**案例29-球面地理划分算法**

1. **主要知识点**

球面的绘制不同于普通几何体体面的绘制，球面的绘制包含了两极处三角形面片的绘制以及中间部分四边形面片的绘制。由于本案例绘制球面的地理划分，所以找到点表是绘制的关键。

本案例中假设面片单位夹角分别为gAlpha=4°，gBeta=4°，因此纬度区域则有个，经度区域则有个，纬度方向除去南北极有个点，所以球面上共有个点；球面上点的坐标按行计算，在纬度增加一个单位夹角时，在该纬度上经度依次增加一个单位夹角，直到循环到第个停止，如此循环即可求得球面地理划分的点表。

1. **案例描述：**

本案例利用球面地理划分算法，绘制地理划分球面。

1. **实现步骤：**
2. 添加基础类与添加绘制立方体的CSphereDiv类。
3. 在CSphereDiv类中计算顶点坐标、读入面表，绘制图形，透视变换参数初始化、设置视点位置以及透视变换。
4. 在CTestView中添加消息响应函数，在OnDraw中调用DoubleBuffer函数。
5. **主要算法**

1. CSphereDiv类：

public:

void ReadPoint();//读入点表

void ReadFace();//读入面表

void InitParameter();//参数初始化

void SetViewPoint();//设置视点

CP2i PersPectiveProjection(CP3d P);//透视投影

void Draw(CDC\* pDC);//绘制球面线框

public:

int nClientWidth;//屏幕客户区宽度

int nClientHeight;//屏幕客户区高度

int nHWidth, nHHeight;//屏幕客户区的半宽和半高

double R, Theta, Phi, d;//视点在用户坐标系中的球坐标

double k[9];

CP3d ViewPoint;//视点球坐标位置

CP2i ScreenCoorP;//屏幕坐标系的二维坐标点

CP3d \*V;//球面顶点一维数组

CFace \*\*F;//小面的二维数组

int N1, N2;//N1为纬度区域,N2为经度区域

double Alpha, Beta;//x方向旋转α角,y方向旋转β角

void CSphereDiv::ReadPoint()//点表

{

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

V = new CP3d[(N1 - 1)\*N2 + 2]; //V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 = 300;//r为球体半径

//计算北极点坐标

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

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);

}

}

//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;

}

void CSphereDiv::ReadFace()//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 CSphereDiv::InitParameter()//透视变换参数初始化

{

k[1] = sin(PI \* Theta / 180);

k[2] = sin(PI \* Phi / 180);

k[3] = cos(PI \* Theta / 180);

k[4] = cos(PI \* Phi / 180);

k[5] = k[2] \* k[3];

k[6] = k[2] \* k[1];

k[7] = k[4] \* k[3];

k[8] = k[4] \* k[1];

}

CP2i CSphereDiv::PersPectiveProjection(CP3d P)//透视投影

{

CP3d ViewCoorP;//观察坐标系内的点

ViewCoorP.x = k[3] \* P.x - k[1] \* P.z;//观察坐标系三维坐标

ViewCoorP.y = -k[8] \* P.x + k[2] \* P.y - k[7] \* P.z;

ViewCoorP.z = -k[6] \* P.x - k[4] \* P.y - k[5] \* P.z + R;

ScreenCoorP.x = int(d \* ViewCoorP.x / ViewCoorP.z);//屏幕坐标系二维坐标

ScreenCoorP.y = int(d \* ViewCoorP.y / ViewCoorP.z);

return ScreenCoorP;

}

void CSphereDiv::SetViewPoint()//设置视点

{

ViewPoint.x = R \* k[6];

ViewPoint.y = R \* k[4];

ViewPoint.z = R \* k[5];

}

void CSphereDiv::Draw(CDC\* pDC)//绘制球面线框模型

{

CLine \*line = new CLine;

CP2i Point3[3], t3;//南北极顶点数组

CP2i Point4[4], t4;//球体顶点数组

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

{

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

{

CVector3 ViewVector(V[F[i][j].vI[0]], 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]]);

F[i][j].fNormal.Normalize();//单位化法矢量

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

{

if (3 == F[i][j].vN)//三角形面片

{

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

{

PersPectiveProjection(V[F[i][j].vI[m]]);

Point3[m] = ScreenCoorP;

}

for (int n = 0; n < 3; n++)

{

if (0 == n)

{

line->MoveTo(pDC, nHWidth + Point3[n].x, nHHeight - Point3[n].y);

t3 = Point3[n];

}

else

line->LineTo(pDC, nHWidth + Point3[n].x, nHHeight - Point3[n].y);

}

line->LineTo(pDC, nHWidth + t3.x, nHHeight - t3.y);//闭合多边形

}

else//四边形面片

{

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

{

PersPectiveProjection(V[F[i][j].vI[m]]);

Point4[m] = ScreenCoorP;

}

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

{

if (0 == n)

{

line->MoveTo(pDC, nHWidth + Point4[n].x, nHHeight - Point4[n].y);

t4 = Point4[n];

}

else

line->LineTo(pDC, nHWidth + Point4[n].x, nHHeight - Point4[n].y);

}

line->LineTo(pDC, nHWidth + t4.x, nHHeight - t4.y);//闭合多边形

}

}

}

}

delete line;

}

2.CTestView类：

public:

void DoubleBuffer(CDC\* pDC);//双缓存绘图

void DrawObject(CDC\* pDC);//绘图

protected:

BOOL bPlay;//动画开关

CTransform3 tran;//变换对象

CSphereDiv spherediv;

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

{

CRect rect;//定义客户区

GetClientRect(&rect);//获得客户区的大小

spherediv.nClientWidth = rect.Width();//屏幕客户区宽度

spherediv.nClientHeight = rect.Height();//屏幕客户区高度

spherediv.nHWidth = spherediv.nClientWidth / 2;//屏幕客户区半宽

spherediv.nHHeight = spherediv.nClientHeight / 2;//屏幕客户区半高

CDC memDC;

memDC.CreateCompatibleDC(pDC);

CBitmap NewBitmap, \*pOldBitmap;

NewBitmap.CreateCompatibleBitmap(pDC, spherediv.nClientWidth, spherediv.nClientHeight);

pOldBitmap = memDC.SelectObject(&NewBitmap);

memDC.FillSolidRect(rect, pDC->GetBkColor());

DrawObject(&memDC);

pDC->BitBlt(0, 0, spherediv.nClientWidth, spherediv.nClientHeight, &memDC, 0, 0, SRCCOPY);

memDC.SelectObject(pOldBitmap);

NewBitmap.DeleteObject();

}

void CTestView::DrawObject(CDC \* pDC)

{

spherediv.Draw(pDC);

}

1. **实现效果：**

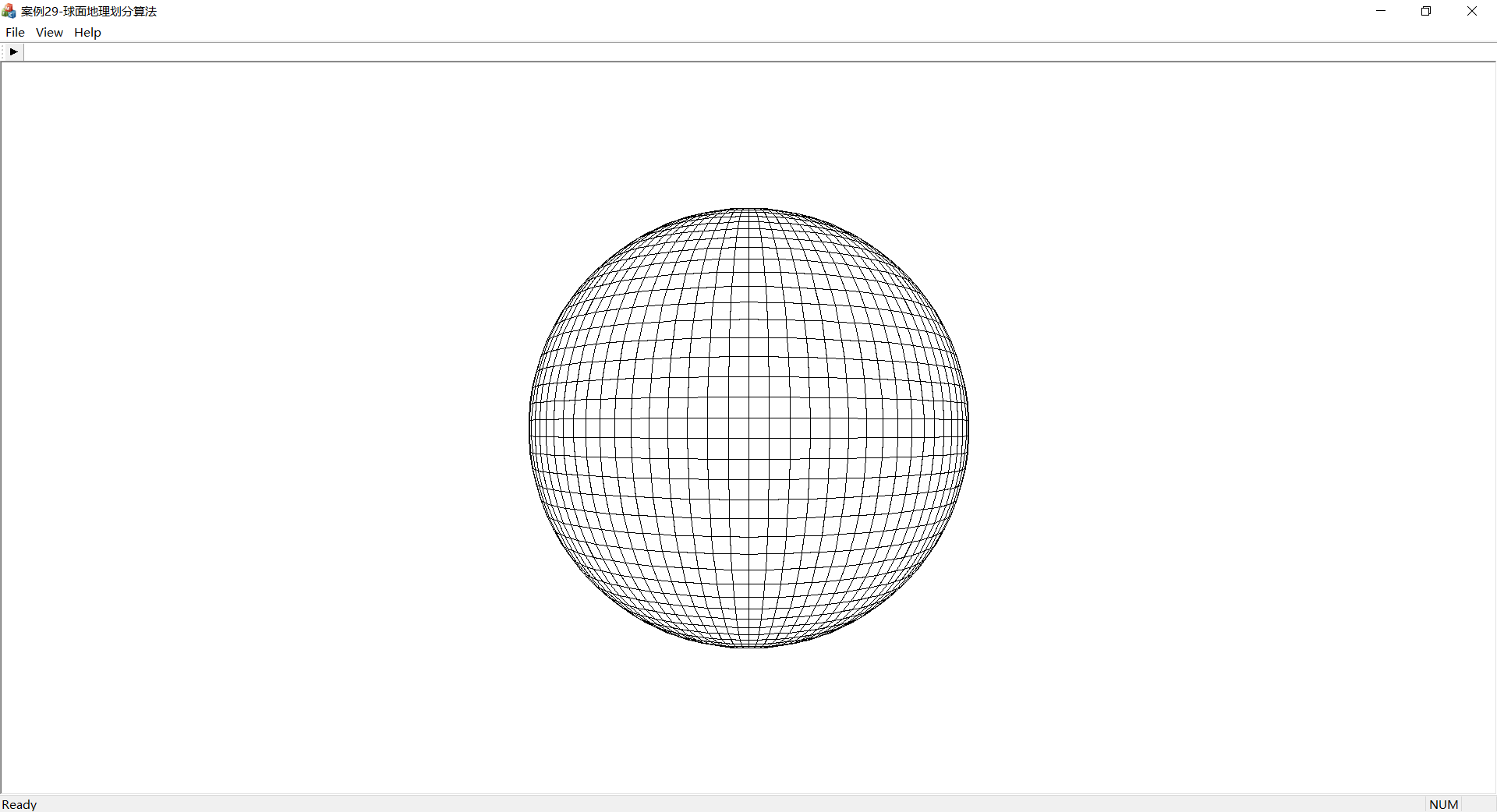


图1球面地理划分算法效果图