**案例38-叠加长方体模型Z-Buffer消隐算法**

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ZBuffer消隐算法对立方体进行消隐。

1. **知识点**

本案例主要用ZBuffer消隐算法对长方体进行消隐，该算法的思路如下：

1. 设置帧缓冲器颜色为背景色
2. 确定深度缓冲器的宽度、高度和初始深度。一般将初始深度置为最大深度值。
3. 对于多边形表面中的每一像素(x,y)，计算其深度值z(x,y)。
4. 将z与存储在深度缓冲器中的(x,y)处的深度值zBuffer(x,y)进行比较。
5. 如果z(x,y)≤zBuffer(x,y),则将次像素的颜色写入帧缓冲器，且用z(x,y)重置zBuffer(x,y)。
6. **实现步骤**
7. 添加基础类与添加绘制长方体的CCuboid类。
8. 在CCuboid类中计算顶点坐标、读入面表、绘制图形，调用CZBuffer类中的着色器对图像进行消隐以及着色。
9. 在CProjection类中对透视变换参数初始化、设置视点位置以及透视变换。
10. 在CTestView中添加消息响应函数，在OnDraw中调用DoubleBuffer函数。
11. **主要算法**

1. CCuboid类

public:

CCuboid(void);

virtual ~CCuboid(void);

void SetParameter(double nLength, double nWidth, double nHeight);

void ReadPoint(void);

void ReadFacet(void);

void Draw(CDC\* pDC, CZBuffer\* pZBuffer);

public:

double nLength, nWidth, nHeight;

CP3d P[8];

CFacet F[6];

CProjection projection;

CCuboid::CCuboid(void)

{

}

CCuboid::~CCuboid(void)

{

}

void CCuboid::SetParameter(double nLength, double nWidth, double nHeight)

{

this->nLength = nLength;

this->nWidth = nWidth;

this->nHeight = nHeight;

}

void CCuboid::ReadPoint(void)

{

P[0].x = -nLength / 2; P[0].y = -nHeight / 2; P[0].z = -nWidth / 2;

P[1].x = nLength / 2; P[1].y = -nHeight / 2; P[1].z = -nWidth / 2;

P[2].x = nLength / 2; P[2].y = nHeight / 2; P[2].z = -nWidth / 2;

P[3].x = -nLength / 2; P[3].y = nHeight / 2; P[3].z = -nWidth / 2;

P[4].x = -nLength / 2; P[4].y = -nHeight / 2; P[4].z = nWidth / 2;

P[5].x = nLength / 2; P[5].y = -nHeight / 2; P[5].z = nWidth / 2;

P[6].x = nLength / 2; P[6].y = nHeight / 2; P[6].z = nWidth / 2;

P[7].x = -nLength / 2; P[7].y = nHeight / 2; P[7].z = nWidth / 2;

}

void CCuboid::ReadFacet(void)

{

F[0].SetVertexNumber(4); F[0].Index[0] = 0; F[0].Index[1] = 3; F[0].Index[2] = 2; F[0].Index[3] = 1; F[0].c = CRGB(1.0, 1.0, 1.0);//后面

F[1].SetVertexNumber(4); F[1].Index[0] = 4; F[1].Index[1] = 5; F[1].Index[2] = 6; F[1].Index[3] = 7; F[1].c = CRGB(1.0, 1.0, 1.0);//前面

F[2].SetVertexNumber(4); F[2].Index[0] = 0; F[2].Index[1] = 4; F[2].Index[2] = 7; F[2].Index[3] = 3; F[2].c = CRGB(1.0, 1.0, 1.0);//左面

F[3].SetVertexNumber(4); F[3].Index[0] = 1; F[3].Index[1] = 2; F[3].Index[2] = 6; F[3].Index[3] = 5; F[3].c = CRGB(1.0, 1.0, 1.0);//右面

F[4].SetVertexNumber(4); F[4].Index[0] = 3; F[4].Index[1] = 7; F[4].Index[2] = 6; F[4].Index[3] = 2; F[4].c = CRGB(0.1, 0.3, 0.4);//顶面

F[5].SetVertexNumber(4); F[5].Index[0] = 0; F[5].Index[1] = 1; F[5].Index[2] = 5; F[5].Index[3] = 4; F[5].c = CRGB(0.1, 0.3, 0.4);//底面

}

void CCuboid::Draw(CDC\* pDC, CZBuffer\* pZBuffer)

{

CP3d ScreenPoint[4];//三维投影点

CP3d ViewPoint = projection.GetEye();//视点

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

{

CVector3 ViewVector(P[F[nFace].Index[0]], ViewPoint);// 面的视向量

ViewVector = ViewVector.Normalize();//视向量规范化

CVector3 Vector01(P[F[nFace].Index[0]], P[F[nFace].Index[1]]);//边向量

CVector3 Vector02(P[F[nFace].Index[0]], P[F[nFace].Index[2]]);//边向量

CVector3 FacetNormal = CrossProduct(Vector01, Vector02);//面法向量

FacetNormal = FacetNormal.Normalize();

for (int nPoint = 0; nPoint < F[nFace].Number; nPoint++)

{

ScreenPoint[nPoint] = projection.PerspectiveProjection3(P[F[nFace].Index[nPoint]]);

ScreenPoint[nPoint].c = F[nFace].c;

}

pZBuffer->SetPoint(ScreenPoint[0], ScreenPoint[1], ScreenPoint[2]);

pZBuffer->GouraudShader(pDC);

pZBuffer->SetPoint(ScreenPoint[0], ScreenPoint[2], ScreenPoint[3]);

pZBuffer->GouraudShader(pDC);

}

}

2.CZBuffer类

public:

CZBuffer(void);

virtual ~CZBuffer(void);

void SetPoint(CP3d P0, CP3d P1, CP3d P2);//设置三角形顶点

void InitialDepthBuffer(int nWidth, int nHeight, double zDepth);//初始化深度缓冲器

void GouraudShader(CDC\* pDC);//光滑着色

private:

void EdgeFlag(CPoint2 PStart, CPoint2 PEnd, BOOL bFeature);//打边标记

CRGB LinearInterp(double t, double tStart, double tEnd, CRGB iStart, CRGB cEnd);//光强线性插值

void SortVertex(void);//顶点排序

private:

CP3d P0, P1, P2;//三角形的浮点数顶点

CPoint3 point0, point1, point2;//三角形的整数顶点坐标

CPoint2\* SpanLeft; //跨度的起点数组标志

CPoint2\* SpanRight;//跨度的终点数组标志

int nIndex;//记录扫描线条数

double\*\* zBuffer;//深度缓冲区

int nWidth, nHeight;//缓冲区宽度和高度

CZBuffer::CZBuffer(void)

{

}

CZBuffer::~CZBuffer(void)

{

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

{

delete[] zBuffer[i];

zBuffer[i] = NULL;

}

if (zBuffer != NULL)

{

delete zBuffer;

zBuffer = NULL;

}

}

void CZBuffer::SetPoint(CP3d P0, CP3d P1, CP3d P2)

{

this->P0 = P0;

this->P1 = P1;

this->P2 = P2;

point0.x = ROUND(P0.x);

point0.y = ROUND(P0.y);

point0.z = P0.z;

point0.c = P0.c;

point1.x = ROUND(P1.x);

point1.y = ROUND(P1.y);

point1.z = P1.z;

point1.c = P1.c;

point2.x = ROUND(P2.x);

point2.y = ROUND(P2.y);

point2.z = P2.z;

point2.c = P2.c;

}

void CZBuffer::GouraudShader(CDC\* pDC)

{

SortVertex();//三角形顶点排序

//定义三角形覆盖的扫描线条数

int nTotalLine = point1.y - point0.y + 1;

//定义span的起点与终点数组

SpanLeft = new CPoint2[nTotalLine];

SpanRight = new CPoint2[nTotalLine];

//判断三角形与P0P1边的位置关系，0-1-2为右手系

int nDeltz = (point1.x - point0.x) \* (point2.y - point1.y) - (point1.y - point0.y) \* (point2.x - point1.x);//法向量的z坐标

if (nDeltz > 0)//三角形位于P0P1边的左侧

{

nIndex = 0;

EdgeFlag(point0, point2, TRUE);

EdgeFlag(point2, point1, TRUE);

nIndex = 0;

EdgeFlag(point0, point1, FALSE);

}

else//三角形位于P0P1边的右侧

{

nIndex = 0;

EdgeFlag(point0, point1, TRUE);

nIndex = 0;

EdgeFlag(point0, point2, FALSE);

EdgeFlag(point2, point1, FALSE);

}

double CurrentDepth = 0.0;//当前扫描线的深度

CVector3 Vector01(P0, P1), Vector02(P0, P2);

CVector3 fNormal = CrossProduct(Vector01, Vector02);

double A = fNormal.x, B = fNormal.y, C = fNormal.z;//平面方程Ax+By+Cz＋D=0的系数

double D = -A \* P0.x - B \* P0.y - C \* P0.z;//系数D

if (fabs(C) < 1e-4)

C = 1.0;

double DepthStep = -A / C;//深度步长

for (int y = point0.y; y < point1.y; y++)//下闭上开

{

int n = y - point0.y;

BOOL bInFlag = FALSE;//跨度内外测试标志，初始值为假表示三角形外部

for (int x = SpanLeft[n].x; x < SpanRight[n].x; x++)//左闭右开

{

if (bInFlag == FALSE)

{

CurrentDepth = -(A \* x + B \* y + D) / C;//z=-(Ax+By+D)/C

bInFlag = TRUE;

x -= 1;

}

else

{

CRGB clr = LinearInterp(x, SpanLeft[n].x, SpanRight[n].x, SpanLeft[n].c, SpanRight[n].c);

if (CurrentDepth <= zBuffer[x + nWidth / 2][y + nHeight / 2])//如果当前采样点的深度小于帧缓冲器中原采样点的深度

{

zBuffer[x + nWidth / 2][y + nHeight / 2] = CurrentDepth;//使用当前采样点的深度更新深度缓冲器

pDC->SetPixelV(x, y, RGB(clr.red \* 255, clr.green \* 255, clr.blue \* 255));

}

CurrentDepth += DepthStep;

}

}

}

if (SpanLeft)

{

delete[]SpanLeft;

SpanLeft = NULL;

}

if (SpanRight)

{

delete[]SpanRight;

SpanRight = NULL;

}

}

void CZBuffer::EdgeFlag(CPoint2 PStart, CPoint2 PEnd, BOOL bFeature)

{

int dx = PEnd.x - PStart.x;

int dy = PEnd.y - PStart.y;

double m = double(dx) / dy;

double x = PStart.x;

for (int y = PStart.y; y < PEnd.y; y++)

{

CRGB crColor = LinearInterp(y, PStart.y, PEnd.y, PStart.c, PEnd.c);

if (bFeature)

SpanLeft[nIndex++] = CPoint2(ROUND(x), y, crColor);

else

SpanRight[nIndex++] = CPoint2(ROUND(x), y, crColor);

x += m;

}

}

void CZBuffer::SortVertex(void)

{

CPoint3 pt[3];

pt[0] = point0;

pt[1] = point1;

pt[2] = point2;

for (int i = 0; i < 2; i++)

{

for (int j = i + 1; j < 3; j++)

{

int k = i;

if (pt[k].y >= pt[j].y)

k = j;

if (k == j)

{

CPoint3 ptTemp = pt[i];

pt[i] = pt[k];

pt[k] = ptTemp;

}

}

}

point0 = pt[0];

point1 = pt[2];

point2 = pt[1];

}

CRGB CZBuffer::LinearInterp(double t, double tStart, double tEnd, CRGB cStart, CRGB cEnd)//颜色线性插值

{

CRGB color;

color = (tEnd - t) / (tEnd - tStart) \* cStart + (t - tStart) / (tEnd - tStart) \* cEnd;

return color;

}

void CZBuffer::InitialDepthBuffer(int nWidth, int nHeight, double zDepth)//初始化深度缓冲

{

this->nWidth = nWidth, this->nHeight = nHeight;

zBuffer = new double \*[nWidth];

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

zBuffer[i] = new double[nHeight];

for (int i = 0; i < nWidth; i++)//初始化深度

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

zBuffer[i][j] = zDepth;

}

3.CTestView类

public:

void DoubleBuffer(CDC\* pDC);//双缓冲绘图

void DrawObject(CDC\* pDC);// 绘制六棱锥表面

protected:

double Alpha, Beta;//x方向旋转α角,y方向旋转β角

BOOL bPlay;//动画开关

CCuboid cuboidTop;//立方体

CCuboid cuboidButton;//长方体

CTransform3 cuboidTop\_transform;

CTransform3 cuboidButton\_transform;

CTestView::CTestView() noexcept

{

// TODO: add construction code here

bPlay = FALSE;

double cuboidTop\_nLength = 250;

double cuboidTop\_nWidth = 350;

double cuboidTop\_nHeight = 200;

cuboidTop.SetParameter(cuboidTop\_nLength, cuboidTop\_nWidth, cuboidTop\_nHeight);

cuboidTop.ReadPoint();

cuboidTop.ReadFacet();

cuboidTop\_transform.SetMatrix(cuboidTop.P, 8);

cuboidTop\_transform.Translate(0, 0, 50);

double cuboidButton\_nLength = 500;

double cuboidButton\_nWidth = 500;

double cuboidButton\_nHeight = 80;

cuboidButton.SetParameter(cuboidButton\_nLength, cuboidButton\_nWidth, cuboidButton\_nHeight);

cuboidButton.ReadPoint();

cuboidButton.ReadFacet();

cuboidButton\_transform.SetMatrix(cuboidButton.P, 8);

cuboidButton\_transform.Translate(0, -140, 0);

cuboidTop\_transform.RotateX(20);

cuboidButton\_transform.RotateX(20);

}

CTestView::~CTestView()

{

}

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

{

CRect rect;

GetClientRect(&rect);

pDC->SetMapMode(MM\_ANISOTROPIC);

pDC->SetWindowExt(rect.Width(), rect.Height());

pDC->SetViewportExt(rect.Width(), -rect.Height());

pDC->SetViewportOrg(rect.Width() / 2, rect.Height() / 2);

CDC memDC;//声明内存DC

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

CBitmap NewBitmap, \*pOldBitmap;

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

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

//memDC.FillSolidRect(rect, pDC->GetBkColor());//设置客户区背景色

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

memDC.SetMapMode(MM\_ANISOTROPIC);//内存DC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);//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DC中的位图拷贝到显示DC

memDC.SelectObject(pOldBitmap);

NewBitmap.DeleteObject();

memDC.DeleteDC();

}

void CTestView::DrawObject(CDC\* pDC)//绘制图形

{

CZBuffer\* pZBuffer = new CZBuffer;

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

cuboidTop.Draw(pDC, pZBuffer);

cuboidButton.Draw(pDC, pZBuffer);

delete pZBuffer;

}

1. **实现效果**

叠加长方体模型Z-Buffer消隐算法效果如图38-1。

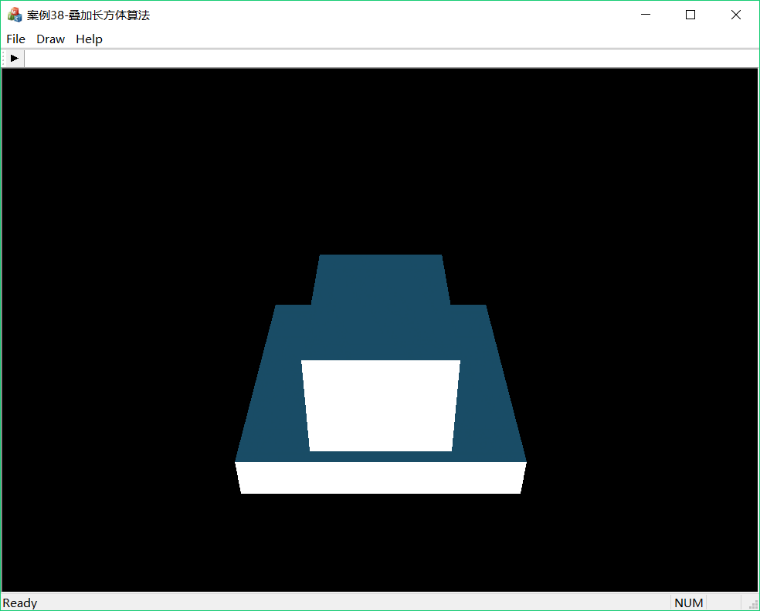


图38-1 叠加长方体模型Z-Buffer消隐算法效果图

1. **实现效果**

Z-Buffer算法的最大优点在于算法简单，与场景复杂度无关，不像之前的背面剔除算法，只适用于凸面体，无法对凹面体消隐。但Z-Buffer算法的缺点是需要占用大量的存储单元。