第 3 章

三维数据的二维投影

物体的空间位置以及形状、颜色与连接等数据存储在计算机中，这些数据唯一地代表着这个物体。给出物体的这些三维空间数据，在屏幕上绘制出该数据的二维投影图形，首先要给定投影平面与视点位置及视线方向；然后把空间点、线、面等投影到二维平面上；再把这个物体绘制出来。因为在屏幕上绘制物体只能绘制出该物体的一个投影面，至于绘制出物体的哪个（投影面）部分与（设定的）观察者所在位置有关。

三维数据的二维绘制，最困难的是确定哪些面（数据点）应该显示，哪些面（数据点）不显示，这个问题就是隐藏面处理问题，在本章中研究这一问题。

另外，绘制出的物体表面图必须具有亮度差别，否则该物体就不可见。这种亮度差别就是光照效果，利用颜色灰度来模拟光照效果在第 5 章中介绍。

3.1 三维数据投影

为了真实地表示三维物体，必须按照一定的规则把三维数据显示在二维平面上，这个过程在计算机图形学中叫作投影。目前常见的计算机屏幕只能显示二维图形，然后利用二维图形表达三维数据（形状）。

3.1.1 三维数据与二维显示

下面方程式表示三维空间的一条螺旋线（如图 3-1 所示）。

$$\begin{cases} x = \sin t \\ y = \cos t \quad 0 \leqslant t \leqslant \pi \\ z = t \end{cases}$$

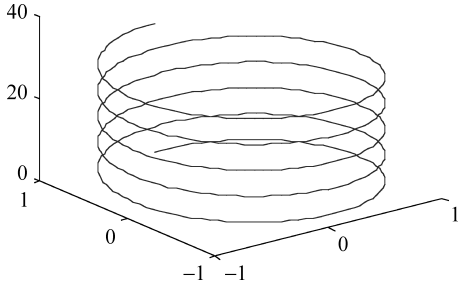


图 3-1 三维螺旋曲线

该螺旋线由三维空间的一些连续的点构成，其各个点的空间位置由三个坐标值（x，y，z）决定，因为使用了参数方程，所以 x，y，z 都是 t 的函数。

使用线（框）投影图也可以绘制出具有一定真实效果的曲面图形。下面的函数是一个曲面方程：

$$z = x\mathrm{e}^{(-x^2-y^2)} \quad -2 \leqslant x \leqslant 2, -2 \leqslant y \leqslant 2$$

x，y 都是间隔 0.1 取值，绘制出的各个点构成了一些（不连续的）曲线，这些曲线又构成了（线框）曲面，如图 3-2 所示。

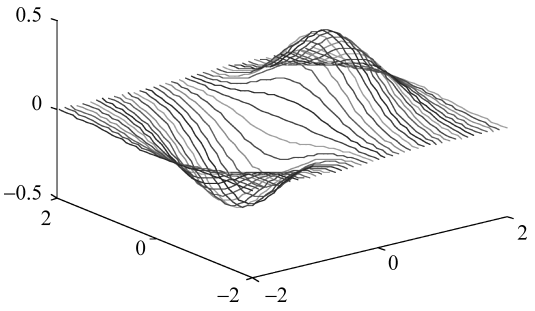


图 3-2　三维线框曲面

在计算机上绘制三维曲线，首先，计算机只能用离散来表达连续，这里 t 间隔某个较小的数取值；其次，计算机上没有“三维”的东西，所以需要投影。三维图形在计算机上都是凭借一个或者连续的多个二维图形表示，并且，一个平面上同一时刻只能绘制出一个投影面，绘制三维图形的说法本身并不严谨，应该说，绘制三维物体的某个角度上在某个平面上的投影图。图 3-1 与图 3-2 看上去是一个立体图，其实这里只是选择了从左上方观察（准确地说，这个图的视线方位角 −37.5°，水平角 30°）。

3.1.2　绘制空间直角坐标系

空间直角坐标系的三个坐标轴如果简化为由三个空间线段构成，这三个线段的起始坐标都是（0，0，0），终点坐标是（1，0，0），（0，1，0），（0，0，1），分别代表着 X、Y、Z 轴。那么，在绘制三维图形时，坐标系如何显示呢？这是首先要解决的问题。

图 3-1 与图 3-2 是用 MATLAB 绘制出来的，在 MATLAB 中，有绘制三维图形的函数，当调用这些函数进行绘图时，自动绘制出三维坐标系。而在 VC++ 中还没有这样的功能，所以首先需要写一个函数，用来绘制三维坐标系。

【例 3-1】　用一种投影的方法绘制三维坐标系。

这里使用一种斜平行投影方法，投影平面为 XOY 平面。

设定投影方向矢量为（x_p，y_p，z_p），形体上的一点模型坐标数据为（x，y，z），要确定该点在 XOY 平面上的投影（x_s，y_s），可以使用下面的公式计算。

$$\begin{cases} x_s = x - \dfrac{x_p}{z_p}z \\ y_s = y - \dfrac{y_p}{z_p}z \end{cases} \tag{3-1}$$

按照公式 3-1，把点(0,0,0)，(1,0,0)，(0,1,0)，(0,0,1)坐标代入表达式，就可以计算出每个(端)点应该绘制的位置，然后绘制(投影后的)线段，即坐标轴。

设计一个函数程序如下，把该函数放在视图文件中，OnDraw()函数的上面。

```
DrawAxis(float x,float y,float z,float xp,float yp,float zp,CDC * p)
{
    float xs,ys,zs;
    xs=x-xp/zp * z;
    ys=y-yp/zp * z;
    p->MoveTo(200,200);
    p->LineTo(xs * 200,ys * 200);
}
```

在 OnDraw()函数中写入下面的调用语句。

```
float t[3]={1.0,1.5,1.0};
DrawAxis(1.0,0.0,0.0,t[0],t[1],t[2],pDC);
DrawAxis(0.0,1.0,0.0,t[0],t[1],t[2],pDC);
DrawAxis(0.0,0.0,1.0,t[0],t[1],t[2],pDC);
```

运行结果如图 3-3(a)所示的坐标系。水平的为 X 轴，竖直的为 Y 轴，斜的为 Z 轴。

如果把投影向量变为下面所示，运行结果如图 3-3(b)所示的坐标系。

```
float t[3]={-1.8,1.5,1.0};
```

如果把投影向量变为下面所示，运行结果如图 3-3(c)所示的坐标系。

```
float t[3]={-1.8,-1.5,1.0};
```

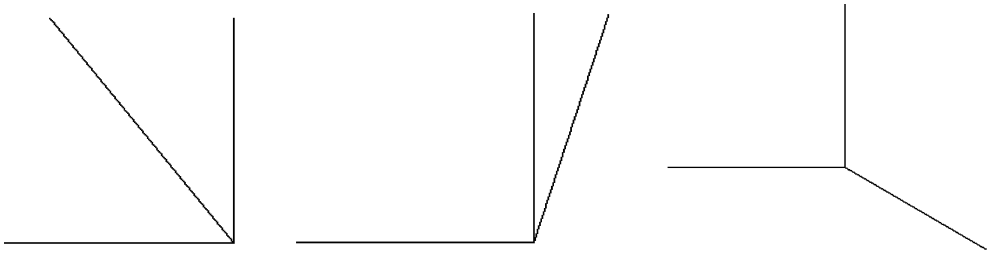


图 3-3　从不同角度投影的三个坐标系

从式 3-1 可以看出，是将三维点(x,y,z)变成了二维点(x_s,y_s)，在变换的过程中使用了投影方向矢量为(x_p,y_p,z_p)，借助于这个矢量实现了到二维的变换。对于 z 为 0 的点，变换后 x，y 的值不变；对于 z 不为 0 的点，当 x_p，y_p 不全为 0 时，z 值的大小影响新的 x，y 值。事实上，这个变换本质上是相似变换。

3.2 三维螺旋线的平行投影

在第1章例1-8中，绘制了三维螺旋线的三个投影图，即在三个坐标面上的投影图。这种投影比较简单，只绘制三维空间点的某两个坐标就可以。在这一节中将讨论如何绘制出如图3-1所示的具有真实感的投影图形。

3.2.1 参数方程及三维空间点的二维绘制

在图形学中，使用参数方程表示并绘制曲线与曲面更加方便，随着参数的变化，x，y，z也随着变化。因为x，y，z都是参数的函数，所以不需要利用x，y计算z等。

【例3-2】 绘制螺旋线。

把例3-1中的函数DrawCurve()修改为如下所示。

```
void DrawCurve(double p[3],CDC *pDC)
{
    double x[628],y[628],z[628];
    double xs[628],ys[628],a=120,b=1;
    int m=0;
    for(double t=0;t<62.8;t=t+0.1)
    {
        x[m]=a*cos(t);
        y[m]=a*sin(t);
        z[m]=b*t;
        m++;
    }
    xs[0]=x[0]-p[0]/p[2]*z[0]+200;
    ys[0]=y[0]-p[1]/p[2]*z[0]+200;
    pDC->MoveTo(xs[0],ys[0]);
    for(int i=0;i<628;i++)
    {
        xs[i]=x[i]-p[0]/p[2]*z[i]+200;
        ys[i]=y[i]-p[1]/p[2]*z[i]+200;
        pDC->LineTo((int)xs[i],(int)ys[i]);
    }
}
```

OnDraw()函数中的调用语句不变，绘制出如图3-4所示的螺旋线。根据投影向量，将计算出来的三维点再变换为二维点，实现了投影。

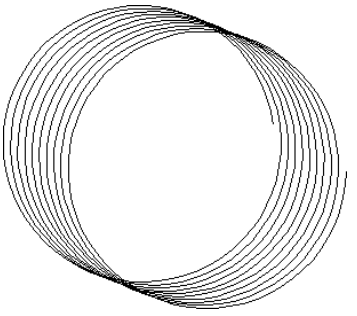


图3-4 螺旋线

利用这种方法可以绘制出其他三维曲线，如例3-3所示。

【例 3-3】 三维空间中的维维安尼(Viviani)曲线是一个球与柱体的交线,其一般方程如式 3-2 所示,其参数方程如式 3-3 所示,能够推导出式 3-2 与式 3-3 是等价的。下面给出系数,绘制该曲线的具有真实感的三维图形。

$$\begin{cases} x^2+y^2+z^2=a^2 \\ x^2+y^2-ax=0 \end{cases} \tag{3-2}$$

$$\begin{cases} x=a\cos^2\theta \\ y=a\cos\theta\sin\theta \quad (0\leqslant\theta<2\pi) \\ z=a\sin\theta \end{cases} \tag{3-3}$$

使用参数方程编写程序如下,放在 OnDraw()函数的上面,以备调用。

```
void DrawCurve(float p[3],CDC * pDC)
{
    float x[628],y[628],z[628];
    float xs[628],ys[628],a=120;
    int m=0;
    for(float t=0;t<6.28;t=t+0.01)
    {
        x[m]=a * cos(t) * cos(t);
        y[m]=a * cos(t) * sin(t);
        z[m]=a * sin(t);
        m++;
    }
    xs[0]=x[0]-p[0]/p[2] * z[0]+200;  //先计算第一点,然后把焦点移到该位置
    ys[0]=y[0]-p[1]/p[2] * z[0]+200;
    pDC->MoveTo(xs[0],ys[0]);
    for(int i=0;i<628;i++)
    {
        xs[i]=x[i]-p[0]/p[2] * z[i]+200;
        ys[i]=y[i]-p[1]/p[2] * z[i]+200;
        pDC->LineTo((int)xs[i],(int)ys[i]);
    }
}
```

在 OnDraw()函数中写入下面的语句进行调用。

```
float k[3]={1.0,-0.4,1.0};
    DrawCurve(k,pDC);
```

程序运行后,绘制出如图 3-5 所示的图形。

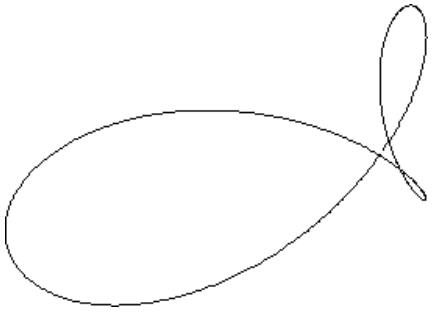


图 3-5　Viviani 曲线

3.2.2　不同角度的三维螺旋线投影

为了更好地理解三维数据在二维平面上的显示问题,下面设计程序,显示视向量不同时三

维螺旋线的投影图。

【例 3-4】 设计程序，绘制不同角度的三维螺旋线投影图。

设计函数如下，写在单文档程序的 OnDraw()函数上面。

```
void DrawCurve(double p[3],int s[2],CDC * pDC)
{
    double x[628],y[628],z[628];
    double xs[628],ys[628],a=60,b=1;
    int m=0;
    for(double t=0;t<62.8;t=t+0.1)
    {
        x[m]=a * cos(t);
        y[m]=a * sin(t);
        z[m]=b * t;
        m++;
    }
    xs[0]=x[0]-p[0]/p[2] * z[0]+200;
    ys[0]=y[0]-p[1]/p[2] * z[0]+250;
    pDC->MoveTo(xs[0]+s[0],ys[0]+s[1]);
    for(int i=0;i<628;i++)
    {
        xs[i]=x[i]-p[0]/p[2] * z[i]+200+s[0];
        ys[i]=y[i]-p[1]/p[2] * z[i]+250+s[1];
        pDC->LineTo((int)xs[i],(int)ys[i]);
    }
}
```

在 OnDraw()函数中写入下面的代码。

```
double k1[3]={0.2,0.5,0.3};
int s1[2]={0,0};
DrawCurve(k1,s1,pDC);
double k2[3]={0.6,0.2,-0.5};
int s2[2]={100,100};
DrawCurve(k2,s2,pDC);
double k3[3]={-0.5,0.2,0.6};
int s3[2]={200,0};
DrawCurve(k3,s3,pDC);
```

编译运行，绘制出如图 3-6 所示的三个投影图。

如果设定投影方向为 k1[3]={0.0,1.0,0.5}，那么绘制出的图形如图 3-7 所示，该图就近似于在平面中绘制一些圆，每个逐渐上移。

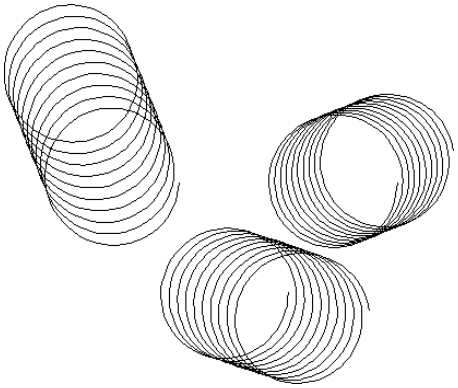


图 3-6　三维螺旋线的投影图

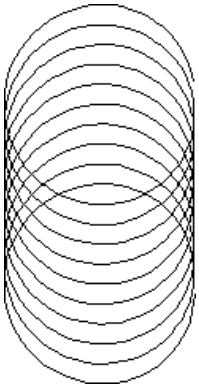


图 3-7　三维螺旋线的正立投影图

【思考题】 下面的数据是一个柱体的各个顶点坐标，绘制其在某个平面上的二维图形（投影图）。

$X=$

2.0000	0.6180	−1.6180	−1.6180	0.6180	2.0000
2.0000	0.6180	−1.6180	−1.6180	0.6180	2.0000

$Y=$

0	1.9021	1.1756	−1.1756	−1.9021	0
0	1.9021	1.1756	−1.1756	−1.9021	0

$Z=$

0	0	0	0	0	0
1	1	1	1	1	1

选择哪个投影平面对于显示效果也很重要，例如，有时把图形投影到与视线方向垂直的一个平面上，显示效果更加真实。

3.3　三维数据的透视投影

投影一般分为平行投影与透视投影两大类，3.2 节中使用的都是平行投影，平行投影简单，但是一般从显示效果上看，透视投影更加真实。

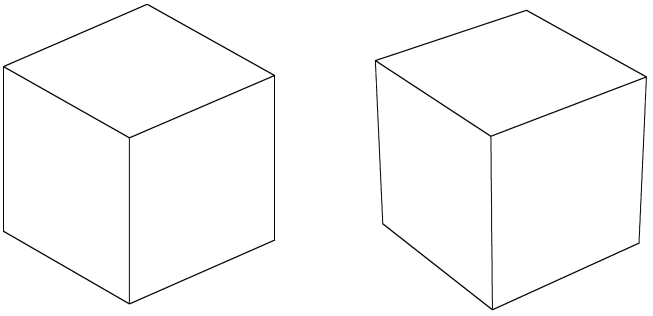
3.3.1　平行投影与透视投影

投影就是三维数据的二维显示。

投影一般分为平行投影与透视投影，正平行投影也称为正交投影。平行投影包括正平行投影与斜平行投影，透视投影分为一点透视与多点透视。

这里主要研究正平行投影与一点透视投影，简称平行投影与透视投影。

平行投影与透视投影是两种不同的投影方式。图 3-8(a)是一个正方体的平行投影，图 3-8(b)是一个正方体的透视投影。



(a) 平行投影　　(b) 透视投影

图 3-8　平行投影与透视投影

平行投影将物体所占有的空间投影成一个对边平行的直六面体，所以空间中的正方体也被投影成平行六面体。这种投影方式与相机的相对距离无关，相机的远近不影响投影物体的大小。所以，如果要保持物体的实际大小，一般使用这种投影方法。实际上，透视投影与视点位置以及视点到物体的距离有关。

3.3.2　观察坐标系下的一点透视投影

设视点为(a,b,c)，在引入一个（中间坐标系）观察坐标系后，视平面（投影平面）在观察方向上离视点的距离为z_s，令$u=\sqrt{a^2+b^2+c^2}$，$v=\sqrt{a^2+b^2}$，形体的顶点坐标为(x_w,y_w,z_w)，变换到观察坐标系下的坐标为(x_e,y_e,z_e)，则透视投影变换矩阵为：

$$[x_e \quad y_e \quad z_e \quad 1]=[x_w \quad y_w \quad z_w \quad 1]\begin{bmatrix} -b/v & -ac/uv & -a/u & 0 \\ a/v & -bc/uv & -b/u & 0 \\ 0 & v/u & -c/u & 0 \\ 0 & 0 & u & 1 \end{bmatrix} \tag{3-4}$$

展开后为：

$$\begin{aligned} x_e &= \frac{-b}{v}\cdot x_w+\frac{a}{v}\cdot y_w \\ y_e &= \frac{-ac}{uv}\cdot x_w-\frac{-bc}{uv}\cdot y_w+\frac{v}{u}\cdot z_w \\ z_e &= \frac{-a}{u}\cdot x_w-\frac{-b}{u}\cdot y_w+\frac{c}{u}\cdot z_w+u \end{aligned} \tag{3-5}$$

如果经透视投影到视平面(x_s,y_s)，那么透视投影公式为：

$$\begin{aligned} x_s &= x_e\cdot z_s/z_e \\ y_s &= y_e\cdot z_s/z_e \end{aligned} \tag{3-6}$$

【例 3-5】 绘制具有透视效果的螺旋线投影图。

利用式 3-5 和式 3-6 编写下面的函数。

```
void DrawCurve(double p[3],CDC * pDC)
{
```

```
    double x[628],y[628],z[628],xe[628],ye[628],ze[628];
    double xs[628],ys[628],zs=250,a,b,c,u,v;
    int m=0;
    a=p[0];b=p[1];c=p[2];
    u=sqrt(a*a+b*b+c*c);v=sqrt(a*a+b*b);
    for(double t=0;t<62.8;t=t+0.1)        //计算三维模型数据点坐标(用户坐标系)
    {
        x[m]=60*cos(t);
        y[m]=60*sin(t);
        z[m]=t;
        m++;
    }
    xe[0]=-b/v*x[0]+a/v*y[0];
    ye[0]=-a*c/(u*v)*x[0]-b*c/(u*v)*y[0]+v/u*z[0];
    ze[0]=-a/u*x[0]-b/u*y[0]-c/u*z[0]+u;
    xs[0]=xe[0]*zs/ze[0]+200;             //计算绘图的起始点坐标
    ys[0]=ye[0]*zs/ze[0]+250;
    pDC->MoveTo(xs[0],ys[0]);
    for(int i=0;i<628;i++)
    {
        xe[i]=-b/v*x[i]+a/v*y[i];         //计算三维模型数据点坐标(观察坐标系)
        ye[i]=-a*c/(u*v)*x[i]-b*c/(u*v)*y[i]+v/u*z[i];
        ze[i]=-a/u*x[i]-b/u*y[i]-c/u*z[i]+u;
        xs[i]=xe[i]*zs/ze[i]+200;         //计算投影到视平面上的点坐标
        ys[i]=ye[i]*zs/ze[i]+250;
        pDC->LineTo((int)xs[i],(int)ys[i]);
    }
}
```

在 OnDraw()函数中调用 DrawCurve()函数如下。

```
double k1[3]={200,150,100};               //视点坐标
DrawCurve(k1,pDC);
```

运行程序,绘制出如图 3-9(a)所示图形。

如果把视点坐标改为如下:

```
double k1[3]={-200,150,-10};
```

运行程序,绘制出如图 3-9(b)所示图形。

如果把观察方向上离视点的距离以及视点坐标改为如下:

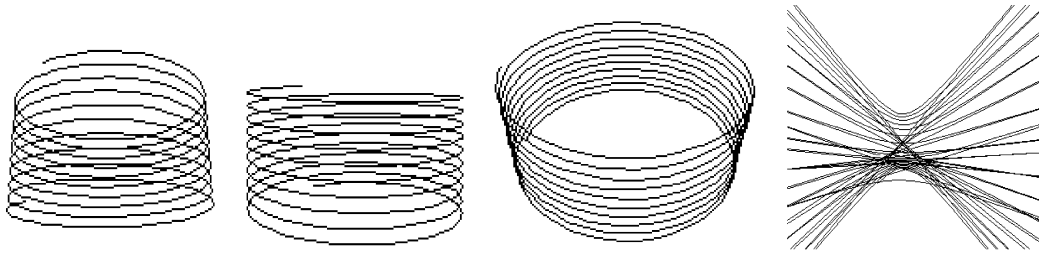
```
zs=200;double k1[3]={0,150,-100};
```

运行程序,绘制出如图 3-9(c)所示图形。

如果把观察方向上离视点的距离以及视点坐标改为如下:

```
zs=20;double k1[3]= {10,50,10};
```

运行程序，绘制出如图 3-9(d)所示图形。之所以出现这样的图形，是因为视点位于螺旋线之内造成的。



(a) (b) (c) (d)

图 3-9 螺旋线透视投影

如图 3-9(d)所示图形还没有全部粘贴过来，实际上也没有全部显示在屏幕上。普通的计算机屏幕只有(有限的)平面区域，特别是有时视角范围所限，所以需要对图形进行裁剪。

3.4 裁 剪

裁剪是从二维图形或者三维图形中剪切出需要(或者可见)的部分，裁剪可以在模型中进行，也可以在投影到平面后进行。

3.4.1 二维图形裁剪

二维图形裁剪与图像剪切是不一样的。图像是用像素等描述的，所以，图像剪切时使用数组下标就很容易把一幅图像剪切。二维图形是由一些几何图形构成的，几何图形本身以及几何图形之间有一定的关联，所以二维图形裁剪可以充分地利用图形之间的联系。

二维图形裁剪技术可以分为：直线段裁剪、多边形裁剪、曲线裁剪、字符裁剪等。

裁剪是因为人的视域具有有限性，裁剪后才可以真实地再现场景。

在裁剪时，要计算点边等是否位于视景体之内，端点都在视景体内的线段保留。对于一个端点在内，另一个端点在外的线段要计算交点，实现裁剪。

求交也是图形绘制时一个复杂的、必须进行的工作。

如果使用点集表示物体，那么裁剪工作简单容易，一般不需要求交计算；但是，如果使用线、多边形、曲线等表示物体，在裁剪时，就需要进行求交计算，以便决定哪一部分保留(进入视域)，哪一部分裁减掉(不进行绘制)。

3.4.2 三维图形裁剪

有时三维场景构造很大，物体很多，不是场景中所有物体都能进入可视范围，这时就需要裁剪。在三维图形处理过程中，裁剪是关键的一步。一般是先进行三维裁剪，然后对裁剪后保留下来的物体进行消隐，最后进行二维显示。

OpenGL 中的投影函数，例如 glOrtho()等能够实现自动裁剪。

```
void glOrtho (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top,
GLdouble near,GLdouble far)
```

该函数创建一个正交平行视景体。其中，近裁剪平面是一个矩形，矩形左下角点三维空间坐标是(left，bottom，－near)，右上角点是(right，top，－near)；远裁剪平面也是一个矩形，左下角点空间坐标是(left，bottom，－far)，右上角点是(right，top，－far)。所有的 near 和 far 值同时为正或同时为负。如果没有其他变换，投影的方向平行于 Z 轴，且视点朝向 Z 负轴。这意味着物体在视点前面时 far 和 near 都为负值，物体在视点后面时 far 和 near 都为正值。

该函数对位于视景体内的景物进行裁剪保留，之外的剪除。

如果把例 3-1 程序中的投影裁剪语句修改为下面所示：

```
glOrtho(-0.9,0.9,-0.9*(GLfloat)h/(GLfloat)w,0.9*(GLfloat)h/(GLfloat)w,
-10.0,10.0);
```

程序运行后，会绘制出一个被裁剪的球。

除了平行投影函数 glOrtho()外，OpenGL 中还提供了透视投影函数 glFrustum()。这个函数原型为：

```
void glFrustum(GLdouble left, GLdouble Right, GLdouble bottom, GLdouble top,
GLdouble near,GLdouble far);
```

该函数创建一个透视视景体，如图 3-10 所示。其参数定义近裁剪平面的左下角点和右上角点的三维空间坐标，即(left，bottom，－near)和(right，top，－near)。最后一个参数 far 是远裁剪平面的 Z 负值，远裁剪平面的左下角点和右上角点空间坐标由函数根据透视投影原理自动生成。near 和 far 表示离视点的远近，它们总为正值。

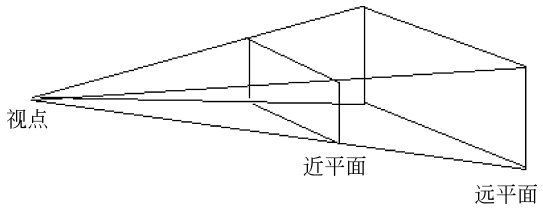


图 3-10　透视投影视景体示意图

透视方式不同，裁剪的效果也会不同。

3.5　视点变化下的多面体绘制

多面体指由顶点、边线、面构成的实体，在数学上，多面体是指由若干个平面多边形所围成的几何体。在计算机图形绘制过程中，更多的是使用多面体表示物体，而不是使用点集表示物体。计算机绘制过程中，没有真正的连续与光滑，都是近似的表示，在绝大多数

场合，使用多面体表示并绘制物体都可以达到视觉要求。

下面通过例题讨论多面体投影问题。

3.5.1 线框正方体投影绘制

上面的例题虽然是绘制坐标系、绘制三维螺旋线，但是具有一定的通用性。例如，可以修改其程序绘制平行投影下的真实感的立方体。

【例 3-6】 用平行投影方法绘制线框正方体。

设计一个函数程序如下，把该函数放在视图文件中，OnDraw()函数的上面。

```
void DrawCube(float x[8][3],float p[3],CDC * pDC)
{float xs[8],ys[8];
    for(int i=0;i<8;i++)
    {
    xs[i]=x[i][0]-p[0]/p[2]*x[i][2]+200;  //加 200 的目的是把图形移到窗口中心
    ys[i]=x[i][1]-p[1]/p[2]*x[i][2]+200;
    }
    pDC->MoveTo(xs[0],ys[0]);
    pDC->LineTo(xs[1],ys[1]);
    pDC->LineTo(xs[2],ys[2]);
    pDC->LineTo(xs[3],ys[3]);
    pDC->LineTo(xs[0],ys[0]);
    pDC->MoveTo(xs[4],ys[4]);
    pDC->LineTo(xs[5],ys[5]);
    pDC->LineTo(xs[6],ys[6]);
    pDC->LineTo(xs[7],ys[7]);
    pDC->LineTo(xs[4],ys[4]);
    pDC->MoveTo(xs[4],ys[4]);
    pDC->LineTo(xs[0],ys[0]);
    pDC->MoveTo(xs[5],ys[5]);
    pDC->LineTo(xs[1],ys[1]);
    pDC->MoveTo(xs[6],ys[6]);
    pDC->LineTo(xs[2],ys[2]);
    pDC->MoveTo(xs[7],ys[7]);
    pDC->LineTo(xs[3],ys[3]);
}
```

在 OnDraw()函数中写入下面的调用语句。

```
float t[3]={1.0,1.2,1.0};
float v[8][3]={{0,0,100},{100,0,100},{100,100,100},{0,100,100},
{0,0,0},{100,0,0},{100,100,0},{0,100,0}};
DrawCube(v,t,pDC);
```

运行结果如图 3-11(a)所示。

如果把投影向量变为如下所示，绘制出如图 3-11(b)所示的坐标系。

```
float t[3]={-1.0,1.2,1.0};
```

如果把投影向量变为如下所示，绘制出如图 3-11(c)所示的坐标系。

```
float t[3]={1.0,-0.4,1.0};
```

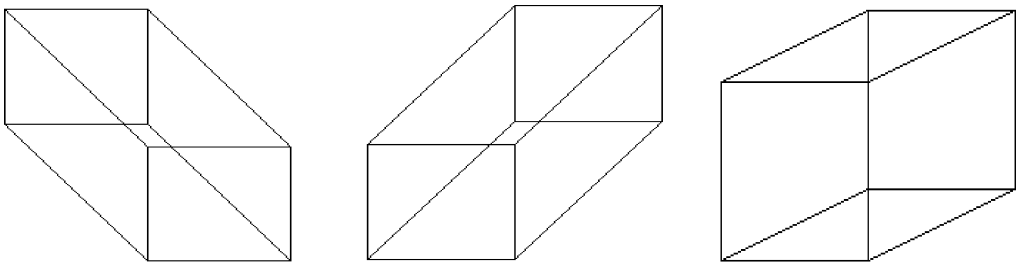


图 3-11　从不同角度投影到 *XOY* 面的正方体

因为是投影到 *XOY* 平面，所以，立方体看上去像一个(长高不等的)长方体。因为是投影到水平面，所以，有时投影后的长方体变得很长；事实上，应该修改该程序，投影到垂直于视线的平面上，这样更加真实。

上面介绍的是投影到 *XOY* 平面的平行投影，还可以投影到其他平面。另外，投影方式还有透视投影等，透视投影遵循近大远小的法则。

3.5.2　视点变化下的线框正方体绘制

改变视线方向，物体的投影会随之改变，利用这一原理可以制作出动画效果。

【例 3-7】　改进例 3-6 程序制作动画。

例 3-6 中函数 DrawCube()不变，在 OnDraw()函数中写入下面的语句。

```
float t[3];
float v[8][3]={{0,0,100},{100,0,100},{100,100,100},{0,100,100},
               {0,0,0},{100,0,0},{100,100,0},{0,100,0}};
for(int i=1;i<20;i++)
{
    t[0]=rand()/32767.0;
    t[1]=rand()/32767.0;
    t[2]=rand()/32767.0;
    CPen pen1,pen2;
    pen1.CreatePen(i,i,RGB(0,0,225));
    pDC->SelectObject(&pen1);
    DrawCube(v,t,pDC);
    Sleep(100);
    pen2.CreatePen(i,i,RGB(255,255,225));
```

```
    pDC->SelectObject(&pen2);
    DrawCube(v,t,pDC);
}
```

运行程序，随机绘制出一个又一个正方体（有时看上去像长方体）。

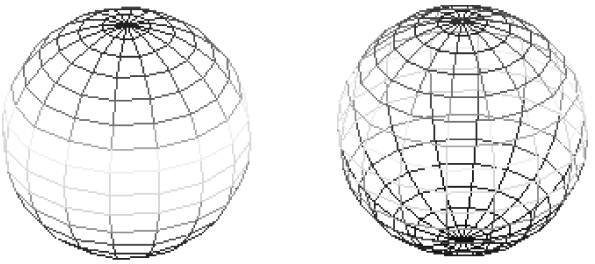
上面是改变投影方向制作动画，也可以随机改变顶点设计新的动画。

3.6 隐藏面检测

隐藏面检测也是三维物体数据二维绘制的关键一步。一般情况下，为了追求真实的效果，都要消除隐藏线（面），即被遮挡住的物体的部分。

3.6.1 隐藏线面

图 3-12(a)是消除影藏线的线框球，图 3-12(b)是没有消除隐藏线的球。



(a) 消除隐藏线　　(b) 显示隐藏线

图 3-12　网格球

对于线框图来说，不去掉隐藏线还可以近似地识别出物体；但是，对于用面表示的物体，如果不去掉隐藏面，那么便无法正确认识物体。

视点改变或者物体运动，都会导致隐藏面变化。也就是说，原先隐藏的面可能变为可见，原来可见的面可能隐藏起来。

在图形学中，一般每次绘制的时候，只绘制可见的线面，不绘制隐藏的线与面，实现真实感图形绘制。虽然要进行大量的计算，但是这项工作是必需的。

物体上哪部分可见主要决定于两个要素，一个是视点，一个是物体的形状以及各个面的位置关系。一种常用的检测方法是背面检测法。

3.6.2 一个正方体的六个面

下面以一个正方体为例，研究当给定视线方向时，其每个面是否可见。

【例 3-8】 一个正方体其顶点（坐标）依次为 1(1 1 1)，2(1 2 1)，3(2 2 1)，4(2 1 1)，5(1 1 2)，6(1 2 2)，7(2 2 2)，8(2 1 2)。六个面分别命名为 A，由 1，2，3，4 四个顶点围成；B，由 2，6，7，3 四个顶点围成；C，由 4，3，7，8 四个顶点围成；D，由 1，5，8，4 四个顶点围成；E，由 1，2，6，5 四个顶点围成；F，由 5，6，7，8 四个顶点围成；绘制出该正方体的示意图。

因为没有规定视线方向等，所以绘制示意图比较简单，如图 3-13 所示就是在某个视线方向上的投影图。

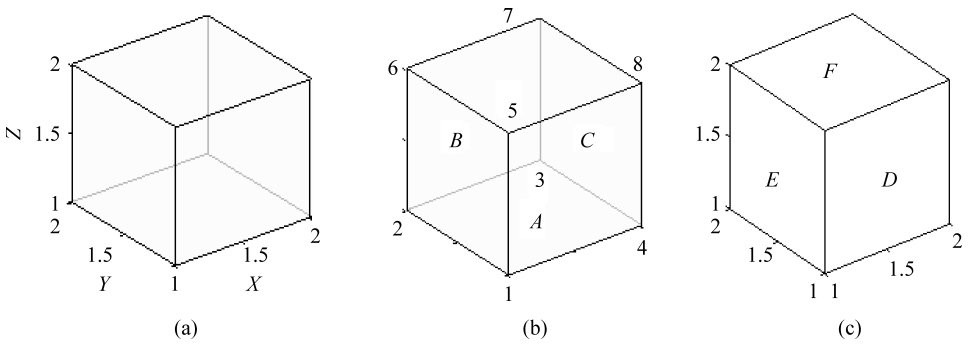


图 3-13　一个正方体的顶点与面

图 3-13(a)是带有坐标值的图形，可以根据该图得到每个顶点的坐标值。也可以大致看出，该图的视线方向大约沿着点(1,1,2)指向(1.8,2,1.4)，即视向量大约为{0.8,1,−0.6}。

在如图 3-13 所示情形下，面 A、面 B、面 C 不可见。

那么，对于给定的一个视向量 $\boldsymbol{V}=\{v_1,v_2,v_3\}$，如何计算出该正方体哪个面可见，哪个面不可见？

【思考题】 转换视点方向，最少能看见正方体的几个面？最多可以看见正方体的几个面？

3.6.3　背面检测方法

关于多面体隐藏面检测，有一个简单实用的方法，就是背面检测法。

以例 3-8 所示正方体为例，可以按照例 3-9 方法计算隐藏面。

【例 3-9】 当视点在远方，视向量为{−1,−1,−0.6}时，计算例 3-8 中的正方体哪个面可见。

在图形绘制过程中，需要根据数据进行计算，计算哪个面可见，然后显示出来。那么如何根据正方体的数据与视向量计算可见面？

关于这个正方体的可见面与隐藏面判断比较简单，因为其各个面的法向量方向(此处规定正方向指向物体外侧)容易得到，分别为：nA={0,0,−1};nB={0,1,0};nC={1,0,0};nD={0,−1,0};nE={−1,0,0};nF={0,0,1};。现在计算视向量与上述法向量的夹角的余弦值，如果该余弦值为正，那么夹角小于 90°，不可见；如果该余弦值为负，那么夹角大于 90°，可见。

计算可知：面 A、面 D、面 E 不可见，面 B、面 C、面 F 可见。

例 3-9 使用的判定方法叫作背面检测(Back-Face Detection)方法。

下面给出隐藏面的背面检测方法的描述：实体模型边界表示法为每个面规定了一个正方向，定义垂直于该面并且背离物体的方向为该面的正方向。

【算法 3-1】 凸多面体的隐藏面背面检测法。

对于单个凸多面体，可以按如下步骤计算不可见面。

（1）求一个面所在平面的法向量 $\boldsymbol{n}$。

（2）给定视线向量 $\boldsymbol{v}$。

（3）计算视线向量与法向量的数量积。

（4）根据 $\boldsymbol{n}$ 与 $\boldsymbol{v}$ 数量积的符号判断该面是否被遮挡，符号为正，被遮挡；符号为负，没有被遮挡。

另外，对于单个凸多面体，背面检测法可以完全检测出不可见面。但是，对于复杂物体或者多个物体，这种方法不能把隐藏面完全检测出来。这时，可以先使用这种简单高效的方法，然后再使用其他方法。

3.6.4 多面体的隐藏面计算

下面使用背面检测法计算一个多面体的面是否可见。

【例 3-10】 图 3-14 是一个多面体的示意图（投影图）。在计算机中存储着该多面体各个顶点的三维坐标，其中，A、B、C、D、E 的三维空间坐标依次为 $A(0,0,1)$，$B(0,-0.7,0.7)$，$C(0.7,0,0.7)$，$D(0,-1,0)$，$E(-1,0,0)$。

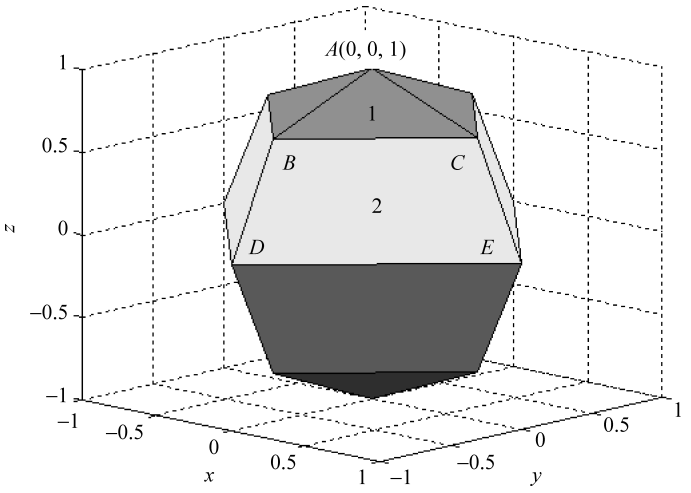


图 3-14 一个多面体

计算当视点在远方，视线方向是沿着向量$(-1,1,0.5)$的方向，那么面 1（面 ABC）是否可见？请通过计算说明。

首先求出向量 $\boldsymbol{AB}$ 与向量 $\boldsymbol{AC}$ 分别为$\{0,-0.7,-0.3\}$，$\{0.7,0,-0.3\}$，设面 ABC 的法向量为$\{x,y,z\}$，根据向量垂直内积为 0 得到的方程组：

$$\begin{cases}-0.7y-0.3z=0\\0.7x-0.3z=0\end{cases}$$

计算得到面 ABC 的一个指向外侧的法向量为：$\boldsymbol{N}_{ABC}=\{0.3,-0.3,0.7\}$，接下来计算视向量与 $\boldsymbol{N}_{ABC}$ 的夹角的余弦值：

$$\cos\alpha=\frac{(-1)\times(0.3)+1\times(-0.3)+0.5\times0.7}{\sqrt{(-1)^2+1^2+0.5^2}\ \sqrt{0.3^2+(-0.3)^2+0.7^2}}<0$$

所以，面 ABC 可见。

上面主要是介绍了背面检测法，实际上，还有一些常用的消除隐藏面线的方法。

3.6.5　其他检测方法

1. 深度缓冲器(Depth-Buffer Method)法

该算法用来检测被隐藏的点、线或面。这是一种算法简单、比较实用的方法。

【算法 3-2】　深度缓冲器消隐算法。

初始化二维数组 A(用来存储颜色)，屏幕上的点的位置作为数组下标，数组中的值设为背景色。

初始化二维数组 Z(用来存储深度)，屏幕上的点的位置作为数组下标，数组中的值设置成为 0。

```
for(各个多边形)
  {把该多边形投影到视平面(屏幕)上;
   for (多边形所覆盖的每个像素点(x,y))
     {计算多边形在该像素的深度值 Z(x,y);
      if (Z(x,y)大于数组 Z 在这个位置上原有的值)
          {把 Z(x,y)存入数组 Z,替换该位置原先的值
           把多边形在(x,y)处的颜色值存入数组 A;
          }
     }
}
```

上面的算法假设物体投影到 XOY 平面上。

多边形在各个像素点处的深度值可以从顶点的深度值用增量方法算得。

在默认的视点下，使用深度缓冲器方法。

(1) 给出投影平面，这里根据方位角与仰角计算出投影平面的法线。再参考多边形的顶点坐标确定投影平面的具体位置，也就是给出平面方程。这里把投影平面放到柱体的后面。

(2) 从第一个面开始，计算每个面上的点到投影平面的距离，保留距离大的点的位置及其颜色。

(3) 把最后保留下来的颜色在相应的点的位置上绘制出来。

程序设计时，首先编写一个函数 fplane()用来计算平面方程。该函数的输入是视点的方位角与仰角，还有各个顶点坐标的最值。

然后，编写函数 fdist()用来计算一个多边形上每个点到投影平面的距离。该函数输入是多边形顶点，还有投影平面的参数。

还要编写一个函数 fin()用来完成屏幕点颜色的替换。

深度缓冲器法简单实用，但是在计算时需要很大的存储空间，另外也没有充分地利用物体本身的一些连接关系。

2. 画家方法

这种方法按照多边形离观察者的远近建立一张深度优先级表，距离观察者近的优先级高。

如果空间中各个多边形都可以区分出远近，那么，先把最远处的多边形绘制在屏幕上，然后从远到近绘制其他多边形。

如果投影区域重叠，便使用近处的多边形颜色覆盖先绘制的远处的多边形。

建立深度优先级表是关键的一步。在建立优先级表时，一个复杂的问题是多边形互相覆盖或者互相贯穿。也就是两个多边形A与B，A上有的部分距离观察者远于B上有的部分，但是A上有的部分距离观察者近于B上有的部分。如果检测到这种情况，就需要对多边形进行分割，然后对分割后的多边形进行排序。

除了上面介绍的三种消隐方法外，还有区域细分（Warnock）算法、扫描线算法等多种消隐方法，可以参考其他有关资料。

习　　题

一、程序修改题

1. 修改例3-1程序，为每个坐标轴加上箭头，并标上 X,Y,Z 与原点 O。

2. 如果将式3-1改成下面式3-7：

$$\begin{cases} x_s = x - \dfrac{x_p}{z_p + x_p} z \\ y_s = y - \dfrac{y_p}{z_p + y_p} z \end{cases} \tag{3-7}$$

修改例3-1程序，绘制出投影后的三个坐标轴。

3. 修改例3-3，改变视向量，从多个角度观察该三维曲线，进一步制作出动画效果。

4. 运行例3-4程序，改变视向量会发现有时绘制结果有些失真，修改式3-1，然后再修改程序，观察实验结果。

5. 将 $u=\sqrt{a^2+b^2+c^2}$ 与 $v=\sqrt{a^2+b^2}$ 修改为 $u=|a|+|b|+|c|$ 与 $v=|a|+|b|$，然后修改程序，观察分析。

6. 修改例3-6程序，使得绘制正方体线框图时，不绘制出被隐藏的边。

7. 修改例3-6程序，绘制正方体表面实体图，即每次绘制出可见的各个面，每个面使用不同的颜色填充。

8. 修改例3-7程序，有规律地修改视点位置，例如，视点位置在一个圆上慢慢运动，然后根据视点位置改变，制作正方体投影变化的动画效果。

9. 修改例3-7程序，令视点不变，长（正）方体的顶点变化制作动画。

二、计算题

1. 例3-2程序是将三维点投影到二维平面，然后用画线函数连接起来，问绘制出的前5个点（即前4个折线段的5个端点）分别是什么？

2. 例3-3绘制的第一个点与最后一个点分别是什么？

3. 当投影向量为 float $t[3]=\{-1.8,1.5,1.0\}$ 时，使用式3-1计算(1,1,1)点投影到平面的什么位置，即在屏幕的什么位置绘制。如果投影向量变为 $\boldsymbol{t}[3]=\{-1.8,-1.5,$

1.0}，根据式 3-1 计算(1,1,1)点应该在什么位置绘制。

4. 在 3.2 节的思考题中，给出了一个柱体的顶点坐标，如果视点在远处，视线方向是(1,1,1)指向(0,0,0)，根据式 3-1 绘制出该棱柱各个边的投影图，然后分析、计算判断：如果是实心不透明柱体，哪些棱与顶点是看不到的。

5. 如果在与视线垂直平面上投影(平行投影)，计算第 4 题中各个顶点投影后的二维点坐标值。

6. 如果在图 3-10 中，视点位置为(0,0,0)，近平面(四边形)的 4 个顶点分别为(2,−3,3)，(2,3,3)，(2,3,−3)，(2,−3,−3)，近平面与远平面之间的距离为 1，求远平面(四边形)四个顶点的坐标。

7. 式 3-1 是计算平行投影的公式，式 3-5 和式 3-6 是计算透视投影的公式，现在给定投影向量为{−1.8,1.5,1.0}，视点位置为(2,2,2)，再给定一个空间四边形的四个顶点(1,0,1)，(1,1,1)，(0,1,1)，(0,0,1)，用两种投影方法计算投影到 XOY 平面上的新四边形的四个顶点坐标，然后绘制出来，对比分析。

8. 例 3-10 的条件不变，计算面 $BCED$ 是否可见。

9. 图 3-15 是视线方向位于水平角 30°，方位角−37.5°时的投影图，该视线方向转换为向量是{0.5272,0.6871,−0.5000}，计算这种情况下，如图 3-15 所示多面体的面 1 与面 2 的向外的法向量与视线向量的数量积符号。

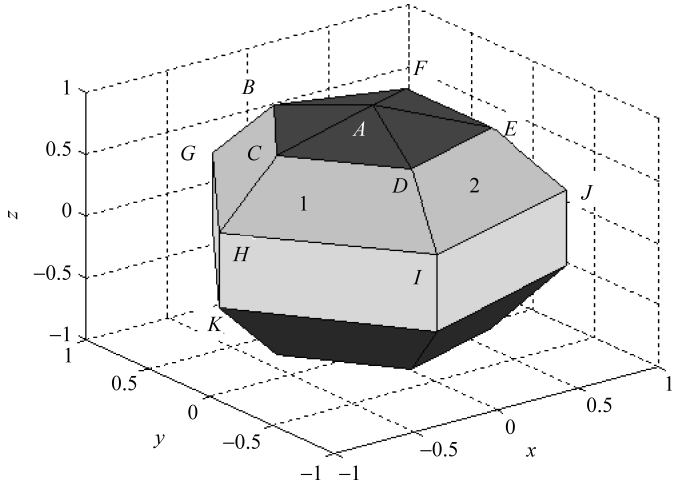


图 3-15　一个凸多面体

$$\boldsymbol{X}=\begin{matrix}
0 & 0 & 0 & 0 & 0 & 0\\
-0.5878 & -0.1816 & 0.4755 & 0.4755 & -0.1816 & -0.5878\\
-0.9511 & -0.2939 & 0.7694 & 0.7694 & -0.2939 & -0.9511\\
-0.9511 & -0.2939 & 0.7694 & 0.7694 & -0.2939 & -0.9511\\
-0.5878 & -0.1816 & 0.4755 & 0.4755 & -0.1816 & -0.5878\\
0 & 0 & 0 & 0 & 0 & 0
\end{matrix}$$

$\boldsymbol{Y}=$

0	0	0	0	0	0
0	−0.5590	−0.3455	0.3455	0.5590	0
0	−0.9045	−0.5590	0.5590	0.9045	0
0	−0.9045	−0.5590	0.5590	0.9045	0
0	−0.5590	−0.3455	0.3455	0.5590	0
0	0	0	0	0	0

$\boldsymbol{Z}=$

−1.0000	−1.0000	−1.0000	−1.0000	−1.0000	−1.0000
−0.8090	−0.8090	−0.8090	−0.8090	−0.8090	−0.8090
−0.3090	−0.3090	−0.3090	−0.3090	−0.3090	−0.3090
0.3090	0.3090	0.3090	0.3090	0.3090	0.3090
0.8090	0.8090	0.8090	0.8090	0.8090	0.8090
1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

上面数据是MATLAB中的表示方法，意思是，最上面5个点汇聚成一点，即A点，坐标是(0,0,1)；A、D、I等位于两条边线的重合处，坐标分别为(0,0,1)，(−0.5878,0,0.8090)，(−0.9511,0,0.8090)，数据选择时，按列从下到上。

10. 先根据图3-15找到面1与面2的四个顶点，然后根据背面检测法，判断当视线方向为(1,1,1)指向(0,0,0)时，面1与面2是否可见。

11. 继续题10(如图3-15所示多面体的数据)，计算当视向量为{−1,−1,1}时，面1与面2是否可见。

12. 读图3-13，然后完成下面各题。

(1) 写出8个顶点的坐标。

(2) 写出6个面的指向物体外侧的法向量。

(3) 当视向量为{0.8,1.0,−0.6}时，计算视向量与6个面的法向量(指向外侧)的乘积。

13. 参考算法3-2，当视向量为{0.8,1.0,−0.6}时，计算例3-8中正方体顶点(数据)相对于XOY平面的深度值。

三、程序设计题

1. 参考例1-8与例3-3，编写程序绘制出维维安尼(Viviani)曲线在三个坐标面上的正投影。

2. 参考例3-3，编写程序，绘制函数图像$z=0.85x+0.04y+0.01$与面片$z=-0.04x+0.85y+1.6$，其中，$0\leqslant x\leqslant 1$，$0\leqslant y\leqslant 1$。

3. 参考图3-10，设计程序，输入视点位置、近平面(多边形)四个顶点坐标、远平面(多边形)四个顶点坐标，另外输入一个空间中的点，判断该点是否位于裁剪空间之内。

4. 参考例3-10，设计程序，输入是该多面体的顶点(坐标数据)、视向量方向，输出是每个面是否可见。

5. 设计程序，以一个长方体为对象，实现深度缓冲器算法。