**案例42-球体Gouraud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. **知识点**

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

（1）计算三角形网格顶点的平均法向量。在图42-1所示的三角形网格中，顶点*P*被*n*（*n* = 8）个三角形共享。*P*点的法向量***N***应取共享*P*点的所有三角形网格的表面法向量***N****i*的平均值，即



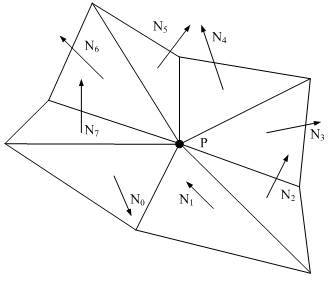


图42-1 计算点法向量

式中，***N****i*为共享顶点*P*的三角形网格的面法向量，***N***为点法向量。

（2）对三角形网格的每个顶点调用光照模型计算获得的光强。

（3）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光强。

（4）在扫描线与三角形相交跨度内，首先使用线性插值算法计算三角形内每一点的光强，然后将光强分解为RGB三原色的颜色值。

GouraudShader采用双线性插值算法计算多边形内一点*f*处的光强，如图42-2所示：



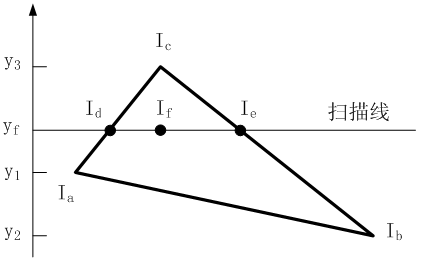


图42-2 光强的双线性插值

与三角形顶点坐标联系起来，有



GouraudShader容易与扫描线算法结合起来，计算三角形网格内各点的光强。

1. **实现步骤**
2. 添加基础类与添加绘制立方体的CSphere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CSphere类中计算顶点坐标、读入面表，绘制图形，调用光照函数计算光照。
6. 在CTestView类中调用CSphere类的绘制函数。
7. 在CTestView中添加消息响应函数，在OnDraw中调用DoubleBuffer函数。
8. **主要算法**

1. CSphere类：

public:

CSphere();

virtual~CSphere();

void ReadVertex();//读入点表

void ReadFace();//读入面表

void CalVertexColor();//计算顶点光照

void Draw(CDC\* pDC);//绘制物体

public:

CP3d \*V;//球的顶点一维数组

CFace \*\*F;//面的二维数组

int N1, N2;//N1为纬度区域,N2为经度区域

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;

CSphere::CSphere()

{

V = NULL; F = NULL;

}

CSphere::~CSphere()

{

if (pLight != NULL)

{

delete pLight;

pLight = NULL;

}

if (pMaterial != NULL)

{

delete pMaterial;

pMaterial = NULL;

}

delete[]V;

for (int i = 0; i < N1; i++) {

delete[] F[i];

//防止野指针的产生

F[i] = nullptr;

}

delete[] F;

F = nullptr;

}

void CSphere::ReadVertex()//点表

{

int gAlpha = 4, gBeta = 4;//面片夹角

N1 = 180 / gAlpha, N2 = 360 / gBeta;//N1为纬度区域,N2为经度区域

V = new CP3d[(N1 - 1) \* N2 + 2];//V为球的顶点

//纬度方向除南北极点外有"N1－1"个点，"2"代表南北极两个点

double gAlpha1, gBeta1, r = 300;//r为球体半径

//计算北极点坐标

V[0].x = 0, V[0].y = r, V[0].z = 0;

//按行循环计算球体上的点坐标

for (int i = 0; i < N1 - 1; i++)

{

gAlpha1 = (i + 1) \* gAlpha \* PI / 180;

for (int j = 0; j < N2; j++)

{

gBeta1 = j \* gBeta \* PI / 180;

V[i \* N2 + j + 1].x = r \* sin(gAlpha1) \* sin(gBeta1);

V[i \* N2 + j + 1].y = r \* cos(gAlpha1);

V[i \* N2 + j + 1].z = r \* sin(gAlpha1) \* cos(gBeta1);

}

}

//计算南极点坐标

V[(N1 - 1) \* N2 + 1].x = 0, V[(N1 - 1) \* N2 + 1].y = -r, V[(N1 - 1) \* N2 + 1].z = 0;

}

void CSphere::ReadFace()//面表

{

//设置二维动态数组

F = new CFace \*[N1];//设置行

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

F[n] = new CFace[N2];//设置列

for (int j = 0; j < N2; j++)//构造北极三角形面片

{

int tempj = j + 1;

if (tempj == N2) tempj = 0;//面片的首尾连接

int NorthIndex[3];//北极三角形面片索引号数组

NorthIndex[0] = 0;

NorthIndex[1] = j + 1;

NorthIndex[2] = tempj + 1;

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

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

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

}

for (int i = 1; i < N1 - 1; i++)//构造球面四边形面片

{

for (int j = 0; j < N2; j++)

{

int tempi = i + 1;

int tempj = j + 1;

if (tempj == N2) tempj = 0;

int BodyIndex[4];//球面四边形面片索引号数组

BodyIndex[0] = (i - 1) \* N2 + j + 1;

BodyIndex[1] = (tempi - 1) \* N2 + j + 1;

BodyIndex[2] = (tempi - 1) \* N2 + tempj + 1;

BodyIndex[3] = (i - 1) \* N2 + 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 < N2; j++)//构造南极三角形面片

{

int tempj = j + 1;

if (tempj == N2) tempj = 0;

int SouthIndex[3];//南极三角形面片索引号数组

SouthIndex[0] = (N1 - 2) \* N2 + j + 1;

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

SouthIndex[2] = (N1 - 2) \* N2 + tempj + 1;

F[N1 - 1][j].SetNum(3);

for (int k = 0; k < F[N1 - 1][j].vN; k++)

F[N1 - 1][j].vI[k] = SouthIndex[k];

}

}

void CSphere::CalVertexColor()//计算顶点颜色

{

for (int nVertex = 0; nVertex < (N1 - 1) \* N2 + 2; nVertex++)//遍历所有点

{

CVector3 VNormal(V[nVertex]);//点的位置矢量代表共享该点的所有面的平均法矢量

V[nVertex].c = pLight->Illuminate(projection.ViewPoint, V[nVertex], VNormal, pMaterial);//调用光照函数

}

}

void CSphere::Draw(CDC \*pDC)//绘制球面

{

CalVertexColor();

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];//南北极顶点数组

CPi3 Point4[4];//球体顶点数组

CVector3 Normal3[3]; //南北极顶点的法矢量数组

CVector3 Normal4[4]; //南北极顶点的法矢量数组

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

{

for (int j = 0; j < N2; 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]]);

F[i][j].fNormal.Normalize();//单位化法矢量

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

{

if (3 == F[i][j].vN)//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;

}

zbuf->SetPoint(Point3, 3);//初始化

zbuf->CreateBucket();//创建桶表

zbuf->CreateEdge();//创建边表

zbuf->Gouraud(pDC);//颜色渐变填充三角形

zbuf->ClearMemory();

}

else//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;

}

zbuf->SetPoint(Point4, 4);//初始化

zbuf->CreateBucket();//创建桶表

zbuf->CreateEdge();//创建边表

zbuf->Gouraud(pDC);//颜色渐变填充三角形

zbuf->ClearMemory();

}

}

}

}

delete zbuf;//释放内存

}

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

CCube cube;

void CTestView::InitialLightingScene(void)

{

cube.LightNum = 1;//光源个数

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

cube.pLight->Light[0].SetPosition(0, 0, 200);//设置光源位置坐标

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

{

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

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

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

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

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

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

}

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

cube.pMaterial->SetAmbient(CRGB(0.247, 0.200, 0.075));//材质对环境光光的反射率

cube.pMaterial->SetDiffuse(CRGB(0.752, 0.606, 0.226));//材质对漫反射光的反射率

cube.pMaterial->SetSpecular(CRGB(0.628, 0.556, 0.366));//材质对镜面反射光的反射率

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

cube.pMaterial->SetExp(50);//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

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

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

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

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

CDC memDC;

memDC.CreateCompatibleDC(pDC);

CBitmap NewBitmap, \*pOldBitmap;

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

pOldBitmap = memDC.SelectObject(&NewBitmap);

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

DrawObject(&memDC);

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

memDC.SelectObject(pOldBitmap);

NewBitmap.DeleteObject();

}

void CTestView::DrawObject(CDC \* pDC)

{

sphere.Draw(pDC);

}

1. **实现效果**

球体Gouraud明暗处理如图42-3所示。

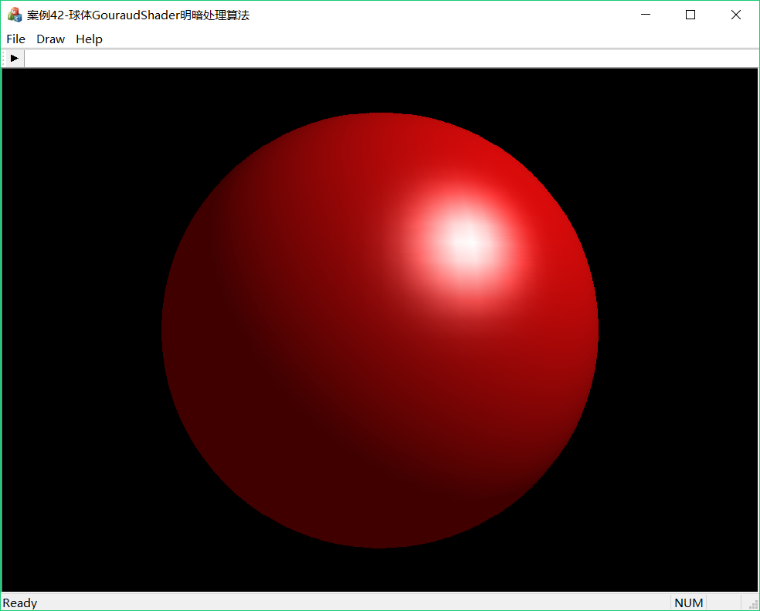


图42-3 球体Gouraud明暗处理效果图

1. **知识补充**

GouraudShader计算三角形网格每个顶点的光强，三角形内部使用3个顶点的光强进行线性插值，高光处仍能看到四边形网格的白色边界线。