**案例52-圆环面图像纹理映射算法**

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**

本案例讲一幅位图映射到圆环面上，并进行光照计算。

1. **知识点**

图像纹理映射需要建立物体表面上每一采样点与已知图像纹理上各点的对应关系，取图像纹理上点的颜色值作为物体表面上采样点的颜色值，然后采用光照模型计算该点处的光强。

1. **实现步骤**
2. 添加基础类与添加绘制圆环的CTorus类。
3. 在CTorus类中计算顶点坐标、读入面表，绘制图形。
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. 添加CTexture类，用于读入纹理。在CTorus类种调用读入纹理函数并调用ZBuffer类中的着色函数计算每一点的纹理值以及光强值。
6. 在CProjection类中进行透视变化。
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. CTorus类

public:

CTorus();

virtual~CTorus();

void ReadVertex();//读入顶点坐标

void CalNormal();//计算法矢量

void ReadFace();//读入面坐标

void Draw(CDC \*pDC);//绘制圆环面

public:

CP3d \*V;//圆环顶点一维数组

CT2 \*T;//圆环纹理一维数组

CVector3 \*N;//圆环顶点法矢量数组

CFace \*\*F;//面的二维数组

int N1, N2;//N1为周向网格,N2纵向网格

int gAlpha, gBeta;//纬度角与经度角

CLighting \*pLight;//光照环境

CMaterial \*pMaterial;//物体材质

CProjection projection;

CTexture texture;

CTorus::CTorus()

{

}

CTorus::~CTorus()

{

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;

}

if (N != NULL)

{

delete[]N;

N = NULL;

}

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

{

delete[] F[n];

F[n] = NULL;

}

delete[]F;

F = NULL;

}

void CTorus::ReadVertex()//点表

{

int tAlpha = 10, tBeta = 10;//等分角度

int r1 = 200, r2 = 80;//圆环半径和环截面半径

N1 = 360 / tAlpha, N2 = 360 / tBeta;//面片数量为N1×N2

V = new CP3d[N1\*N2];//顶点动态数组

T = new CT2[N1\*N2];//纹理动态数组

N = new CVector3[N1\*N2];//法矢量动态数组

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

{

double tAlpha1 = tAlpha \* i\*PI / 180;

for (int j = 0; j < N2; j++)//顶点赋值

{

double tBeta1 = tBeta \* j\*PI / 180;

V[i\*N2 + j].x = (r1 + r2 \* sin(tBeta1))\*sin(tAlpha1);

V[i\*N2 + j].y = r2 \* cos(tBeta1);

V[i\*N2 + j].z = (r1 + r2 \* sin(tBeta1))\*cos(tAlpha1);

//计算顶点的纹理

T[i\*N2 + j].u = (2 \* PI - tAlpha1) / (2 \* PI)\*(texture.bmp.bmWidth - 1);

T[i\*N2 + j].v = tBeta1 / (2 \* PI)\*(texture.bmp.bmHeight - 1);

}

}

}

void CTorus::CalNormal()//计算法矢量

{

for (int i = 0; i < N1; i++)//周向

{

for (int j = 0; j < N2; 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) Beforei = N1 - 1;

if (0 == j) Beforej = N2 - 1;

if (N1 == Afteri) Afteri = 0;

if (N2 == Afterj) Afterj = 0;

CVector3 vN0, vN1, vN2, vN3, AveN;//相邻四个面片的法矢量及平均法矢量

CVector3 vEdge01(V[Beforei\*N2 + Afterj], V[i\*N2 + Afterj]);

CVector3 vEdge02(V[Beforei\*N2 + Afterj], V[i\*N2 + j]);

vN0 = Cross(vEdge01, vEdge02);

CVector3 vEdge11(V[i\*N2 + Afterj], V[Afteri\*N2 + Afterj]);

CVector3 vEdge12(V[i\*N2 + Afterj], V[Afteri\*N2 + j]);

vN1 = Cross(vEdge11, vEdge12);

CVector3 vEdge21(V[i\*N2 + j], V[Afteri\*N2 + j]);

CVector3 vEdge22(V[i\*N2 + j], V[Afteri\*N2 + Beforej]);

vN2 = Cross(vEdge21, vEdge22);

CVector3 vEdge31(V[Beforei\*N2 + j], V[i\*N2 + j]);

CVector3 vEdge32(V[Beforei\*N2 + j], V[i\*N2 + Beforej]);

vN3 = Cross(vEdge31, vEdge32);

AveN = (vN0 + vN1 + vN2 + vN3) / AveN.Mag();//顶点法矢量的平均值

N[i\*N2 + j] = AveN;

}

}

}

void CTorus::ReadFace()//面表

{

F = new CFace \*[N1];

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

{

F[n] = new CFace[N2];

}

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

{

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

{

int tempi = i + 1;

int tempj = j + 1;

if (N2 == tempj) tempj = 0;

if (N1 == tempi) tempi = 0;

F[i][j].SetNum(4);//面的边数

F[i][j].vI[0] = i \* N2 + j;//建立面的边号

F[i][j].vI[1] = i \* N2 + tempj;

F[i][j].vI[2] = tempi \* N2 + tempj;

F[i][j].vI[3] = tempi \* N2 + j;

}

}

}

void CTorus::Draw(CDC \*pDC)//绘制圆环面

{

CalNormal();

CZBuffer \*zbuf = new CZBuffer;//申请内存

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

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)

{

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]];

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

}

if ((N2 - 1) == j)//消除函数纹理的接缝

{

Texture4[1].v = texture.bmp.bmHeight - 1;

Texture4[2].v = texture.bmp.bmHeight - 1;

}

if ((N1 - 1) == i)

{

Texture4[2].u = 1.0;

Texture4[3].u = 1.0;

}

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.Image);//填充四边形

zbuf->ClearMemory();

}

}

}

delete zbuf;

}

2.CTexture类

public:

CTexture();

virtual~CTexture();

void ReadImage();//读入纹理

public:

COLORREF \*\*Image;//二维动态数组

BITMAP bmp;//BITMAP结构

BYTE \*im;

CTexture::CTexture()

{

}

CTexture::~CTexture()

{

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

{

delete[]Image[n];

Image[n] = NULL;

}

delete[]Image;

Image = NULL;

}

void CTexture::ReadImage()//读入纹理

{

CBitmap NewBitmap;

NewBitmap.LoadBitmap(IDB\_BITMAP1);//调入DDB位图

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

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

im = new BYTE[nbytesize];

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

Image = new COLORREF\*[bmp.bmHeight];

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

Image[n1] = new COLORREF[bmp.bmWidth];

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

{

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

{

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

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

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

}

}

delete[]im;

}

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

CTorus torus;

void CTestView::InitialLightingScene(void)

{

LightNum = 1;//光源个数

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

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

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

{

torus.pLight->LightSource[i].L\_Diffuse = CRGB(0.7, 0.7, 0.7); //光源的漫反射颜色

torus.pLight->LightSource[i].L\_Specular = CRGB(0.3, 0.3, 0.3);//光源镜面高光颜色

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

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

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

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

}

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

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

torus.pMaterial->SetDiffuse(CRGB(0.4, 0.4, 0.4));//材质对漫反射光的反射率

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

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

torus.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)

{

torus.Draw(pDC);

}

1. **实现效果**

圆环面图像纹理映射效果如图52-1所示。

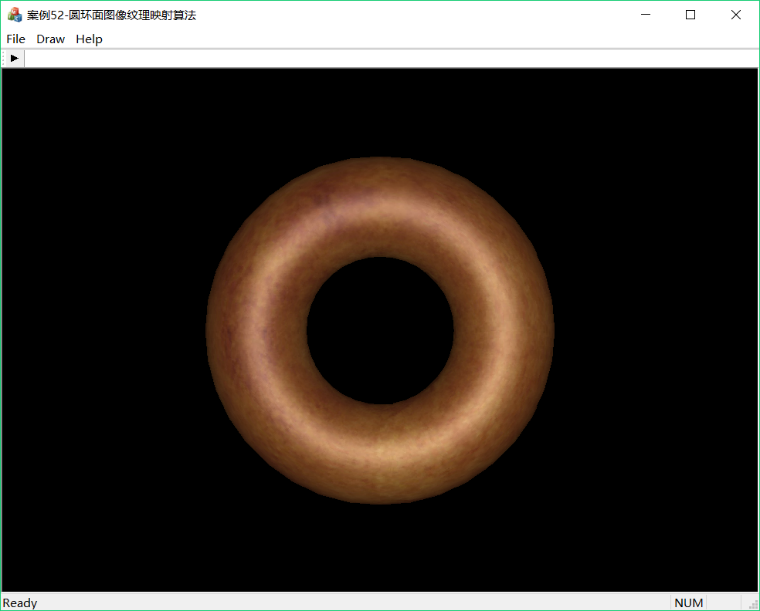


图52-1 圆环面图像纹理映射效果图

1. **知识补充**

这里的圆环面与光照部分的圆环面一样。但是与圆柱面不同的是，圆环面中只有四边形面片，没有三角形面片，这里只需要对四边形绑定纹理坐标即可，但是消除接缝的问题在圆环面中仍然存在，所以也需要进行纹理接缝的消除。