**案例45-圆锥Phong明暗处理算法**

文档编写：霍波魏

校稿/修订：孔令德

时间2019~2020

联系方式：QQ997796978

**说明：**本套案例由孔令德开发，原版本为Visual C++6.0，配套于孔令德的著作《计算机图形学-基于MFC三维图形开发》一书。孔令德计算机工程研究所的学生霍波魏在学习计算机图形学期间，对本套案例进行了升级并编写了学习文档。现在程序的编写和程序的解释都是基于Windows 10操作系统，使用Microsoft visual studio 2017平台的MFC（英文版）开发。

1. **案例描述**

本案例通过PhongShader着色器对圆锥进行着色。圆锥分为侧面与底面两部分。底面为三角形面片，侧面大部分为四边形网格，在靠近顶点处的面片为三角形面片。与圆柱面类似，圆锥的底面用角度来进行面片划分，而侧面用周向夹角和纵向间距来划分面片。

1. **知识点**

本案例通过PhongShader对立方体进行明暗处理。具体的PhongShader步骤如下：

（1）计算三角形网格顶点的平均法向量，即



式中，Ni为共享顶点的三角形网格的面法向量，N为点法向量。

（2）按照扫描线顺序使用线性插值计算三角形网格边界上每一点的法向量。

（3）在扫描线与三角形相交跨度内，使用线性插值算法计算三角形内每一点的法向量。

（4）首先对三角形网格内的每一点使用法向量调用光照模型计算获得的光强，然后将光强分解为RGB三原色的颜色值。需要注意的是，插值后的法向量也需要归一化为单位向量，才能用于光强计算。

Phong采用双线性插值计算多边形内一点f处的法向量，如图45-1所示：

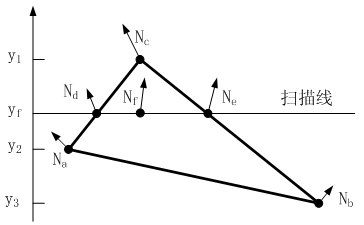


图45-1 点法向量的双线性插值



与多边形顶点的坐标联系起来，有



1. **实现步骤**
2. 添加基础类与添加绘制立方体的CCone类。
3. 在CProjection类中进行透视变化。
4. 在CLightSource类中对光源参数进行初始化，在CMaterial类中对材质属性进行初始化，在CLighting类中对光强进行计算。
5. 在CCone类中计算顶点坐标、读入面表，绘制图形。
6. 在CTestView类中调用CCone类的绘制函数。
7. 在CTestView类中添加消息响应函数，在OnDraw中调用DoubleBuffer函数。
8. **主要算法**

1. CCone类：

public:

CCone();

virtual~CCone();

void ReadVertex();//读入点表

void ReadFace();//读入面表

void Draw(CDC\* pDC);//绘制物体

void CalNormal();//计算法矢量

public:

CP3d \*V;//点表

CFace \*\*F;//面表

CVector3 \*N;

double h;//圆锥高度

double r;//圆锥底面半径

int N1, N2;//N1为周向网格,N2纵向网格

CLighting \*pLight;//光照环境

CMaterial \*pMaterial;//物体材质

int nClientWidth;//屏幕客户区宽度

int nClientHeight;//屏幕客户区高度

int nHWidth, nHHeight;//屏幕客户区的半宽和半高

CProjection projection;

CCone::CCone()

{

}

CCone::~CCone()

{

if (pLight != NULL)

{

delete pLight;

pLight = NULL;

}

if (pMaterial != NULL)

{

delete pMaterial;

pLight = NULL;

}

if (V != NULL)

{

delete[]V;

V = NULL;

}

if (N != NULL)

{

delete[]N;

N = NULL;

}

for (int n = 0; n < N2 + 1; n++)//注意撤销次序,先列后行,与设置相反

{

delete[]F[n];

F[n] = NULL;

}

delete[]F;

F = NULL;

}

void CCone::ReadVertex()//点表

{

r = 260;//圆锥底面半径

h = 400;//圆锥的高

int cTheta = 10;//面片夹角

N1 = 360 / cTheta;//N1周向网格

int cNum = 10;//纵向间距

N2 = ROUND(h / cNum);//N2为纵向网格数

V = new CP3d[N1 \* N2 + 2];

N = new CVector3[N1 \* N2 + 2];

double cTheta1, cNum1;

V[0].x = 0; V[0].y = 0; V[0].z = 0;//圆锥底面中心

for (int i = 0; i < N2; i++)//纵向

{

cNum1 = i \* cNum;

for (int j = 0; j < N1; j++)//周向

{

cTheta1 = j \* cTheta \* PI / 180;

double r1 = (N2 - i) \* r / N2;

V[i \* N1 + j + 1].x = r1 \* cos(cTheta1);

V[i \* N1 + j + 1].y = i \* cNum;

V[i \* N1 + j + 1].z = r1 \* sin(cTheta1);

}

}

V[N1 \* N2 + 1].x = 0; V[N1 \* N2 + 1].y = h; V[N1 \* N2 + 1].z = 0;//圆锥顶点

}

void CCone::ReadFace()//面表

{

//设置二维动态数组

F = new CFace \*[N2 + 1];//设置行

for (int n = 0; n < N2 + 1; n++)

{

F[n] = new CFace[N1];//设置列

}

for (int j = 0; j < N1; j++)//构造底部三角形面片

{

int tempj = j + 1;

if (tempj == N1) tempj = 0;//面片的首尾连接

int BottomIndex[3];//底部三角形面片索引号数组

BottomIndex[0] = 0;

BottomIndex[1] = j + 1;

BottomIndex[2] = tempj + 1;

F[0][j].SetNum(3);

for (int k = 0; k < F[0][j].vN; k++)//传入面中点的索引

F[0][j].vI[k] = BottomIndex[k];

}

for (int i = 1; i < N2; i++)//构造圆锥侧面的四边形面片

{

for (int j = 0; j < N1; j++)

{

int tempi = i + 1;

int tempj = j + 1;

if (tempj == N1) tempj = 0;

int BodyIndex[4];//圆锥体四边形面片索引号数组

BodyIndex[0] = (i - 1) \* N1 + j + 1;

BodyIndex[1] = (tempi - 1) \* N1 + j + 1;

BodyIndex[2] = (tempi - 1) \* N1 + tempj + 1;

BodyIndex[3] = (i - 1) \* N1 + tempj + 1;

F[i][j].SetNum(4);

for (int k = 0; k < F[i][j].vN; k++)

F[i][j].vI[k] = BodyIndex[k];

}

}

for (int j = 0; j < N1; j++)//构造侧面顶角附近的三角形面片

{

int tempj = j + 1;

if (tempj == N1) tempj = 0;

int TopIndex[3];//顶部三角形面片索引号数组

TopIndex[0] = N1 \* N2 + 1;

TopIndex[1] = (N1 \* (N2 - 1)) + tempj + 1;

TopIndex[2] = (N1 \* (N2 - 1)) + j + 1;

F[N2][j].SetNum(3);

for (int k = 0; k < F[N2][j].vN; k++)

F[N2][j].vI[k] = TopIndex[k];

}

}

void CCone::CalNormal()//计算顶点法矢量

{

for (int i = 0; i < N2 + 1; i++)//周向

{

for (int j = 0; j < N1; j++)//纵向

{

//计算顶点的平均法矢量

int Beforei = i - 1, Afteri = i + 1;

int Beforej = j - 1, Afterj = j + 1;

if (0 == i) continue;

if (0 == j) Beforej = N1 - 1;

if (N2 + 1 == Afteri) continue;

if (N1 == Afterj) Afterj = 0;

CVector3 vN0, vN1, AveN;//相邻2个面片的法矢量及平均法矢量

CVector3 vEdge01(V[Beforei \* N1 + Afterj + 1], V[Beforei \* N1 + j + 1]);

CVector3 vEdge02(V[Beforei \* N1 + Afterj + 1], V[i \* N1 + j + 1]);

vN0 = Cross(vEdge01, vEdge02);

CVector3 vEdge11(V[Beforei \* N1 + j + 1], V[Beforei \* N1 + Beforej + 1]);

CVector3 vEdge12(V[Beforei \* N1 + j + 1], V[i \* N1 + Beforej + 1]);

vN1 = Cross(vEdge11, vEdge12);

AveN = (vN0 + vN1) / AveN.Mag();//顶点法矢量的平均值

N[i \* N1 + j + 1] = AveN;

}

}

}

void CCone::Draw(CDC \*pDC)//绘制球面

{

CalNormal();

CZBuffer \*zbuf = new CZBuffer;//申请内存

zbuf->InitDeepBuffer(800, 800, 1000);//初始化深度缓冲器

zbuf->GetScreen(nHWidth, nHHeight);

CPi3 Point3[3];//底面与顶面三角形顶点数组

CVector3 Normal3[3];//底面与顶面三角形法矢量数组

CPi3 Point4[4];//侧面四边形顶点数组

CVector3 Normal4[4];//侧面四边形法矢量数组

for (int i = 0; i < N2 + 1; i++)

{

for (int j = 0; j < N1; j++)

{

CVector3 ViewVector(V[F[i][j].vI[0]], projection.ViewPoint);//面的视矢量

ViewVector = ViewVector.Normalize();//单位化视矢量

F[i][j].SetFaceNormal(V[F[i][j].vI[0]], V[F[i][j].vI[1]], V[F[i][j].vI[2]]);//计算小面片法矢量

F[i][j].fNormal.Normalize();//单位化法矢量

if (Dot(ViewVector, F[i][j].fNormal) >= 0)

{

if (3 == F[i][j].vN)//处理三角形面片

{

for (int m = 0; m < F[i][j].vN; m++)

{

projection.PerProject(V[F[i][j].vI[m]]);

Point3[m] = projection.ScreenP;

Normal3[m] = F[i][j].fNormal;

}

zbuf->SetPoint(Point3, Normal3, 3);//初始化

zbuf->CreateBucket();//创建桶表

zbuf->CreateEdge();//创建边表

zbuf->Phong(pDC, projection.ViewPoint, pLight, pMaterial);//填充三角形

zbuf->ClearMemory();

}

else//处理四边形面片

{

for (int m = 0; m < F[i][j].vN; m++)

{

projection.PerProject(V[F[i][j].vI[m]]);

Point4[m] = projection.ScreenP;

Normal4[m] = N[F[i][j].vI[m]];

}

zbuf->SetPoint(Point4, Normal4, 4);//初始化

zbuf->CreateBucket();//创建桶表

zbuf->CreateEdge();//创建边表

zbuf->Phong(pDC, projection.ViewPoint, pLight, pMaterial);//填充四边形

zbuf->ClearMemory();

}

}

}

}

delete zbuf;

}

2.CLighting类：

public:

CLighting();

CLighting(int);

virtual ~CLighting();

void SetLightNumber(int);//设置光源数量

CRGB Illuminate(CP3d, CP3d, CVector3, CMaterial \*);//计算光照

public:

int LightNum;//光源数量

CLightSource \*Light;//光源数组

CRGB Ambient;//环境光

CLighting:: CLighting()

{

LightNum = 1;

Light = new CLightSource[LightNum];

Ambient = CRGB(1.0, 1.0, 1.0);//环境光恒定不变

}

CLighting::~ CLighting()

{

if (Light)

{

delete[]Light;

Light = NULL;

}

}

void CLighting::SetLightNumber(int lnum)

{

if (Light)

delete[]Light;

LightNum = lnum;

Light = new CLightSource[lnum];

}

CLighting:: CLighting(int lnum)

{

LightNum = lnum;

Light = new CLightSource[lnum];

Ambient = CRGB(0.3, 0.3, 0.3);

}

CRGB CLighting:: Illuminate(CP3d ViewPoint, CP3d Point, CVector3 Normal, CMaterial \*pMaterial)

{

CRGB ResultI = pMaterial->M\_Emit;//材质自身发散色为初始值

for (int i = 0; i < LightNum; i++)//来自光源

{

if (Light[i].L\_OnOff)

{

CRGB I = CRGB(0.0, 0.0, 0.0);

CVector3 VL(Point, Light[i].L\_Position);//指向光源的矢量

double d = VL.Mag();//光传播的距离，等于矢量VL的模

VL = VL.Normalize();//单位化光矢量

CVector3 VN = Normal;

VN = VN.Normalize();//单位化法矢量

//第1步，加入漫反射光

if (Light[i].b\_Diffuse)

{

double CosTheta = max(Dot(VL, VN), 0);

I += Light[i].L\_Diffuse\*pMaterial->M\_Diffuse\*CosTheta;

}

//第2步，加入镜面反射光

if (Light[i].b\_Specular)

{

CVector3 VV(Point, ViewPoint);//VV为视矢量

VV = VV.Normalize();//单位化视矢量

CVector3 VH = (VL + VV) / (VL + VV).Mag();//平分矢量

double nHN = pow(max(Dot(VH, VN), 0), pMaterial->M\_n);

I += Light[i].L\_Specular\*pMaterial->M\_Specular\*nHN;

}

//第3步，光强衰减

double c0 = Light[i].L\_C0;//c0为常数衰减因子

double c1 = Light[i].L\_C1;//c1线性衰减因子

double c2 = Light[i].L\_C2;//c2二次衰减因子

double f = (1.0 / (c0 + c1 \* d + c2 \* d\*d));//光强衰减函数

f = min(1.0, f);

ResultI += I \* f;

}

else

ResultI += Point.c;//物体自身颜色

}

//第4步，加入环境光

if (Light[0].b\_Ambient)

ResultI += Ambient \* pMaterial->M\_Ambient;

//第5步，颜色归一化到[0,1]区间

ResultI.Normalize();

//第6步，返回所计算顶点的光强颜色

return ResultI;

}

3.CTestView类：

public:

void DoubleBuffer(CDC\* pDC);//双缓冲绘图

void DrawObject(CDC\* pDC);//绘制物体

void InitialLightingScene(void);

protected:

BOOL bPlay;//动画开关

CTransform3 tran;//变换对象

int LightNum;//光源数量

CCone cone;

void CTestView::InitialLightingScene(void)

{

LightNum = 1;//光源个数

cylinder.pLight = new CLighting(LightNum);//一维光源动态数组

cylinder.pLight->Light[0].SetPosition(0, 0, 800);//设置光源位置坐标

for (int i = 0; i < LightNum; i++)

{

cylinder.pLight->Light[i].L\_Diffuse = CRGB(1.0, 1.0, 1.0);//光源的漫反射颜色

cylinder.pLight->Light[i].L\_Specular = CRGB(1.0, 1.0, 1.0);//光源镜面高光颜色

cylinder.pLight->Light[i].L\_C0 = 1.0;//常数衰减系数

cylinder.pLight->Light[i].L\_C1 = 0.0000001;//线性衰减系数

cylinder.pLight->Light[i].L\_C2 = 0.00000001;//二次衰减系数

cylinder.pLight->Light[i].L\_OnOff = TRUE;//光源开启

}

cylinder.pMaterial = new CMaterial;//一维材质动态数组

cylinder.pMaterial->SetAmbient(CRGB(0.175, 0.012, 0.012));//材质对环境光光的反射率

cylinder.pMaterial->SetDiffuse(CRGB(0.614, 0.041, 0.041));//材质对漫反射光的反射率

cylinder.pMaterial->SetSpecular(CRGB(1.0, 1.0, 1.0));//材质对镜面反射光的反射率

cylinder.pMaterial->SetEmit(CRGB(0.2, 0.0, 0.0));//材质自身发散的颜色

cylinder.pMaterial->SetExp(30);//高光指数

}

void CTestView::DoubleBuffer(CDC\* pDC)//双缓冲

{

CRect rect;//定义客户区

GetClientRect(&rect);//获得客户区的大小

cylinder.nClientWidth = rect.Width();//屏幕客户区宽度

cylinder.nClientHeight = rect.Height();//屏幕客户区高度

cylinder.nHWidth = cylinder.nClientWidth / 2;//屏幕客户区半宽

cylinder.nHHeight = cylinder.nClientHeight / 2;//屏幕客户区半高

CDC memDC;

memDC.CreateCompatibleDC(pDC);

CBitmap NewBitmap, \*pOldBitmap;

NewBitmap.CreateCompatibleBitmap(pDC, cylinder.nClientWidth, cylinder.nClientHeight);

pOldBitmap = memDC.SelectObject(&NewBitmap);

//memDC.FillSolidRect(rect,pDC->GetBkColor());

DrawObject(&memDC);

pDC->BitBlt(0, 0, cylinder.nClientWidth, cylinder.nClientHeight, &memDC, 0, 0, SRCCOPY);

memDC.SelectObject(pOldBitmap);

NewBitmap.DeleteObject();

}

void CTestView::DrawObject(CDC \* pDC)

{

cone.Draw(pDC);

}

1. **实现效果**

圆锥Phong明暗处理如图45-2所示。

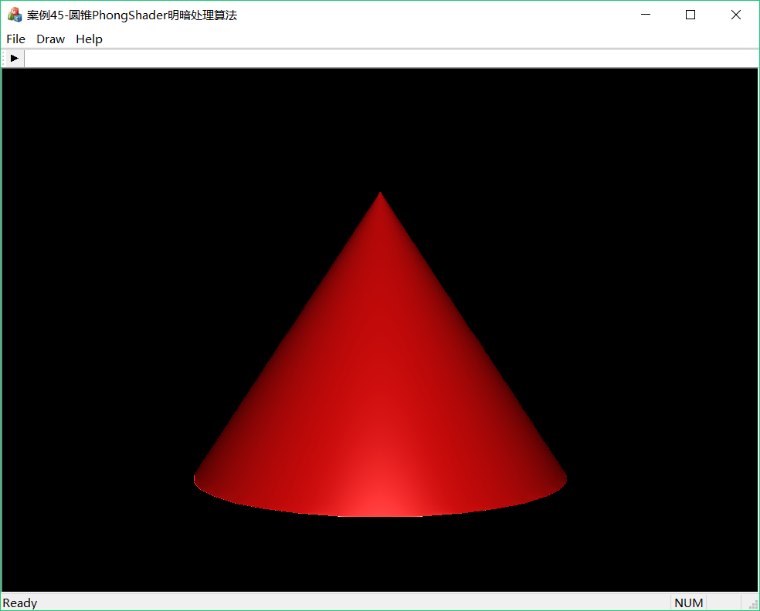


图45-2 圆锥Phong明暗处理效果图