**案例47-简单透明模型算法**

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**

本案例通过简单透明模型算法制作球体位于透明立方体中心的三维动画。其中球体时非透明物体，立方体为可调节透明度的物体。

1. **知识点**

简单透明模型将物体A上的各像素处的光强与其后的另一个物体B上相应像素处的光强作线性插值以确定物体A上各像素最终显示的光强。

 （47-1）

式(47-1)中为物体A上某一像素的光强，为物体B上相应像素的光强；t为透明度，其值通常曲子RGB模型的alpha分量。当t=1.0时，物体A透明，可以完全看到物体B；当t=0.0时，物体A完全不透明，物体B被物体A遮挡。当t的取值位于区间[0，1]内时，物体的最终颜色时物体A的颜色与物体B的颜色的融合。

1. **实现步骤**
2. 添加基础类与添加绘制几何体的CGeometory类。
3. 在CProjection类中进行透视变化。
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. 在CGeometory类中计算顶点坐标、读入面表，绘制图形。
6. 在绘制图形时调用CZBuffer类中的Phong函数进着色。
7. 在CTestView类中调用CGeometory类的绘制函数。
8. 在CTestView类中添加消息响应函数，在OnDraw中调用DoubleBuffer函数。
9. **主要算法**

1. CGeometory类

public:

CGeometry();

virtual~CGeometry();

void ReadSphereVertex();//读入球面顶点

void ReadSphereFace();//读入球小面

void ReadCubeVertex();//读入立方体顶点

void ReadCubeFace();//读入立方体表面

void DrawSphere(CDC \*pDC, CZBuffer \*zbuf);//绘制球面

void DrawCube(CDC\* pDC, CZBuffer \*zbuf);//绘制立方体

public:

CP3d \*SV;//球的顶点一维数组

CFace \*\*SF;//球的小面的二维数组

CP3d CV[8];//点表

CFace CF[6];//面表

int N1, N2;//N1为纬度区间,N2为经度区间

CLighting \*pLight;//光照环境

CMaterial \*pMaterial;//物体材质

CProjection projection;

CGeometry::CGeometry()

{

}

CGeometry::~CGeometry()

{

if (pLight != NULL)

{

delete pLight;

pLight = NULL;

}

if (pMaterial != NULL)

{

delete[]pMaterial;

pMaterial = NULL;

}

if (SV != NULL)

{

delete[] SV;

SV = NULL;

}

for (int n = 0; n < N1; n++)//注意撤销次序,先列后行,与设置相反

{

delete[] SF[n];

SF[n] = NULL;

}

delete[] SF;

SF = NULL;

}

void CGeometry::ReadSphereVertex()//读入球面顶点坐标

{

int gAlpha = 4, gBeta = 4;//面片夹角

N1 = 180 / gAlpha, N2 = 360 / gBeta;//N1为纬度区域,N2为经度区域

SV = new CP3d[(N1 - 1)\*N2 + 2];//SV为球的顶点

//纬度方向除南北极点外有"N1－1"个点，"2"代表南北极两个点

double gAlpha1, gBeta1, r = 100;//r为球体半径

//计算北极点坐标

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

//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;

SV[i\*N2 + j + 1].x = r \* sin(gAlpha1)\*sin(gBeta1);

SV[i\*N2 + j + 1].y = r \* cos(gAlpha1);

SV[i\*N2 + j + 1].z = r \* sin(gAlpha1)\*cos(gBeta1);

}

}

//计算南极点坐标

SV[(N1 - 1)\*N2 + 1].x = 0, SV[(N1 - 1)\*N2 + 1].y = -r, SV[(N1 - 1)\*N2 + 1].z = 0;

}

void CGeometry::ReadSphereFace()//读入球的面表

{

//设置二维动态数组

SF = new CFace \*[N1];//设置行

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

SF[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;

SF[0][j].SetNum(3);

for (int k = 0; k < SF[0][j].vN; k++)

SF[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;

SF[i][j].SetNum(4);

for (int k = 0; k < SF[i][j].vN; k++)

SF[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;

SF[N1 - 1][j].SetNum(3);

for (int k = 0; k < SF[N1 - 1][j].vN; k++)

SF[N1 - 1][j].vI[k] = SouthIndex[k];

}

}

void CGeometry::ReadCubeVertex()//读入立方体顶点坐标

{

//顶点的三维坐标(x,y,z),立方体边长为2a,t为透明度

double a = 160; double t = 0.7;

CV[0].x = -a; CV[0].y = -a; CV[0].z = -a; CV[0].c.alpha = t;

CV[1].x = +a; CV[1].y = -a; CV[1].z = -a; CV[1].c.alpha = t;

CV[2].x = +a; CV[2].y = +a; CV[2].z = -a; CV[2].c.alpha = t;

CV[3].x = -a; CV[3].y = +a; CV[3].z = -a; CV[3].c.alpha = t;

CV[4].x = -a; CV[4].y = -a; CV[4].z = +a; CV[4].c.alpha = t;

CV[5].x = +a; CV[5].y = -a; CV[5].z = +a; CV[5].c.alpha = t;

CV[6].x = +a; CV[6].y = +a; CV[6].z = +a; CV[6].c.alpha = t;

CV[7].x = -a; CV[7].y = +a; CV[7].z = +a; CV[7].c.alpha = t;

}

void CGeometry::ReadCubeFace()//读入立方体面表

{

//面的顶点数、面的顶点索引号与面的颜色

CF[0].SetNum(4); CF[0].vI[0] = 4; CF[0].vI[1] = 5; CF[0].vI[2] = 6; CF[0].vI[3] = 7;

CF[1].SetNum(4); CF[1].vI[0] = 0; CF[1].vI[1] = 3; CF[1].vI[2] = 2; CF[1].vI[3] = 1;

CF[2].SetNum(4); CF[2].vI[0] = 0; CF[2].vI[1] = 4; CF[2].vI[2] = 7; CF[2].vI[3] = 3;

CF[3].SetNum(4); CF[3].vI[0] = 1; CF[3].vI[1] = 2; CF[3].vI[2] = 6; CF[3].vI[3] = 5;

CF[4].SetNum(4); CF[4].vI[0] = 2; CF[4].vI[1] = 3; CF[4].vI[2] = 7; CF[4].vI[3] = 6;

CF[5].SetNum(4); CF[5].vI[0] = 0; CF[5].vI[1] = 1; CF[5].vI[2] = 5; CF[5].vI[3] = 4;

}

void CGeometry::DrawSphere(CDC \*pDC, CZBuffer \*zbuf)//绘制球面

{

CPi3 Point3[3];//南北极顶点数组

CVector3 Normal3[3];

CPi3 Point4[4];//球体顶点数组

CVector3 Normal4[4];

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

{

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

{

CVector3 ViewVector(SV[SF[i][j].vI[0]], projection.ViewPoint);//面的视矢量

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

SF[i][j].SetFaceNormal(SV[SF[i][j].vI[0]], SV[SF[i][j].vI[1]], SV[SF[i][j].vI[2]]);

SF[i][j].fNormal.Normalize();//单位化法矢量

if (Dot(ViewVector, SF[i][j].fNormal) >= 0)//背面剔除

{

if (3 == SF[i][j].vN)

{

for (int m = 0; m < SF[i][j].vN; m++)

{

projection.PerProject(SV[SF[i][j].vI[m]]);

Point3[m] = projection.ScreenP;

Normal3[m] = CVector3(SV[SF[i][j].vI[m]]);

}

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

zbuf->CreateBucket();//创建桶表

zbuf->CreateEdge();//创建边表

zbuf->Phong(pDC, projection.ViewPoint, pLight, pMaterial, SPHERE);//颜色渐变填充三角形

zbuf->ClearMemory();

}

else

{

for (int m = 0; m < SF[i][j].vN; m++)

{

projection.PerProject(SV[SF[i][j].vI[m]]);

Point4[m] = projection.ScreenP;

Normal4[m] = CVector3(SV[SF[i][j].vI[m]]);

}

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

zbuf->CreateBucket();//创建桶表

zbuf->CreateEdge();//创建边表

zbuf->Phong(pDC, projection.ViewPoint, pLight, &pMaterial[0], SPHERE);//颜色渐变填充四边形

zbuf->ClearMemory();

}

}

}

}

}

void CGeometry::DrawCube(CDC\* pDC, CZBuffer \*zbuf)//绘制立方体表面

{

CPi3 Point[4];//面的二维顶点数组

CVector3 Normal4[4];//面的法矢量

for (int nFace = 0; nFace < 6; nFace++)//面循环

{

for (int nVertex = 0; nVertex < CF[nFace].vN; nVertex++)//顶点循环

{

projection.PerProject(CV[CF[nFace].vI[nVertex]]);//透视投影

Point[nVertex] = projection.ScreenP;

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

}

zbuf->SetPoint(Point, Normal4, 4);//设置顶点

zbuf->CreateBucket();//创建桶表

zbuf->CreateEdge();//创建边表

zbuf->Phong(pDC, projection.ViewPoint, pLight, &pMaterial[1], CUBE);//颜色渐变填充四边形

zbuf->ClearMemory();

}

}

2.CZBuffer类

public:

CZBuffer();

virtual ~CZBuffer();

void CreateBucket();//创建桶

void CreateEdge();//边表

void AddEt(CAET \*pNewEdge);//合并ET表

void ETOrder();//ET表排序

void Phong(CDC \*pDC, CP3d ViewPoint, CLighting \*pLight, CMaterial \*pMaterial, int Sign);//Phong填充函数

CRGB GetTransColor(CRGB clr1, CRGB clr2);//计算透明颜色

void InitDeepBuffer(int Width, int Height, double Depth, CRGB BkClr);//初始化深度缓存

CRGB Interpolation(double t, double t1, double t2, CRGB clr1, CRGB clr2);//线性插值

CVector3 Interpolation(double t, double t1, double t2, CVector3 vec1, CVector3 vec2);//法矢线性插值

void SetPoint(CPi3 \*p, CVector3 \*n, int m);

void ClearMemory();//清理内存

void DeleteAETChain(CAET\* pAET);//删除边表

protected:

int PNum;//顶点个数

CPi3 \*P;//顶点数组

CVector3 \*N;//顶点的法矢量动态数组

CAET \*pHeadE, \*pCurrentE, \*pEdge;//有效边表结点指针

CBucket \*pCurrentB, \*pHeadB;

double \*\*zBuffer;//深度缓冲区

CRGB \*\*cBuffer;//颜色缓冲区

int Width, Height;//缓冲区参数

CZBuffer::CZBuffer()

{

P = NULL;

pHeadE = NULL;

pCurrentB = NULL;

pEdge = NULL;

pCurrentE = NULL;

pHeadB = NULL;

zBuffer = NULL;

cBuffer = NULL;

}

CZBuffer::~CZBuffer()

{

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

{

delete[] zBuffer[i];

delete[] cBuffer[i];

zBuffer[i] = NULL;

cBuffer[i] = NULL;

}

if (zBuffer != NULL)

{

delete zBuffer;

zBuffer = NULL;

}

if (cBuffer != NULL)

{

delete cBuffer;

cBuffer = NULL;

}

ClearMemory();

}

void CZBuffer::SetPoint(CPi3 \*p, CVector3 \*n, int m)

{

P = new CPi3[m];

N = new CVector3[m];

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

{

P[i] = p[i];

N[i] = n[i];

}

PNum = m;

}

void CZBuffer::CreateBucket()//创建桶表

{

int yMin, yMax;

yMin = yMax = P[0].y;

for (int i = 1; i < PNum; i++)//查找多边形所覆盖的最小和最大扫描线

{

if (P[i].y < yMin)

yMin = P[i].y;//扫描线的最小值

if (P[i].y > yMax)

yMax = P[i].y;//扫描线的最大值

}

for (int y = yMin; y <= yMax; y++)

{

if (yMin == y)//建立桶头结点

{

pHeadB = new CBucket;//建立桶的头结点

pCurrentB = pHeadB;//pCurrentB为CBucket当前结点指针

pCurrentB->ScanLine = yMin;

pCurrentB->pET = NULL;//没有链接边表

pCurrentB->pNext = NULL;

}

else//其他扫描线

{

pCurrentB->pNext = new CBucket;//建立桶的其他结点

pCurrentB = pCurrentB->pNext;

pCurrentB->ScanLine = y;

pCurrentB->pET = NULL;

pCurrentB->pNext = NULL;

}

}

}

void CZBuffer::CreateEdge()//创建边表

{

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

{

pCurrentB = pHeadB;

int j = (i + 1) % PNum;//边的第二个顶点，P[i]和P[j]构成边

if (P[i].y < P[j].y)//边的终点比起点高

{

pEdge = new CAET;

pEdge->x = P[i].x;//计算ET表的值

pEdge->yMax = P[j].y;

pEdge->k = (P[j].x - P[i].x) / (P[j].y - P[i].y);//代表1/k

pEdge->ps = P[i];//绑定顶点和颜色

pEdge->pe = P[j];

pEdge->ns = N[i];

pEdge->ne = N[j];

pEdge->pNext = NULL;

while (pCurrentB->ScanLine != P[i].y)//在桶内寻找该边的yMin

{

pCurrentB = pCurrentB->pNext;//移到yMin所在的桶结点

}

}

if (P[j].y < P[i].y)//边的终点比起点低

{

pEdge = new CAET;

pEdge->x = P[j].x;

pEdge->yMax = P[i].y;

pEdge->k = (P[i].x - P[j].x) / (P[i].y - P[j].y);

pEdge->ps = P[i];

pEdge->pe = P[j];

pEdge->ns = N[i];

pEdge->ne = N[j];

pEdge->pNext = NULL;

while (pCurrentB->ScanLine != P[j].y)

{

pCurrentB = pCurrentB->pNext;

}

}

if (int(P[j].y) != P[i].y)

{

pCurrentE = pCurrentB->pET;

if (pCurrentE == NULL)

{

pCurrentE = pEdge;

pCurrentB->pET = pCurrentE;

}

else

{

while (pCurrentE->pNext != NULL)

{

pCurrentE = pCurrentE->pNext;

}

pCurrentE->pNext = pEdge;

}

}

}

}

void CZBuffer::Phong(CDC \*pDC, CP3d ViewPoint, CLighting \*pLight, CMaterial \*pMaterial, int Sign)//填充多边形

{

double z = 0.0;//当前扫描线的z

double zStep = 0.0;//当前扫描线随着x增长的z步长

double A, B, C, D;//平面方程Ax+By+Cz＋D=0的系数

CVector3 V01(P[0], P[1]), V02(P[0], P[2]);

CVector3 VN = Cross(V01, V02);

A = VN.x; B = VN.y; C = VN.z;

D = -A \* P[0].x - B \* P[0].y - C \* P[0].z;

zStep = -A / C;//计算直线z增量

CAET \*pT1, \*pT2;

pT1 = NULL;

pT2 = NULL;

pHeadE = NULL;

for (pCurrentB = pHeadB; pCurrentB != NULL; pCurrentB = pCurrentB->pNext)

{

for (pCurrentE = pCurrentB->pET; pCurrentE != NULL; pCurrentE = pCurrentE->pNext)

{

pEdge = new CAET;

pEdge->x = pCurrentE->x;

pEdge->yMax = pCurrentE->yMax;

pEdge->k = pCurrentE->k;

pEdge->ps = pCurrentE->ps;

pEdge->pe = pCurrentE->pe;

pEdge->ns = pCurrentE->ns;

pEdge->ne = pCurrentE->ne;

pEdge->pNext = NULL;

AddEt(pEdge);

}

ETOrder();

pT1 = pHeadE;

if (pT1 == NULL)

return;

while (pCurrentB->ScanLine >= pT1->yMax)//下闭上开

{

CAET \* pAETTEmp = pT1;

pT1 = pT1->pNext;

delete pAETTEmp;

pHeadE = pT1;

if (pHeadE == NULL)

return;

}

if (pT1->pNext != NULL)

{

pT2 = pT1;

pT1 = pT2->pNext;

}

while (pT1 != NULL)

{

if (pCurrentB->ScanLine >= pT1->yMax)//下闭上开

{

CAET\* pAETTemp = pT1;

pT2->pNext = pT1->pNext;

pT1 = pT2->pNext;

delete pAETTemp;

}

else

{

pT2 = pT1;

pT1 = pT2->pNext;

}

}

CVector3 na, nb, nf;//na、nb代表边上任意点的法矢量，nf代表面上任意点的法矢量

na = Interpolation(pCurrentB->ScanLine, pHeadE->ps.y, pHeadE->pe.y, pHeadE->ns, pHeadE->ne);

nb = Interpolation(pCurrentB->ScanLine, pHeadE->pNext->ps.y, pHeadE->pNext->pe.y, pHeadE->pNext->ns, pHeadE->pNext->ne);

BOOL bInFlag = FALSE;//区间内外测试标志，初始值为假表示区间外部

double xb, xe;//扫描线和有效边相交区间的起点和终点坐标

for (pT1 = pHeadE; pT1 != NULL; pT1 = pT1->pNext)

{

if (FALSE == bInFlag)

{

xb = pT1->x;

z = -(xb\*A + pCurrentB->ScanLine\*B + D) / C;//z=-(Ax+By+D)/C

bInFlag = TRUE;

}

else

{

xe = pT1->x;

for (double x = xb; x < xe; x++)//左闭右开

{

nf = Interpolation(x, xb, xe, na, nb);

CRGB c = pLight->Illuminate(ViewPoint, CP3d(Round(x), pCurrentB->ScanLine, z), nf, pMaterial);

if (z <= zBuffer[Round(x) + Width / 2][pCurrentB->ScanLine + Height / 2])//如果新采样点的深度大于原采样点的深度

{

if (SPHERE == Sign)//球不透明，写入帧缓冲

{

cBuffer[Round(x) + Width / 2][pCurrentB->ScanLine + Height / 2] = c;

zBuffer[Round(x) + Width / 2][pCurrentB->ScanLine + Height / 2] = z;

pDC->SetPixelV(Round(x), pCurrentB->ScanLine, RGB(c.red \* 255, c.green \* 255, c.blue \* 255));

}

else

{

CRGB ctrsns = GetTransColor(c, cBuffer[Round(x) + Width / 2][pCurrentB->ScanLine + Height / 2]);

zBuffer[Round(x) + Width / 2][pCurrentB->ScanLine + Height / 2] = z;

pDC->SetPixelV(Round(x), pCurrentB->ScanLine, RGB(ctrsns.red \* 255, ctrsns.green \* 255, ctrsns.blue \* 255));

}

}

z += zStep;

}

bInFlag = FALSE;

}

}

for (pT1 = pHeadE; pT1 != NULL; pT1 = pT1->pNext)//边的连续性

pT1->x = pT1->x + pT1->k;

}

}

void CZBuffer::AddEt(CAET \*pNewEdge)//合并ET表

{

CAET \*pCE;

pCE = pHeadE;

if (pCE == NULL)

{

pHeadE = pNewEdge;

pCE = pHeadE;

}

else

{

while (pCE->pNext != NULL)

{

pCE = pCE->pNext;

}

pCE->pNext = pNewEdge;

}

}

void CZBuffer::ETOrder()//边表的冒泡排序算法

{

CAET \*pT1, \*pT2;

int Count = 1;

pT1 = pHeadE;

if (pT1 == NULL)

return;

if (pT1->pNext == NULL)//如果该ET表没有再连ET表

return;//桶结点只有一条边，不需要排序

while (pT1->pNext != NULL)//统计边结点的个数

{

Count++;

pT1 = pT1->pNext;

}

for (int i = 0; i < Count - 1; i++)//冒泡排序

{

CAET \*\*pPre = &pHeadE;

pT1 = pHeadE;

for (int j = 0; j < Count - 1 - i; j++)

{

pT2 = pT1->pNext;

if ((pT1->x > pT2->x) || ((pT1->x == pT2->x) && (pT1->k > pT2->k)))

{

pT1->pNext = pT2->pNext;

pT2->pNext = pT1;

\*pPre = pT2;

pPre = &(pT2->pNext);//调整位置为下次遍历准备

}

else

{

pPre = &(pT1->pNext);

pT1 = pT1->pNext;

}

}

}

}

CRGB CZBuffer::Interpolation(double t, double t1, double t2, CRGB clr1, CRGB clr2)//颜色线性插值

{

CRGB color;

color = (t - t2) / (t1 - t2)\*clr1 + (t - t1) / (t2 - t1)\*clr2;

return color;

}

CVector3 CZBuffer::Interpolation(double t, double t1, double t2, CVector3 vec1, CVector3 vec2)//矢量线性插值

{

CVector3 vector;

vector = (t - t2) / (t1 - t2)\*vec1 + (t - t1) / (t2 - t1)\*vec2;

return vector;

}

CRGB CZBuffer::GetTransColor(CRGB clr1, CRGB clr2)//计算透明颜色

{

CRGB color;

double t = P[0].c.alpha;//透明度

color = (1 - t)\*clr1 + t \* clr2;

return color;

}

void CZBuffer::InitDeepBuffer(int Width, int Height, double Depth, CRGB BkClr)//初始化深度与颜色缓存

{

this->Width = Width, this->Height = Height;

zBuffer = new double \*[Width];//深度缓存器

cBuffer = new CRGB \*[Width];//颜色缓存器

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

{

zBuffer[i] = new double[Height];

cBuffer[i] = new CRGB[Height];

}

for (int i = 0; i < Width; i++)//初始化深度缓存

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

{

zBuffer[i][j] = Depth;

cBuffer[i][j] = BkClr;

}

}

void CZBuffer::ClearMemory()

{

DeleteAETChain(pHeadE);

CBucket \*pBucket = pHeadB;

while (pBucket != NULL)//针对每一个桶

{

CBucket \* pBucketTemp = pBucket->pNext;

DeleteAETChain(pBucket->pET);

delete pBucket;

pBucket = pBucketTemp;

}

pHeadB = NULL;

pHeadE = NULL;

if (P != NULL)

{

delete[]P;

P = NULL;

}

if (N != NULL)

{

delete[]N;

N = NULL;

}

}

void CZBuffer::DeleteAETChain(CAET\* pAET)

{

while (pAET != NULL)

{

CAET\* pAETTemp = pAET->pNext;

delete pAET;

pAET = pAETTemp;

}

}

3.CTestView类

public:

void DoubleBuffer(CDC\* pDC);//双缓冲

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

CGeometry geometry;

void CTestView::InitialLightingScene(void)

{

LightNum = 1;//光源个数

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

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

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

{

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

geometry.pLight->Light[i].L\_Specular = CRGB(1.0, 1.0, 1.0);//光源镜面高光颜色

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

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

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

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

}

//球的材质

geometry.pMaterial = new CMaterial[2];//一维材质动态数组

geometry.pMaterial[0].SetAmbient(CRGB(0.175, 0.012, 0.012));//材质对环境光光的反射率

geometry.pMaterial[0].SetDiffuse(CRGB(0.614, 0.041, 0.041));//材质对漫反射光的反射率

geometry.pMaterial[0].SetSpecular(CRGB(1.0, 1.0, 1.0));//材质对镜面反射光的反射率

geometry.pMaterial[0].SetEmit(CRGB(0.2, 0.0, 0.0));//材质自身发散的颜色

geometry.pMaterial[0].M\_n = 30.0;//高光指数

//立方体的材质

geometry.pMaterial[1].SetAmbient(CRGB(0.0, 0.8, 0.0));//材质对环境光光的反射率

geometry.pMaterial[1].SetDiffuse(CRGB(0.0, 0.8, 0.0));//材质对环境光和漫反射光的反射率相等

geometry.pMaterial[1].SetSpecular(CRGB(1.0, 1.0, 1.0));//材质对镜面反射光的反射率

geometry.pMaterial[1].SetEmit(CRGB(0.0, 0.2, 0.0));//材质自身发散的颜色

geometry.pMaterial[1].M\_n = 30.0;//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

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

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

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)//绘制物体

{

CZBuffer \*zbuf = new CZBuffer;//申请深度缓冲内存

zbuf->InitDeepBuffer(800, 800, 1000, CRGB(0.0, 0.0, 0.0));//初始化深度和背景色

geometry.DrawSphere(pDC, zbuf);

geometry.DrawCube(pDC, zbuf);

delete zbuf;//释放内存

}

1. **实现效果**

简单透明模型算法效果图如图47-1所示。

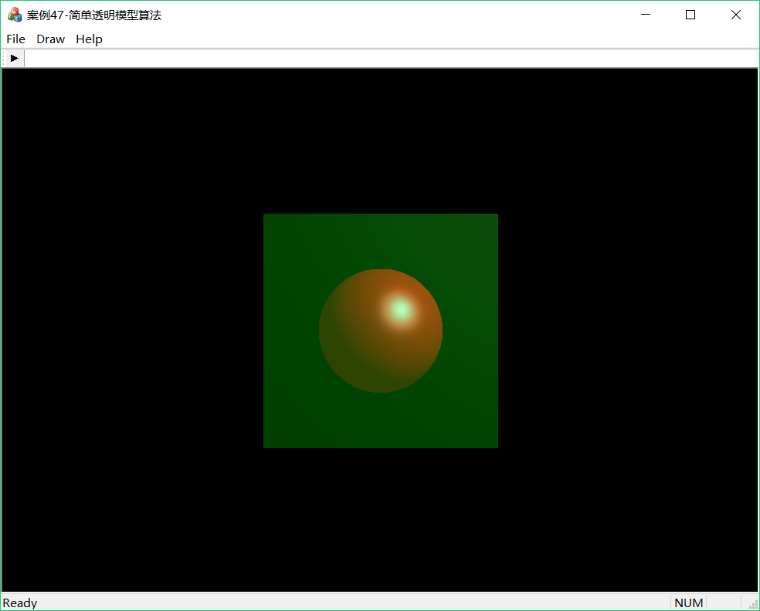


图47-1 简单透明模型算法效果图