp;
      temp = m_whoObject->transMatrix[i][0] * C -
m_whoObject->transMatrix[i][1] * S;
      m_whoObject->transMatrix[i][1] = m_whoObject->transMatrix[i][0] * S +
m_whoObject->transMatrix[i][1] * C;
      m_whoObject->transMatrix[i][0] = temp;
   }
}
void CDrawScene::TransSpaceObject()
{
   Ccg20193DTransDoc *pDoc = (Ccg20193DTransDoc *)m_pView->GetDocument();
   // First Trans Object according to its matrix
   for (int i = 0; i < m_whoObject->polyCount; i++) {
      for (int j = 0; j < m_whoObject->objectSpace[i].polyCount; j++) {
         m_whoObject->objectSpace[i].transObject[j].x =
         m_whoObject->objectSpace[i].polyObject[j].x *
m_whoObject->transMatrix[0][0]+
         m_whoObject->objectSpace[i].polyObject[j].y *
m_whoObject->transMatrix[1][0]+
         m_whoObject->objectSpace[i].polyObject[j].z *
m_whoObject->transMatrix[2][0]+
```

```
            m_whoObject->transMatrix[3][0];
            m_whoObject->objectSpace[i].transObject[j].y =
            m_whoObject->objectSpace[i].polyObject[j].x *
m_whoObject->transMatrix[0][1]+
            m_whoObject->objectSpace[i].polyObject[j].y *
m_whoObject->transMatrix[1][1]+
            m_whoObject->objectSpace[i].polyObject[j].z *
m_whoObject->transMatrix[2][1]+
            m_whoObject->transMatrix[3][1];
            m_whoObject->objectSpace[i].transObject[j].z =
            m_whoObject->objectSpace[i].polyObject[j].x *
m_whoObject->transMatrix[0][2]+
            m_whoObject->objectSpace[i].polyObject[j].y *
m_whoObject->transMatrix[1][2]+
            m_whoObject->objectSpace[i].polyObject[j].z *
m_whoObject->transMatrix[2][2]+
            m_whoObject->transMatrix[3][2];
        }
    }
}
void CDrawScene::projectSpaceObject()
{
    Ccg20193DTransDoc *pDoc = (Ccg20193DTransDoc *)m_pView->GetDocument();
    // Project object to XOY plane
    for (int i = 0; i < m_whoObject->polyCount; i++) {
        if (m_whoObject->objectSpace[i].polyVisible) {
            for (int j = 0; j < m_whoObject->objectSpace[i].clipCount; j++) {
                m_whoObject->objectSpace[i].projectObject[j].x =
                    -(m_whoObject->objectSpace[i].clipObject[j].x - pDoc->eyeX)
/
```

```
                (m_whoObject->objectSpace[i].clipObject[j].z - pDoc->eyeZ)
*
                pDoc->eyeZ + pDoc->eyeX;
            m_whoObject->objectSpace[i].projectObject[j].y =
                -(m_whoObject->objectSpace[i].clipObject[j].y - pDoc->eyeY)
/
                (m_whoObject->objectSpace[i].clipObject[j].z - pDoc->eyeZ)
*
                pDoc->eyeZ + pDoc->eyeY;
            m_whoObject->objectSpace[i].projectObject[j].z = 0.0f;
         }
      }
   }
   // Then draw project result on XOY project plane.
   glColor3f(1.0f, 1.0f, 1.0f);
   for (int i = 0; i < m_whoObject->polyCount; i++) {
      if (m_whoObject->objectSpace[i].polyVisible) {
         glBegin(GL_LINE_STRIP);
         for (int j = 0; j < m_whoObject->objectSpace[i].clipCount; j++)
            glVertex3f(m_whoObject->objectSpace[i].projectObject[j].x,
               m_whoObject->objectSpace[i].projectObject[j].y,
               m_whoObject->objectSpace[i].projectObject[j].z);
         glEnd();
      }
   }
}
```