**案例40-立方体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40-1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



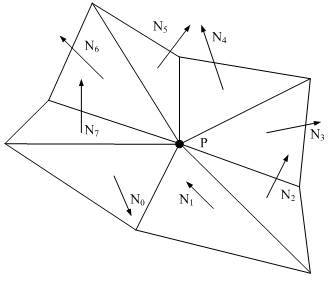


图40-1 计算点法向量

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40-2所示：



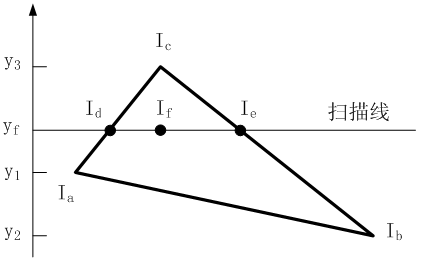


图40-2 光强的双线性插值

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CCube类。
3. 在CCube类中计算顶点坐标、读入面表，绘制图形。
4. 再CProjection类中进行透视变化。
5. 再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。
6. 在CTestView中添加消息响应函数，在OnDraw中调用DoubleBuffer函数。
7. **主要算法**

1. CCube类：

public:

CCube();

virtual~CCube();

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[8];//点表

CFace F[6];//面表

int LightNum;//光源数量

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;

CCube::CCube()

{

}

CCube::~CCube()

{

if (pLight != NULL)

{

delete pLight;

pLight = NULL;

}

if (pMaterial != NULL)

{

delete pMaterial;

pMaterial = NULL;

}

}

void CCube::ReadVertex()//点表

{

//顶点的三维坐标(x,y,z)

int a = 200;//立方体边长为2a

V[0].x = -a; V[0].y = -a; V[0].z = -a;

V[1].x = +a; V[1].y = -a; V[1].z = -a;

V[2].x = +a; V[2].y = +a; V[2].z = -a;

V[3].x = -a; V[3].y = +a; V[3].z = -a;

V[4].x = -a; V[4].y = -a; V[4].z = +a;

V[5].x = +a; V[5].y = -a; V[5].z = +a;

V[6].x = +a; V[6].y = +a; V[6].z = +a;

V[7].x = -a; V[7].y = +a; V[7].z = +a;

}

void CCube::ReadFace()//面表

{

F[0].SetNum(4); F[0].vI[0] = 4; F[0].vI[1] = 5; F[0].vI[2] = 6; F[0].vI[3] = 7;//前面顶点索引

F[1].SetNum(4); F[1].vI[0] = 0; F[1].vI[1] = 3; F[1].vI[2] = 2; F[1].vI[3] = 1;//后面顶点索引

F[2].SetNum(4); F[2].vI[0] = 0; F[2].vI[1] = 4; F[2].vI[2] = 7; F[2].vI[3] = 3;//左面顶点索引

F[3].SetNum(4); F[3].vI[0] = 1; F[3].vI[1] = 2; F[3].vI[2] = 6; F[3].vI[3] = 5;//右面顶点索引

F[4].SetNum(4); F[4].vI[0] = 2; F[4].vI[1] = 3; F[4].vI[2] = 7; F[4].vI[3] = 6;//顶面顶点索引

F[5].SetNum(4); F[5].vI[0] = 0; F[5].vI[1] = 1; F[5].vI[2] = 5; F[5].vI[3] = 4;//底面顶点索引

}

void CCube::CalVertexColor()//计算顶点颜色

{

for (int nVertex = 0; nVertex < 8; 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 CCube::Draw(CDC \*pDC)//绘制球面

{

CalVertexColor();

CPi3 Point[4];//面的顶点坐标

CVector3 Normal4[4];

CZBuffer \*zbuf = new CZBuffer;

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

zbuf->GetScreen(nHWidth, nHHeight);

for (int nFace = 0; nFace < 6; nFace++)

{

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

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

F[nFace].SetFaceNormal(V[F[nFace].vI[0]], V[F[nFace].vI[1]], V[F[nFace].vI[2]]);

F[nFace].fNormal.Normalize();//单位化法矢量

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

{

for (int nVertex = 0; nVertex < F[nFace].vN; nVertex++)//边循环

{

projection.PerProject(V[F[nFace].vI[nVertex]]);

Point[nVertex] = projection.ScreenP;

Normal4[nVertex] = F[nFace].fNormal;

}

zbuf->SetPoint(Point, 4);//初始化

zbuf->CreateBucket();//创建桶表

zbuf->CreateEdge();//创建边表

zbuf->Gouraud(pDC);

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

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)

{

cube.Draw(pDC);

}

1. **实现效果**

立方体Gouraud明暗处理如图40-3所示。

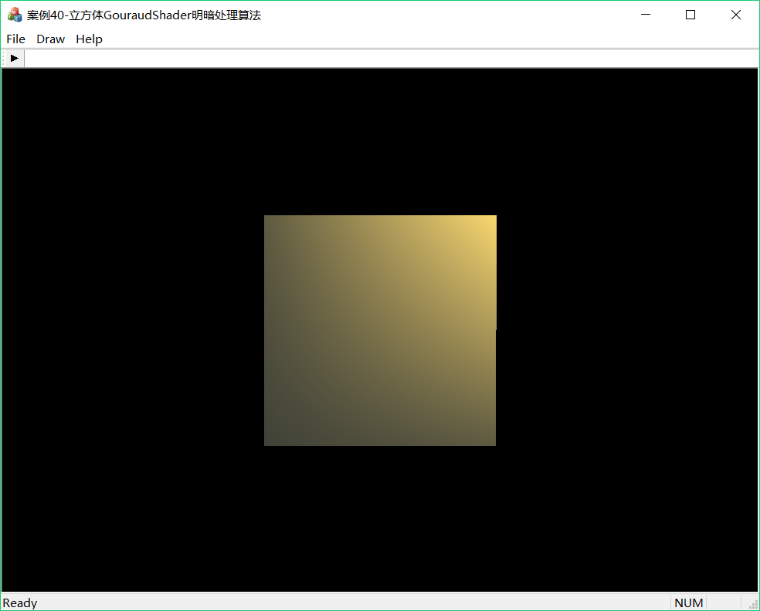


图40-3 立方体Gouraud明暗处理效果图

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。对于立方体而言，GouraudShader只能在立方体的边角处产生高光，无法在立方体面的面上产生高光。