**案例55-球体几何纹理映射反走样算法**

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**

本案例通过双线性内插反走样算法绘制光照凹凸球面三维动画。

1. **知识点**

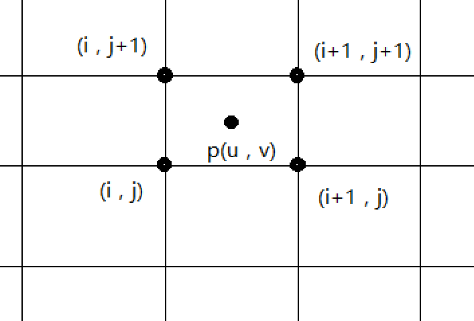
双线性内插反走样算法是常用的反走样方法之一，双线性内插对于目的像素，需要取4个邻近的像素进行双线性内插，缩放后的图像质量高，不会出现像素不连续的情况。对于一个目的像素，设其通过透视投影变换得到的坐标为(u,v)，u和v位于[0，1]区间内。则目的像素的颜色可由原图形中的坐标为(i，j)、(i+1，j)、(i，j+1)、(i+1，j+1)所对应的4个像素的颜色来决定，即

图55-1 双线性插值示意图



式中，为(i，j)处的像素值。双线性内插过程如图55-1所示。

1. **实现步骤**
2. 添加基础类与添加绘制球体的CSphere类。
3. 在CSphere类中计算顶点坐标、读入面表，绘制图形，并绑定为例坐标，调用CZbuffer类中的Phong函数对球面的三维顶点坐标进行双线性纹理插值。
4. 在CTexture类中取目标位图的长×宽为。对颜色位图和高度场位图进行双线性内插。
5. 为球面上每一点读取高度场位图的扰动值，计算并将扰动量和叠加到小面内各点的法矢量上，并做归一化处理。
6. 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。
7. 在CProjection类中进行透视变化。
8. 在CTestView中添加消息响应函数，在OnDraw中调用DoubleBuffer函数。
9. **主要算法**

1. CSphere类

public:

CSphere();

virtual~CSphere();

void ReadVertex();//读入顶点坐标

void ReadFace();//读入面坐标

void Draw(CDC \*);//绘制球面

public:

CP3d \*V;//圆柱顶点一维数组

CT2 \*T;//圆柱纹理一维数组

CFace \*\*F;//面的二维数组

int N1, N2;//N1为经度区间,N2为纬度区间

int gAlpha, gBeta;//纬度角与经度角

CLighting \*pLight;//光照环境

CMaterial \*pMaterial;//物体材质

int r;//球体半径

CProjection projection;

CTexture texture;

CSphere::CSphere()

{

r = 300;//r球面半径

texture.nWidth = Round(2 \* PI\*r);//目的图像的宽度

texture.nHeight = Round(PI\*r);//目的图像的高度

}

CSphere::~CSphere()

{

if (pLight != NULL)

{

delete pLight;

pLight = NULL;

}

if (pMaterial != NULL)

{

delete pMaterial;

pLight = NULL;

}

if (V != NULL)

{

delete[]V;

V = NULL;

}

if (T != NULL)

{

delete[]T;

T = NULL;

}

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

{

delete[] F[n];

F[n] = NULL;

}

delete[]F;

F = NULL;

}

void CSphere::ReadVertex()//点表

{

gAlpha = 4, gBeta = 8;//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];

T = new CT2[(N1 - 1)\*N2 + 2];

double gAlpha1, gBeta1;

//计算北极点坐标

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

T[0].u = 0; T[0].v = 0;//闲置

//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);

T[i\*N2 + j + 1].u = gBeta1 / (2 \* PI)\*(texture.nWidth - 1);

T[i\*N2 + j + 1].v = (PI - gAlpha1) / PI \* (texture.nHeight - 1);

}

}

//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;

T[(N1 - 1)\*N2 + 1].u = 0; T[(N1 - 1)\*N2 + 1].v = 0;//闲置

}

void CSphere::ReadFace()//读入面表

{

//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::Draw(CDC \*pDC)//绘制球面

{

CZBuffer \*zbuf = new CZBuffer;//申请内存

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

zbuf->ReadGradient(texture.Imgx\_Gradient, texture.Imgy\_Gradient);

CPi3 Point3[3];//底面与顶面三角形顶点数组

CT2 Texture3[3];//底面与顶面三角形纹理数组

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

CT2 Texture4[4];//侧面四边形纹理数组

CVector3 Normal4[4];//侧面四边形法矢量数组

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]]);//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] = CVector3(V[F[i][j].vI[m]]);

}

double tempi = i + 1, tempj = j + 1;//对三角形面片进行特殊处理

Texture3[0].u = gBeta \* (j) / 360.0\*(texture.nWidth - 1); Texture3[0].v = (1.0 - gAlpha \* i / 180.0)\*(texture.nHeight - 1);

Texture3[1].u = gBeta \* (j) / 360.0\*(texture.nWidth - 1); Texture3[1].v = (1.0 - gAlpha \* tempi / 180.0)\*(texture.nHeight - 1);

Texture3[2].u = gBeta \* tempj / 360.0\*(texture.nWidth - 1); Texture3[2].v = (1.0 - gAlpha \* i / 180.0)\*(texture.nHeight - 1);

zbuf->SetPoint(Point3, Normal3, Texture3, 3);//初始化

zbuf->CreateBucket();//创建桶表

zbuf->CreateEdge();//创建边表

zbuf->Phong(pDC, projection.ViewPoint, pLight, pMaterial, texture.LImage1);//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] = CVector3(V[F[i][j].vI[m]]);

Texture4[m] = T[F[i][j].vI[m]];

}

if (N2 - 1 == j)//消除图像纹理的接缝

{

Texture4[2].u = texture.nWidth - 1;

Texture4[3].u = texture.nWidth - 1;

}

zbuf->SetPoint(Point4, Normal4, Texture4, 4);//初始化

zbuf->CreateBucket();//创建桶表

zbuf->CreateEdge();//创建边表

zbuf->Phong(pDC, projection.ViewPoint, pLight, pMaterial, texture.LImage1);//填充四边形

zbuf->ClearMemory();

}

}

}

}

delete zbuf;

}

2.CTexture类

public:

CTexture();

virtual~CTexture();

void ReadImage();//读入颜色纹理

void ReadBumpMap();//读入凹凸纹理

void BiLinear\_BMP(COLORREF \*\*, int, int, COLORREF \*\*, int, int);//双线性插值

public:

COLORREF \*\*Image1, \*\*Image2, \*\*LImage1, \*\*LImage2;//二维动态数组

BITMAP bmp1, bmp2;//BITMAP结构体变量

BYTE \*im;

double \*\*Imgx\_Gradient, \*\*Imgy\_Gradient;

int nWidth, nHeight;//图像的宽度和高度

CTexture::CTexture()

{

}

CTexture::~CTexture()

{

for (int n = 0; n < bmp1.bmHeight; n++)

{

delete[]Image1[n];

Image1[n] = NULL;

}

delete[]Image1;

Image1 = NULL;

for (int n = 0; n < bmp2.bmHeight; n++)

{

delete[]Image2[n];

Image2[n] = NULL;

}

delete[]Image2;

Image2 = NULL;

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

{

delete[]LImage1[n];

LImage1[n] = NULL;

delete[]LImage2[n];

LImage2[n] = NULL;

delete Imgx\_Gradient[n];

Imgx\_Gradient[n] = NULL;

delete Imgy\_Gradient[n];

Imgy\_Gradient[n] = NULL;

}

delete[]LImage1;

LImage1 = NULL;

delete[]LImage2;

LImage2 = NULL;

delete[]Imgx\_Gradient;

Imgx\_Gradient = NULL;

delete[]Imgy\_Gradient;

Imgy\_Gradient = NULL;

}

void CTexture::ReadImage()//读入颜色纹理

{

CBitmap NewBitmap;

NewBitmap.LoadBitmap(IDB\_CLRTEXTURE);//导入DDB位图

NewBitmap.GetBitmap(&bmp1);//将CBitmap的信息保存到Bitmap结构体中

int nbytesize = bmp1.bmWidthBytes\*bmp1.bmHeight;

im = new BYTE[nbytesize];

NewBitmap.GetBitmapBits(nbytesize, (LPVOID)im);

Image1 = new COLORREF\*[bmp1.bmHeight];

for (int n1 = 0; n1 < bmp1.bmHeight; n1++)

Image1[n1] = new COLORREF[bmp1.bmWidth];

for (int n1 = 0; n1 < bmp1.bmHeight; n1++)

{

for (int n2 = 0; n2 < bmp1.bmWidth; n2++)

{

int pos = n1 \* bmp1.bmWidthBytes + 4 \* n2;//颜色分量位置

n1 = bmp1.bmHeight - 1 - n1;//位图从左下角向右上角绘制

Image1[n1][n2] = RGB(im[pos + 2], im[pos + 1], im[pos]);

}

}

delete[]im;

LImage1 = new COLORREF\*[nHeight];

for (int n1 = 0; n1 < nHeight; n1++)

LImage1[n1] = new COLORREF[nWidth];

BiLinear\_BMP(LImage1, nWidth, nHeight, Image1, bmp1.bmWidth, bmp1.bmHeight);

}

void CTexture::ReadBumpMap()//读入凹凸纹理

{

CBitmap NewBitmap;

NewBitmap.LoadBitmap(IDB\_BUMPTEXTURE);//导入DDB位图

NewBitmap.GetBitmap(&bmp2);//将CBitmap的信息保存到Bitmap结构体中

int nbytesize = bmp2.bmWidthBytes\*bmp2.bmHeight;

im = new BYTE[nbytesize];

NewBitmap.GetBitmapBits(nbytesize, (LPVOID)im);

Image2 = new COLORREF\*[bmp2.bmHeight];

for (int n1 = 0; n1 < bmp2.bmHeight; n1++)

Image2[n1] = new COLORREF[bmp2.bmWidth];

for (int n1 = 0; n1 < bmp2.bmHeight; n1++)

{

for (int n2 = 0; n2 < bmp2.bmWidth; n2++)

{

int pos = n1 \* bmp2.bmWidthBytes + 4 \* n2;//颜色分量位置

n1 = bmp2.bmHeight - 1 - n1;//位图从左下角向右上角绘制

Image2[n1][n2] = RGB(im[pos + 2], im[pos + 1], im[pos]);

}

}

LImage2 = new COLORREF\*[nHeight];

for (int n1 = 0; n1 < nHeight; n1++)

LImage2[n1] = new COLORREF[nWidth];

BiLinear\_BMP(LImage2, nWidth, nHeight, Image2, bmp2.bmWidth, bmp2.bmHeight);

Imgx\_Gradient = new double\*[nHeight];

Imgy\_Gradient = new double\*[nHeight];

for (int n1 = 0; n1 < nHeight; n1++)

{

Imgx\_Gradient[n1] = new double[nWidth];

Imgy\_Gradient[n1] = new double[nWidth];

}

for (int n1 = 0; n1 < nHeight; n1++)

{

for (int n2 = 0; n2 < nWidth; n2++)

{

int fontx, backx, fonty, backy;//一阶中心差分

fontx = n1 + 1; backx = n1 - 1;

fonty = n2 + 1; backy = n2 - 1;

//检测图片的边界，防止越界

if (backx < 0)

backx = 0;

if (backy < 0)

backy = 0;

if (fontx > nHeight - 1)

fontx = nHeight - 1;

if (fonty > nWidth - 1)

fonty = nWidth - 1;

Imgx\_Gradient[n1][n2] = -(GetRValue(LImage2[n1][fonty]) - GetRValue(LImage2[n1][backy]));

Imgy\_Gradient[n1][n2] = -(GetRValue(LImage2[fontx][n2]) - GetRValue(LImage2[backx][n2]));

}

}

delete[]im;

}

void CTexture::BiLinear\_BMP(COLORREF \*\*DesImage, int DesWidth, int DesHeight, COLORREF \*\*SrcImage, int SrcWidth, int SrcHeight)//双线性插值

{

//比例=源/目标

double WidScale = (double)SrcWidth / (double)DesWidth;

double HeiScale = (double)SrcHeight / (double)DesHeight;

double pm[4];

BYTE red[4], green[4], blue[4];

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

{

double Sy = i \* HeiScale;

int ty = int(Sy);

double v = fabs(Sy - ty);

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

{

double Sx = j \* WidScale;

int tx = int(Sx);

double u = fabs(Sx - tx);

pm[0] = (1 - u)\*(1 - v);

pm[1] = v \* (1 - u);

pm[2] = u \* (1 - v);

pm[3] = u \* v;

if (tx >= SrcWidth - 2)

tx = SrcWidth - 2;

if (ty >= SrcHeight - 2)

ty = SrcHeight - 2;

red[0] = GetRValue(SrcImage[ty][tx]);

red[1] = GetRValue(SrcImage[ty + 1][tx]);

red[2] = GetRValue(SrcImage[ty][tx + 1]);

red[3] = GetRValue(SrcImage[ty + 1][tx + 1]);

green[0] = GetGValue(SrcImage[ty][tx]);

green[1] = GetGValue(SrcImage[ty + 1][tx]);

green[2] = GetGValue(SrcImage[ty][tx + 1]);

green[3] = GetGValue(SrcImage[ty + 1][tx + 1]);

blue[0] = GetBValue(SrcImage[ty][tx]);

blue[1] = GetBValue(SrcImage[ty + 1][tx]);

blue[2] = GetBValue(SrcImage[ty][tx + 1]);

blue[3] = GetBValue(SrcImage[ty + 1][tx + 1]);

double r = 0, g = 0, b = 0;

for (int m = 0; m < 4; m++)

{

r += pm[m] \* red[m];

g += pm[m] \* green[m];

b += pm[m] \* blue[m];

}

DesImage[i][j] = RGB(r, g, b);

}

}

}

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:

int LightNum;//光源数量

BOOL bPlay;//动画开关

CTransform tran;//变换对象

CSphere sphere;

void CTestView::InitialLightingScene(void)

{

LightNum = 1;//光源个数

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

sphere.pLight->LightSource[0].SetPosition(800, 800, 800);//设置第一个光源位置坐标

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

{

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

sphere.pLight->LightSource[i].L\_Specular = CRGB(0.8, 0.8, 0.8);//光源镜面高光颜色

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

sphere.pLight->LightSource[i].L\_C1 = 0.0001;//线性衰减系数

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

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

}

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

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

sphere.pMaterial->SetDiffuse(CRGB(0.6, 0.0, 0.0));//材质对漫反射光的反射率

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

sphere.pMaterial->M\_Emit = CRGB(0.0, 0.0, 0.0);//材质自身发散的颜色

sphere.pMaterial->M\_n = 20.0;//高光指数

}

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

{

CRect rect;//定义客户区矩形

GetClientRect(&rect);//获得客户区的大小

pDC->SetMapMode(MM\_ANISOTROPIC);//pDC自定义坐标系

pDC->SetWindowExt(rect.Width(), rect.Height());//设置窗口范围

pDC->SetViewportExt(rect.Width(), -rect.Height());//设置视区范围,x轴水平向右，y轴垂直向上

pDC->SetViewportOrg(rect.Width() / 2, rect.Height() / 2);//客户区中心为原点

CDC memDC;//内存DC

CBitmap NewBitmap, \*pOldBitmap;//内存中承载的临时位图

memDC.CreateCompatibleDC(pDC);//创建一个与显示pDC兼容的内存memDC

NewBitmap.CreateCompatibleBitmap(pDC, rect.Width(), rect.Height());//创建兼容位图

pOldBitmap = memDC.SelectObject(&NewBitmap);//将兼容位图选入memDC

memDC.SetMapMode(MM\_ANISOTROPIC);//memDC自定义坐标系

memDC.SetWindowExt(rect.Width(), rect.Height());

memDC.SetViewportExt(rect.Width(), -rect.Height());

memDC.SetViewportOrg(rect.Width() / 2, rect.Height() / 2);

rect.OffsetRect(-rect.Width() / 2, -rect.Height() / 2);

DrawObject(&memDC);//向memDC绘制图形

pDC->BitBlt(rect.left, rect.top, rect.Width(), rect.Height(), &memDC, -rect.Width() / 2, -rect.Height() / 2, SRCCOPY);//将内存memDC中的位图拷贝到显示pDC中

memDC.SelectObject(pOldBitmap);//恢复位图

NewBitmap.DeleteObject();//删除位图

}

void CTestView::DrawObject(CDC \*pDC)

{

sphere.Draw(pDC);

}

1. **实现效果**

球面几何纹理映射反走样算法效果如图55-2所示。

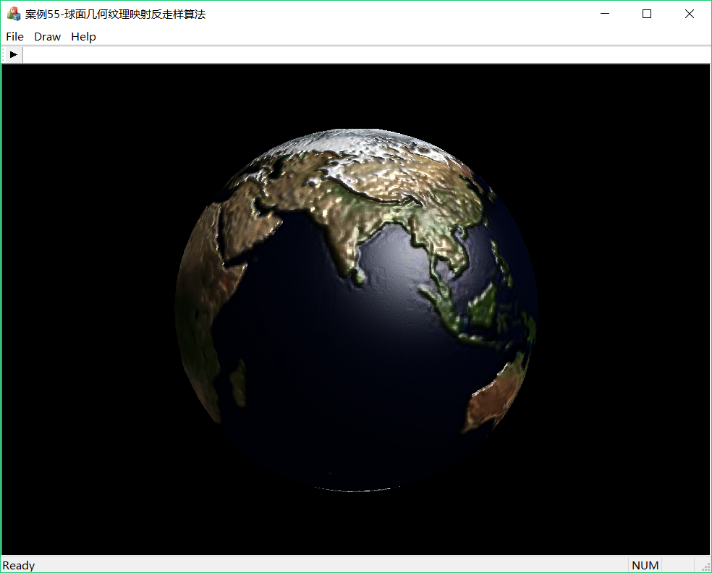


图55-2 球面几何纹理映射反走样算法效果图