**案例50-长方体图像纹理映射算法**

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. **知识点**

本案例通过图像纹理算法，给长方体添加纹理。在物体表面上映射图像纹理时，首先将相应的位图信息读入到一维数组中。

纹理四边形中的纹理坐标u，v定义在[0,1]区间。真实位图中的纹素坐标u和v定义在[0,w-1]和[0,h-1]区间，如图50-1所示。对于由双三次Bezier构造的曲面，一般将一幅图像映射到一片双三次曲面上。每片曲面的参数是u，v，位图的参数是u和v，用曲面的u、v的双线性插值结果作为地址去查询位图的纹素，就可以简单描述为将位图的顶点坐标绑定到曲面上。用这种方法映射的纹理会随着曲面的转动而转动。

 

1. 纹理四边形 (b) 位图

图50-1 定义纹素坐标

1. **实现步骤**
2. 添加基础类与添加绘制长方体的CCube类。
3. 在CCube类中计算顶点坐标、读入面表，绘制图形。
4. 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CCube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. CCube类

public:

CCube();

virtual~CCube();

void ReadPoint();//读入点表

void ReadFace();//读入面表

void Draw(CDC \*pDC);//绘制立方体表面

public:

CP3d V[8];//点表

CFace F[6];//面表

CProjection projection;

CTexture texture;

CLighting \*pLight;//光照环境

CMaterial \*pMaterial;//物体材质

CCube::CCube()

{

}

CCube::~CCube()

{

if (pLight != NULL)

{

delete pLight;

pLight = NULL;

}

if (pMaterial != NULL)

{

delete pMaterial;

pMaterial = NULL;

}

}

void CCube::ReadPoint()//点表

{

double a = 200, b = 200, c = 50;//长方体边长

//顶点的三维坐标(x,y,z)

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

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

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

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

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

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

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

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

}

void CCube::ReadFace()//面表

{

//面的边数、面的顶点编号

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[0].t[0]=CT2(0,0);F[0].t[1]=CT2(437,0);F[0].t[2]=CT2(437,437);F[0].t[3]=CT2(0,437);//前面纹理坐标

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[1].t[0]=CT2(437,0);F[1].t[1]=CT2(437,437);F[1].t[2]=CT2(0,437);F[1].t[3]=CT2(0,0);//后面纹理坐标

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[2].t[0]=CT2(0,0);F[2].t[1]=CT2(107,0);F[2].t[2]=CT2(107,437);F[2].t[3]=CT2(0,437);//左面纹理坐标

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[3].t[0]=CT2(107,0);F[3].t[1]=CT2(107,437);F[3].t[2]=CT2(0,437);F[3].t[3]=CT2(0,0);//右面纹理坐标

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[4].t[0]=CT2(0,0);F[4].t[1]=CT2(437,0);F[4].t[2]=CT2(437,107);F[4].t[3]=CT2(0,107);//顶面纹理坐标

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

F[5].t[0]=CT2(0,0);F[5].t[1]=CT2(437,0);F[5].t[2]=CT2(437,107);F[5].t[3]=CT2(0,107);//底面纹理坐标

}

void CCube::Draw(CDC\* pDC)//绘制立方体

{

CPi3 Point[4];//面的顶点坐标

CT2 Texture[4];//面的纹理坐标

CVector3 Normal4[4];

CZBuffer \*zbuf = new CZBuffer;

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

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;

Texture[nVertex] = F[nFace].t[nVertex];

}

texture.ReadImage(nFace);

zbuf->SetPoint(Point, Normal4, Texture, 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);//纹理映射

zbuf->ClearMemory();

texture.ClearImaMem();

}

}

delete zbuf;

}

2.CTexture类

public:

CTexture();

virtual~CTexture();

void ReadImage(int);//读入图片纹理

void ClearImaMem();//清除内存

public:

COLORREF \*\*Image;//二维动态数组

BITMAP bmp;//BITMAP结构体变量

BYTE \*im;

void CTexture::ReadImage(int nface)//读入BMP图片

{

BYTE Texture[] = { IDB\_BITMAP1,IDB\_BITMAP6,IDB\_BITMAP5,IDB\_BITMAP4,IDB\_BITMAP3,IDB\_BITMAP2 };

CBitmap NewBitmap;

NewBitmap.LoadBitmap(Texture[nface]);//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;

}

void CTexture::ClearImaMem()

{

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;

}

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)

{

LightNum = 1;//光源个数

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

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

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

{

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

cube.pLight->LightSource[i].L\_Specular = CRGB(0.5, 0.5, 0.5);//光源镜面高光颜色

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

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

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

cube.pLight->LightSource[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(1.0, 1.0, 1.0));//材质对镜面反射光的反射率

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

cube.pMaterial->M\_n = 30.0;//高光指数

}

void CTestView::DoubleBuffer()//双缓冲

{

CDC\* pDC = GetDC();

CRect rect;//定义客户区

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());//x轴水平向右，y轴垂直向上

pDC->SetViewportOrg(rect.Width() / 2, rect.Height() / 2);//屏幕中心为原点

CDC MemDC;//内存DC

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

MemDC.CreateCompatibleDC(pDC);//建立与屏幕pDC兼容的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);

DrawObject(&MemDC);

pDC->BitBlt(-rect.Width() / 2, -rect.Height() / 2, rect.Width(), rect.Height(), &MemDC, -rect.Width() / 2, -rect.Height() / 2, SRCCOPY);//将内存位图拷贝到屏幕

MemDC.SelectObject(pOldBitmap);//恢复位图

NewBitmap.DeleteObject();//删除位图

MemDC.DeleteDC();//删除MemDC

ReleaseDC(pDC);//释放DC

}

void CTestView::DrawObject(CDC \*pDC)

{

cube.Draw(pDC);

}

1. **实现效果**

长方体图像纹理映射效果如图50-2所示。



图50-2 长方体图像纹理映射效果图