```cpp
#pragma once
#include <stdio.h>
#include "grid3D.h"
#include "FuncionesOpenGL.h"
#include "Class_Vector.h"



double xx0,yy0,zz0,xx1,yy1,zz1,xx2,yy2,zz2;
int xxi;
int Dibuja3puntos=false;
int QuienGeneraPoligonos;

#define sqr(x) ((x)*(x))


extern void ZGlobal(double*v);
extern int NumON;
extern float NumEscala;
extern int Modo_DibujaEdges,ModoDibujaNormales;
extern int ModoDibujaInterior;
extern int ModoDibujaFrontera;
extern int Modo_DibujaCentroCaras;
extern int Modo_DibujaCentroBloques;
extern double Dominio_Rint,Dominio_Rmax,Dominio_Hsup;



double ppunto(R3 a,R3 b) {
    return(a.x*b.x+a.y*b.y+a.z*b.z);
}

double ppuntodiff(R3 N,R3 b,R3 c) {
    return(N.x*(b.x-c.x)+N.y*(b.y-c.y)+N.z*(b.z-c.z));
}

grid3D::grid3D(void)
{
}

grid3D::~grid3D(void)
{
}


void grid3D::draw_caraGL(int ii[4])
{

    int li;
    double x[4],y[4],z[4],v[3],xg,yg,zg,lambda=0.0,nx,ny,nz,nxx,nyy,nzz,nnn;
    GLdouble winX1,winY1,winX2,winY2,winX3,winY3,winZ,winZ2;
    xg=0;yg=0;zg=0;

    for (li=0;li<4;li++) {
        x[li]=v3D[ii[li]].x;
        y[li]=v3D[ii[li]].y;
        z[li]=v3D[ii[li]].z;
        xg+=x[li]/4;        yg+=y[li]/4;        zg+=z[li]/4;
    }
    for (li=0;li<4;li++) {
        x[li]+=(xg-x[li])*lambda;
        y[li]+=(yg-y[li])*lambda;
        z[li]+=(zg-z[li])*lambda;
    }

    nxx=nx=(y[2]-y[0])*(z[3]-z[1])-(z[2]-z[0])*(y[3]-y[1]);
    nyy=ny=(z[2]-z[0])*(x[3]-x[1])-(x[2]-x[0])*(z[3]-z[1]);
    nzz=nz=(x[2]-x[0])*(y[3]-y[1])-(y[2]-y[0])*(x[3]-x[1]);
    nnn=sqrt(sqr(nxx)+sqr(nyy)+sqr(nzz));

    //  ZGlobal(v);
    FuncionesOpenGL::modelview_calculado=false;
    /*
    FuncionesOpenGL::World2Win(xg,yg,zg,&winX,&winY,&winZ);
    FuncionesOpenGL::World2Win(xg+nxx,yg+nyy,zg+nzz,&winX,&winY,&winZ2);
    if (winZ2>winZ){
    nxx=-nxx;nyy=-nyy;nzz=-nzz;
    }
    */
    FuncionesOpenGL::World2Win(x[0],y[0],z[0],&winX1,&winY1,&winZ);
    FuncionesOpenGL::World2Win(x[1],y[1],z[1],&winX2,&winY2,&winZ);
    FuncionesOpenGL::World2Win(x[2],y[2],z[2],&winX3,&winY3,&winZ);
    if ((winX2-winX1)*(winY3-winY1)-(winY2-winY1)*(winX3-winX1) <0 ){
        nxx=-nxx;nyy=-nyy;nzz=-nzz;
    }

    /*      if (nxx*v[0]+nyy*v[1]+nzz*v[2]<0){
    nxx=-nxx;nyy=-nyy;nzz=-nzz;
    }
    */

    glEnable(GL_NORMALIZE);
    glBegin(GL_QUADS);
    for (li=0;li<4;li++) {
        if (x[li]<xg)
            nx=nxx+(x[li]-xg)*((x[li]-xg))/0.51;
        else
```

```
            nx=nxx-(x[li]-xg)*((x[li]-xg))/0.51;
        if (y[li]<yg)
            ny=nyy+(y[li]-yg)*((y[li]-yg))/0.51;
        else
            ny=nyy-(y[li]-yg)*((y[li]-yg))/0.51;
        if  (z[li]<zg)
            nz=nzz+(z[li]-zg)*((z[li]-zg))/0.51;
        else
            nz=nzz-(z[li]-zg)*((z[li]-zg))/0.51;


        //glNormal3d(nxx,nyy,nzz);
        glNormal3d(nx,ny,nz);
        glVertex3d(x[li], y[li],z[li]);
    }
    glEnd();

    if(ModoDibujaNormales) {
        glPushAttrib( GL_LIGHTING_BIT );
        glDisable( GL_LIGHTING );

        glBegin(GL_LINES);
        glColor3f(1,0,0);
        glVertex3d(xg,yg,zg);
        glVertex3d(xg+nxx/nnn/10,yg+nyy/nnn/10,zg+nzz/nnn/10);
        glEnd();
        glPopAttrib();
    }
}



void grid3D::draw_caraGL(vector<double>F,double minF,double maxF,int ii[4])
{

    int li;
    double x[4],y[4],z[4],v[3],xg,yg,zg,lambda=0.0,nx,ny,nz,nxx,nyy,nzz,nnn;
    GLdouble winX1,winY1,winX2,winY2,winX3,winY3,winZ,winZ2;
    xg=0;yg=0;zg=0;

    for (li=0;li<4;li++) {
        x[li]=v3D[ii[li]].x;
        y[li]=v3D[ii[li]].y;
        z[li]=v3D[ii[li]].z;
        xg+=x[li]/4;        yg+=y[li]/4;        zg+=z[li]/4;
    }
    for (li=0;li<4;li++) {
        x[li]+=(xg-x[li])*lambda;
        y[li]+=(yg-y[li])*lambda;
        z[li]+=(zg-z[li])*lambda;
    }

    nxx=nx=(y[2]-y[0])*(z[3]-z[1])-(z[2]-z[0])*(y[3]-y[1]);
    nyy=ny=(z[2]-z[0])*(x[3]-x[1])-(x[2]-x[0])*(z[3]-z[1]);
    nzz=nz=(x[2]-x[0])*(y[3]-y[1])-(y[2]-y[0])*(x[3]-x[1]);
    nnn=sqrt(sqr(nxx)+sqr(nyy)+sqr(nzz));

        ZGlobal(v);

    glEnable(GL_NORMALIZE);
    int s;
    glBegin(GL_QUADS);
    if ( nxx*v[0]+nyy*v[1]+nzz*v[2]<0) s=-1;//  glNormal3d(-nx,-ny,-nz);
    else                                s=1;//glNormal3d(nx,ny,nz);
    for (li=0;li<4;li++) {
        FuncionesOpenGL::ColorF(minF,maxF,F[ii[li]]);
        if (x[li]<xg)
            nx=nxx+(x[li]-xg)*((x[li]-xg))/10.51;
        else
            nx=nxx-(x[li]-xg)*((x[li]-xg))/10.51;
        if (y[li]<yg)
            ny=nyy+(y[li]-yg)*((y[li]-yg))/10.51;
        else
            ny=nyy-(y[li]-yg)*((y[li]-yg))/10.51;
        if  (z[li]<zg)
            nz=nzz+(z[li]-zg)*((z[li]-zg))/10.51;
        else
            nz=nzz-(z[li]-zg)*((z[li]-zg))/10.51;


        //glNormal3d(nxx,nyy,nzz);
        glNormal3d(s*nx,s*ny,s*nz);
        glVertex3d(x[li], y[li],z[li]);
    }
    glEnd();

}


void Hexa3D::draw_caraGL(int i0,int i1,int i2,int i3)
{
    int li;
    double x[4],y[4],z[4],v[3],xg,yg,zg,lambda=0.0,nx,ny,nz,nxx,nyy,nzz;
    x[0]=papa->v3D[iv[i0]].x;
    x[1]=papa->v3D[iv[i1]].x;   x[2]=papa->v3D[iv[i2]].x;   x[3]=papa->v3D[iv[i3]].x;
    y[0]=papa->v3D[iv[i0]].y;   y[1]=papa->v3D[iv[i1]].y;   y[2]=papa->v3D[iv[i2]].y;   y[3]=papa->v3D[iv[i3]].y;
    z[0]=papa->v3D[iv[i0]].z;   z[1]=papa->v3D[iv[i1]].z;   z[2]=papa->v3D[iv[i2]].z;   z[3]=papa->v3D[iv[i3]].z;
```

```cpp
    xg=0;yg=0;zg=0;
    for (li=0;li<4;li++) {
        xg+=x[li]/4;        yg+=y[li]/4;        zg+=z[li]/4;
    }
    for (li=0;li<4;li++) {
        x[li]+=(xg-x[li])*lambda;
        y[li]+=(yg-y[li])*lambda;
        z[li]+=(zg-z[li])*lambda;
    }

    nxx=nx=(y[2]-y[0])*(z[3]-z[1])-(z[2]-z[0])*(y[3]-y[1]);
    nyy=ny=(z[2]-z[0])*(x[3]-x[1])-(x[2]-x[0])*(z[3]-z[1]);
    nzz=nz=(x[2]-x[0])*(y[3]-y[1])-(y[2]-y[0])*(x[3]-x[1]);

    ZGlobal(v);
    glEnable(GL_NORMALIZE);
    glBegin(GL_QUADS);
    for (li=0;li<4;li++) {
        if (x[li]<xg)
            nx=-nxx+(x[li]-xg)*((x[li]-xg))/0.51;
        else
            nx=-nxx-(x[li]-xg)*((x[li]-xg))/0.51;
        if (y[li]<yg)
            ny=-nyy+(y[li]-yg)*((y[li]-yg))/0.51;
        else
            ny=-nyy-(y[li]-yg)*((y[li]-yg))/0.51;
        if  (z[li]<zg)
            nz=-nzz+(z[li]-zg)*((z[li]-zg))/0.51;
        else
            nz=-nzz-(z[li]-zg)*((z[li]-zg))/0.51;


        if (nxx*v[0]+nyy*v[1]+nzz*v[2]<0)
            glNormal3d(nx,ny,nz);
        else
            glNormal3d(-nx,-ny,-nz);


        //glNormal3d(nx,ny,nz);
        glVertex3d(x[li], y[li],z[li]);
    }
    glEnd();

}

void Hexa3D::draw_caraGL(vector<double>F,double minF,double maxF, int i0,int i1,int i2,int i3)
{
    int li;
    double x[4],y[4],z[4],v[3],nx,ny,nz,nxx,nyy,nzz;
    double lF[4];
    x[0]=papa->v3D[iv[i0]].x;   x[1]=papa->v3D[iv[i1]].x;   x[2]=papa->v3D[iv[i2]].x;   x[3]=papa->v3D[iv[i3]].x;
    y[0]=papa->v3D[iv[i0]].y;   y[1]=papa->v3D[iv[i1]].y;   y[2]=papa->v3D[iv[i2]].y;   y[3]=papa->v3D[iv[i3]].y;
    z[0]=papa->v3D[iv[i0]].z;   z[1]=papa->v3D[iv[i1]].z;   z[2]=papa->v3D[iv[i2]].z;   z[3]=papa->v3D[iv[i3]].z;
    lF[0]=F[iv[i0]];                lF[1]=F[iv[i1]];                lF[2]=F[iv[i2]];                   lF[3]=F[iv[i3]];

    nxx=nx=(y[2]-y[0])*(z[3]-z[1])-(z[2]-z[0])*(y[3]-y[1]);
    nyy=ny=(z[2]-z[0])*(x[3]-x[1])-(x[2]-x[0])*(z[3]-z[1]);
    nzz=nz=(x[2]-x[0])*(y[3]-y[1])-(y[2]-y[0])*(x[3]-x[1]);

    ZGlobal(v);
    glEnable(GL_NORMALIZE);
    {
        glBegin(GL_TRIANGLES );
        if ( nxx*v[0]+nyy*v[1]+nzz*v[2]<0)  glNormal3d(-nx,-ny,-nz);
        else                                glNormal3d(nx,ny,nz);

        FuncionesOpenGL::ColorF(minF,maxF,lF[0]);glVertex3d(x[0], y[0],z[0]);
//      cout<<"Hexa3D::draw_caraGL minF,MaxF,lF[0]="<<minF<<" "<<maxF<<" "<<lF[0]<<endl;
        FuncionesOpenGL::ColorF(minF,maxF,lF[1]);glVertex3d(x[1], y[1],z[1]);
        FuncionesOpenGL::ColorF(minF,maxF,lF[2]);glVertex3d(x[2], y[2],z[2]);
        glEnd();
    }

    {
        glBegin(GL_TRIANGLES );
        if ( nxx*v[0]+nyy*v[1]+nzz*v[2]<0)  glNormal3d(-nx,-ny,-nz);
        else                                glNormal3d(nx,ny,nz);

        FuncionesOpenGL::ColorF(minF,maxF,lF[0]);glVertex3d(x[0], y[0],z[0]);
        FuncionesOpenGL::ColorF(minF,maxF,lF[2]);glVertex3d(x[2], y[2],z[2]);
        FuncionesOpenGL::ColorF(minF,maxF,lF[3]);glVertex3d(x[3], y[3],z[3]);
        glEnd();
    }

}

void Hexa3D::draw_edgeGL(int i0,int i1)
{
    int li;
    vector<Vertex3D> *lv3D;
    lv3D=&(papa->v3D);

    glBegin(GL_LINES);
    glVertex3d((papa->v3D)[iv[i0]].x, (papa->v3D)[iv[i0]].y,(papa->v3D)[iv[i0]].z);
    glVertex3d(papa->v3D[iv[i1]].x, papa->v3D[iv[i1]].y,papa->v3D[iv[i1]].z);
    glEnd();

```

```
}

int grid3D::AddCara(int ib,int i0,int i1,int i2,int i3)
{
    int i;
    int is,is2,is3,ip,iip,iis,iis2,iis3;
    is=i0+i1+i2+i3;
    is2=sqr(i0)+sqr(i1)+sqr(i2)+sqr(i3);
    ip=(i0+1)*(i1+1)*(i2+1)*(i3+1);
    is3=ip/(i0+1)+ip/(i1+1)+ip/(i2+1)+ip/(i3+1);
    for (i=0;i<nCaras;i++) {
        iis=Cara[i].iv[0]+Cara[i].iv[1]+Cara[i].iv[2]+Cara[i].iv[3];
        if (iis != is) continue;
        iis2=sqr(Cara[i].iv[0])+sqr(Cara[i].iv[1])+sqr(Cara[i].iv[2])+sqr(Cara[i].iv[3]);
        if (iis2 != is2) continue;
        iip=(Cara[i].iv[0]+1)*(Cara[i].iv[1]+1)*(Cara[i].iv[2]+1)*(Cara[i].iv[3]+1);
        if (iip != ip) continue;
        iis3=iip/(Cara[i].iv[0]+1)+iip/(Cara[i].iv[1]+1)+iip/(Cara[i].iv[2]+1)+iip/(Cara[i].iv[3]+1);
        if (iis3 != is3) continue;
        break;
    }
    if (i<nCaras) { //Cara existente
        Cara[i].ih[Cara[i].nh]=ib;
        Cara[i].nh++;

        return(i);
    }
    Cara.push_back(Cara3D());
    Cara[nCaras].nv=4;
    Cara[nCaras].nh=1;
    Cara[nCaras].iv[0]=i0;
    Cara[nCaras].iv[1]=i1;
    Cara[nCaras].iv[2]=i2;
    Cara[nCaras].iv[3]=i3;
    Cara[nCaras].ih[0]=ib;
    nCaras++;
    return (nCaras-1);


}
void GeneraPoligonoInicial(R3 a, R3 b, PoligonoPlano &P)
{
    double d1x,d1y,d1z,dab;
    double d2x,d2y,d2z,dd;
    double d3x,d3y,d3z;
    double lambda=1.5;
    d1x=(b.x-a.x); d1y=(b.y-a.y); d1z=(b.z-a.z);
    dab=sqrt(sqr(d1x)+sqr(d1y)+sqr(d1z));
    if (fabs(d1x)>dab/4) {
        d2x=-d1y; d2y=d1x; d2z=0;
    } else if (fabs(d1y)>dab/4) {
        d2x=-d1y; d2y=d1x; d2z=0;
    } if (fabs(d1z)>dab/4) {
        d2x=-d1z; d2y=0; d2z=d1x;
    }
    d3x=(d1y*d2z-d1z*d2y); d3y=(d1z*d2x-d1x*d2z); d3z=(d1x*d2y-d1y*d2x);

    dd=sqrt(sqr(d2x)+sqr(d2y)+sqr(d2z)); d2x /=dd; d2y /=dd; d2z /=dd;
    dd=sqrt(sqr(d3x)+sqr(d3y)+sqr(d3z)); d3x /=dd; d3y /=dd; d3z /=dd;

    P.Dab=dab;
    P.normal.x=d1x/dab;
    P.normal.y=d1y/dab;
    P.normal.z=d1z/dab;
    P.centro.x=(a.x+b.x)/2;
    P.centro.y=(a.y+b.y)/2;
    P.centro.z=(a.z+b.z)/2;
    P.punto.clear();
    P.punto.resize(4);
    P.punto[0].x = (a.x+b.x)/2 + lambda*d2x + lambda*d3x;
    P.punto[0].y = (a.y+b.y)/2 + lambda*d2y + lambda*d3y;
    P.punto[0].z = (a.z+b.z)/2 + lambda*d2z + lambda*d3z;

    P.punto[1].x = (a.x+b.x)/2 - lambda*d2x + lambda*d3x;
    P.punto[1].y = (a.y+b.y)/2 - lambda*d2y + lambda*d3y;
    P.punto[1].z = (a.z+b.z)/2 - lambda*d2z + lambda*d3z;

    P.punto[2].x = (a.x+b.x)/2 - lambda*d2x - lambda*d3x;
    P.punto[2].y = (a.y+b.y)/2 - lambda*d2y - lambda*d3y;
    P.punto[2].z = (a.z+b.z)/2 - lambda*d2z - lambda*d3z;

    P.punto[3].x = (a.x+b.x)/2 + lambda*d2x - lambda*d3x;
    P.punto[3].y = (a.y+b.y)/2 + lambda*d2y - lambda*d3y;
    P.punto[3].z = (a.z+b.z)/2 + lambda*d2z - lambda*d3z;

}

void RecortaPoligono(R3 a, R3 b, PoligonoPlano &P)
{
    double a2,b2,KP,KQ,xpn,ypn,zpn,xnp,ynp,znp;
    int icj,ibB,k,kk,km1,kneg_ini,kneg_fin,nerase;
    a2=sqr(a.x)+sqr(a.y)+sqr(a.z);
    b2=sqr(b.x)+sqr(b.y)+sqr(b.z);
    xx2=b.x;
    yy2=b.y;
    zz2=b.z;
    //busco k tal que KQ(k)>0 (vertice dentro de la zona)
    for(k=0 ; k<P.punto.size() ; k++) {
        KQ= 2*(a.x-b.x)*P.punto[k].x+
            2*(a.y-b.y)*P.punto[k].y+
            2*(a.z-b.z)*P.punto[k].z + b2-a2;
        if (KQ>0) break;
    }
    if (k>=P.punto.size()) {
        P.punto.clear();
        return;    //Poligono fuera de la zona de interes
    }
    //hay un punto con KP>0.....(OK)
    for(kk=0 ; kk < P.punto.size() ; kk++) {
        km1=k;
        k++;if ( k>=P.punto.size() ) k=0;
        KP=KQ;
        KQ= 2*(a.x-b.x)*P.punto[k].x+
            2*(a.y-b.y)*P.punto[k].y+
            2*(a.z-b.z)*P.punto[k].z + b2-a2;
        if (KQ<0) break;
    }
    if (kk>=P.punto.size()) return; //Todoel poligono es positivo
    //Encontre KQ(k) negativo
    kneg_ini=k;
    xpn=( KP*P.punto[k].x - KQ*P.punto[km1].x ) / (KP-KQ);
    ypn=( KP*P.punto[k].y - KQ*P.punto[km1].y ) / (KP-KQ);
    zpn=( KP*P.punto[k].z - KQ*P.punto[km1].z ) / (KP-KQ);
    nerase=0;
    for(kk=0 ; kk < P.punto.size() ; kk++) {
        km1=k;
        k++;if ( k>=P.punto.size() ) k=0;
        KP=KQ;
        KQ= 2*(a.x-b.x)*P.punto[k].x+
            2*(a.y-b.y)*P.punto[k].y+
            2*(a.z-b.z)*P.punto[k].z + b2-a2;
        if (kk>1) nerase++;
        if (KQ>0) break;
    }
    kneg_fin=km1;
    xnp=( KP*P.punto[k].x - KQ*P.punto[km1].x ) / (KP-KQ);
    ynp=( KP*P.punto[k].y - KQ*P.punto[km1].y ) / (KP-KQ);
    znp=( KP*P.punto[k].z - KQ*P.punto[km1].z ) / (KP-KQ);
    if (kneg_fin==kneg_ini) {
        P.punto.insert( P.punto.begin()+kneg_ini , R3() );
        P.punto[kneg_ini  ].x=xpn;
        P.punto[kneg_ini  ].y=ypn;
        P.punto[kneg_ini  ].z=zpn;
        P.punto[kneg_ini+1].x=xnp;
        P.punto[kneg_ini+1].y=ynp;
        P.punto[kneg_ini+1].z=znp;
    } else if (nerase==0) {
        P.punto[kneg_ini].x=xpn;
        P.punto[kneg_ini].y=ypn;
        P.punto[kneg_ini].z=zpn;
        P.punto[kneg_fin].x=xnp;
        P.punto[kneg_fin].y=ynp;
        P.punto[kneg_fin].z=znp;
    } else {
        P.punto[kneg_ini].x=xpn;
        P.punto[kneg_ini].y=ypn;
        P.punto[kneg_ini].z=zpn;
        P.punto[kneg_fin].x=xnp;
        P.punto[kneg_fin].y=ynp;
        P.punto[kneg_fin].z=znp;
        if (kneg_ini+nerase < P.punto.size() ) {
            P.punto.erase( P.punto.begin()+kneg_ini+1 , P.punto.begin()+kneg_ini+nerase+1 );
        } else {
            int n1=P.punto.size()-kneg_ini-1;
            if (n1>0) P.punto.erase( P.punto.begin()+kneg_ini+1 , P.punto.begin()+kneg_ini+n1+1 );
            int n2=nerase-n1;
            P.punto.erase( P.punto.begin() , P.punto.begin()+n2 );
        }
    }
}

void grid3D::CentroCarasBloques()
{
    int i,j,k,l,iCuantosPoligonos,icj,ibb,jj;
    double xg,yg,zg,x0,y0,z0,x1,y1,z1,d1x,d1y,d1z,d2x,d2y,d2z,d3x,d3y,d3z,dij,lambda;
    double Ax,Ay,Az,AA;
    for(i=0;i<nH3D;i++) {
        xg=0;yg=0;zg=0;
        for(j=0;j<8;j++) {
            xg+=v3D[h3D[i].iv[j]].x; yg+=v3D[h3D[i].iv[j]].y; zg+=v3D[h3D[i].iv[j]].z;
        }
        h3D[i].centro.x=xg/8; h3D[i].centro.y=yg/8; h3D[i].centro.z=zg/8;
    }
    for(i=0;i<nCaras;i++) {
        xg=0;yg=0;zg=0;
        Cara[i].nVertexPolig=0;
        for(j=0;j<4;j++) {
            xg+=v3D[Cara[i].iv[j]].x; yg+=v3D[Cara[i].iv[j]].y; zg+=v3D[Cara[i].iv[j]].z;
        }
        Cara[i].centro.x=xg/4; Cara[i].centro.y=yg/4; Cara[i].centro.z=zg/4;
        Ax = (v3D[Cara[i].iv[1]].y - v3D[Cara[i].iv[0]].y) * (v3D[Cara[i].iv[2]].z - v3D[Cara[i].iv[0]].z)
            -(v3D[Cara[i].iv[1]].z - v3D[Cara[i].iv[0]].z) * (v3D[Cara[i].iv[2]].y - v3D[Cara[i].iv[0]].y);
        Ay = (v3D[Cara[i].iv[1]].z - v3D[Cara[i].iv[0]].z) * (v3D[Cara[i].iv[2]].x - v3D[Cara[i].iv[0]].x)
            -(v3D[Cara[i].iv[1]].x - v3D[Cara[i].iv[0]].x) * (v3D[Cara[i].iv[2]].z - v3D[Cara[i].iv[0]].z);
        Az = (v3D[Cara[i].iv[1]].x - v3D[Cara[i].iv[0]].x) * (v3D[Cara[i].iv[2]].y - v3D[Cara[i].iv[0]].y)
            -(v3D[Cara[i].iv[1]].y - v3D[Cara[i].iv[0]].y) * (v3D[Cara[i].iv[2]].x - v3D[Cara[i].iv[0]].x);
        Ax+= (v3D[Cara[i].iv[2]].y - v3D[Cara[i].iv[0]].y) * (v3D[Cara[i].iv[3]].z - v3D[Cara[i].iv[0]].z)
            -(v3D[Cara[i].iv[2]].z - v3D[Cara[i].iv[0]].z) * (v3D[Cara[i].iv[3]].y - v3D[Cara[i].iv[0]].y);
        Ay+= (v3D[Cara[i].iv[2]].z - v3D[Cara[i].iv[0]].z) * (v3D[Cara[i].iv[3]].x - v3D[Cara[i].iv[0]].x)
            -(v3D[Cara[i].iv[2]].x - v3D[Cara[i].iv[0]].x) * (v3D[Cara[i].iv[3]].z - v3D[Cara[i].iv[0]].z);
        Az+= (v3D[Cara[i].iv[2]].x - v3D[Cara[i].iv[0]].x) * (v3D[Cara[i].iv[3]].y - v3D[Cara[i].iv[0]].y)
            -(v3D[Cara[i].iv[2]].y - v3D[Cara[i].iv[0]].y) * (v3D[Cara[i].iv[3]].x - v3D[Cara[i].iv[0]].x);

        double Norma2=sqrt(sqr(Ax)+sqr(Ay)+sqr(Az));
        Cara[i].Area = Norma2/2;
        Cara[i].normal.x = Ax/Norma2;
        Cara[i].normal.y = Ay/Norma2;
        Cara[i].normal.z = Az/Norma2;
        Cara[i].normal.L = 1.0;
    }
#ifdef JSM_NO_HACER_ESTO
    for(i=0;i<nH3D;i++) {
        xg=0;yg=0;zg=0;AA=0;
        for(j=0;j<6;j++) {
            int         ic=h3D[i].icara[j];
            xg+=Cara[ic].centro.x * Cara[ic].Area ;
            yg+=Cara[ic].centro.y * Cara[ic].Area;
            zg+=Cara[ic].centro.z * Cara[ic].Area;
            AA+= Cara[ic].Area;
        }
        //      h3D[i].centro.x=xg/AA; h3D[i].centro.y=yg/AA; h3D[i].centro.z=zg/AA;
    }
#endif
}

void grid3D::generaPoligonos2(int CuantosPoligonos)
{
    int i,j,k,l,iCuantosPoligonos,icj,ibb,jj;
    double xg,yg,zg,x0,y0,z0,x1,y1,z1,d1x,d1y,d1z,d2x,d2y,d2z,d3x,d3y,d3z,dij,lambda;
    double Ax,Ay,Az,AA;
    QuienGeneraPoligonos=2;

    //Primer nivel de vecinos
    for(i=0;i<nH3D;i++) {
        h3D[i].vecino.resize(6);
        h3D[i].tipo_vecino.resize(6);
        for (j=0;j<6;j++) {
            icj=h3D[i].icara[j];
            if (Cara[icj].nh>1) {
                h3D[i].tipo_vecino[j]=ES_BLOQUE;
                ibb=Cara[icj].ih[0]; if (ibb==i) ibb=Cara[icj].ih[1];
                h3D[i].vecino[j]=ibb;
            } else {
                h3D[i].tipo_vecino[j]=ES_CARA;;
                h3D[i].vecino[j]=icj;
            }
        }
    }
    //segundo nivel de vecinos (los vecinos de mis BLOQUES vecinos)
    for(i=0;i<nH3D;i++) {
        int nvecinos_level1=h3D[i].vecino.size();
        for (j=0 ; j<nvecinos_level1 ;j++) {
            if (h3D[i].tipo_vecino[j]==ES_CARA) continue;
            ibb=h3D[i].vecino[j];
            for (k=0 ; k<h3D[ibb].vecino.size() ;k++) {
                if (h3D[ibb].tipo_vecino[k]==ES_CARA) continue;
                if (h3D[ibb].vecino[k]==i) continue;
                for (l=0  ; l<h3D[i].vecino.size() ;l++) {
                    if (h3D[i].tipo_vecino[l]==ES_CARA) continue;
                    if (h3D[i].vecino[l]==h3D[ibb].vecino[k]) break;
                }
                if (l>=h3D[i].vecino.size()) {//es un nuevo vecino
                    h3D[i].vecino.push_back(0);
                    h3D[i].tipo_vecino.push_back(0);
                    h3D[i].vecino[l]=h3D[ibb].vecino[k];
                    h3D[i].tipo_vecino[l]=h3D[ibb].tipo_vecino[k];
                }
            }
        }
    }
    for(i=0;i<nH3D;i++) {
        R3 VA,VB,NB;
        VA=h3D[i].centro;
        for (j=0 ; j< h3D[i].vecino.size() ; j++) {
            if (h3D[i].tipo_vecino[j]==ES_CARA) {
                VB=Cara[ h3D[i].vecino[j] ].centro;
                NB=Cara[ h3D[i].vecino[j] ].normal;
                if (Cara[ h3D[i].vecino[j] ].iBC ==1||Cara[ h3D[i].vecino[j] ].iBC >=10) {
                    double lambda=2*ppuntodiff(NB,VB,VA);
                    Cara[ h3D[i].vecino[j] ].centro.x = VA.x+lambda*NB.x;
                    Cara[ h3D[i].vecino[j] ].centro.y = VA.y+lambda*NB.y;
                    Cara[ h3D[i].vecino[j] ].centro.z = VA.z+lambda*NB.z;
                    VB=Cara[ h3D[i].vecino[j] ].centro;
                }
            }
        }
    }

    int navance=0;
    for(i=0;i<nH3D;i++) {
        if (i>=navance*nH3D/20) {
            printf(".%d",navance);fflush(stdout);
            navance++;
        }
        h3D[i].Poligono.resize( h3D[i].vecino.size() );
        R3 VA,VB;
        VA=h3D[i].centro;
        for (j=0 ; j< h3D[i].vecino.size() ; j++) {
            if (h3D[i].tipo_vecino[j]==ES_BLOQUE) {
                VB=h3D[ h3D[i].vecino[j] ].centro;
            }
            if (h3D[i].tipo_vecino[j]==ES_CARA) {
                VB=Cara[ h3D[i].vecino[j] ].centro;
            }
            GeneraPoligonoInicial(VA,VB, h3D[i].Poligono[j]);
            for (jj=0 ; jj< h3D[i].vecino.size() ; jj++) {
                if (jj==j) continue;
                if (h3D[i].tipo_vecino[jj]==ES_BLOQUE) {
                    VB=h3D[ h3D[i].vecino[jj] ].centro;
                }
                if (h3D[i].tipo_vecino[jj]==ES_CARA) {
                    VB=Cara[ h3D[i].vecino[jj] ].centro;
                }
                RecortaPoligono(VA,VB, h3D[i].Poligono[j]);
                if (h3D[i].Poligono[j].punto.size() == 0) {
                    h3D[i].Poligono.erase(h3D[i].Poligono.begin()+j);
                    h3D[i].vecino.erase(h3D[i].vecino.begin()+j);
                    h3D[i].tipo_vecino.erase(h3D[i].tipo_vecino.begin()+j);
                    j--;
                    break;
                }
            }
        }
    }
    printf("\n");
//calculos de areas y longitudes de trazos
    for (i=0;i<nH3D;i++) {
        for (k=0; k<h3D[i].Poligono.size(); k++) {
            vector<R3> *Pto;
            Pto= &(h3D[i].Poligono[k].punto);
            double Area,Ax,Ay,Az;
            Area=0;
            for (j=1; j<Pto->size(); j++) {
                (*Pto)[j].L = sqrt(   sqr( (*Pto)[j].x- (*Pto)[j-1].x )
                                        + sqr( (*Pto)[j].y- (*Pto)[j-1].y )
                                        + sqr( (*Pto)[j].z- (*Pto)[j-1].z ) );
                if ((*Pto)[j].L <1e-5) {
                    (*Pto).erase( (*Pto).begin() + j);
                    j--;
                    continue;
                }
                if (j<2) continue;
                Az=((*Pto)[j-1].x-(*Pto)[0].x)*((*Pto)[j].y-(*Pto)[0].y)-((*Pto)[j-1].y-(*Pto)[0].y)*((*Pto)[j].x-(*Pto)[0].x);
                Ax=((*Pto)[j-1].y-(*Pto)[0].y)*((*Pto)[j].z-(*Pto)[0].z)-((*Pto)[j-1].z-(*Pto)[0].z)*((*Pto)[j].y-(*Pto)[0].y);
                Ay=((*Pto)[j-1].z-(*Pto)[0].z)*((*Pto)[j].x-(*Pto)[0].x)-((*Pto)[j-1].x-(*Pto)[0].x)*((*Pto)[j].z-(*Pto)[0].z);
                Area+=sqrt(sqr(Ax)+sqr(Ay)+sqr(Az))/2;
            }
            h3D[i].Poligono[k].Area=Area;
            if (Area <1e-10) {
                h3D[i].Poligono.erase(h3D[i].Poligono.begin()+k);
                h3D[i].vecino.erase(h3D[i].vecino.begin()+k);
                h3D[i].tipo_vecino.erase(h3D[i].tipo_vecino.begin()+k);
                k--;
            }
        }
    }
}

void grid3D::generaPoligonos(int CuantosPoligonos)
{
    int i,j,iCuantosPoligonos;
    double xg,yg,zg,x0,y0,z0,x1,y1,z1,d1x,d1y,d1z,d2x,d2y,d2z,d3x,d3y,d3z,dij,lambda;
    QuienGeneraPoligonos=1;
    for(i=0;i<nH3D;i++) {
        xg=0;yg=0;zg=0;
        for(j=0;j<8;j++) {
            xg+=v3D[h3D[i].iv[j]].x; yg+=v3D[h3D[i].iv[j]].y; zg+=v3D[h3D[i].iv[j]].z;
        }
        h3D[i].centro.x=xg/8; h3D[i].centro.y=yg/8; h3D[i].centro.z=zg/8;
    }
    for(i=0;i<nCaras;i++) {
        xg=0;yg=0;zg=0;
        Cara[i].nVertexPolig=0;
        for(j=0;j<4;j++) {
            xg+=v3D[Cara[i].iv[j]].x; yg+=v3D[Cara[i].iv[j]].y; zg+=v3D[Cara[i].iv[j]].z;
        }
        Cara[i].centro.x=xg/4; Cara[i].centro.y=yg/4; Cara[i].centro.z=zg/4;
    }
    iCuantosPoligonos=0;
    for(i=0;i<nCaras;i++) {
            //nn++;
            x0=h3D[Cara[i].ih[0]].centro.x; y0=h3D[Cara[i].ih[0]].centro.y; z0=h3D[Cara[i].ih[0]].centro.z;
        if (Cara[i].nh==2) {
            x1=h3D[Cara[i].ih[1]].centro.x; y1=h3D[Cara[i].ih[1]].centro.y; z1=h3D[Cara[i].ih[1]].centro.z;
        } else {
            x1=Cara[i].centro.x;    y1=Cara[i].centro.y;    z1=Cara[i].centro.z;
        }

```

```
//          Cara[i].centro.x=(x0+x1)/2; Cara[i].centro.y=(y0+y1)/2; Cara[i].centro.z=(z0+z1)/2;
            d1x=(x1-x0); d1y=(y1-y0); d1z=(z1-z0);
            dij=sqrt(sqr(d1x)+sqr(d1y)+sqr(d1z));
            if (fabs(d1x)>dij/4) {
                d2x=-d1y; d2y=d1x; d2z=0;
            } else if (fabs(d1y)>dij/4) {
                d2x=-d1y; d2y=d1x; d2z=0;
            } if (fabs(d1z)>dij/4) {
                d2x=-d1z; d2y=0; d2z=d1x;
            }
            d3x=(d1y*d2z-d1z*d2y); d3y=(d1z*d2x-d1x*d2z); d3z=(d1x*d2y-d1y*d2x);

            Cara[i].Dij=dij;
            Cara[i].nx=d1x/dij; Cara[i].ny=d1y/dij; Cara[i].nz=d1z/dij;

            dij=sqrt(sqr(d2x)+sqr(d2y)+sqr(d2z)); d2x /=dij; d2y /=dij; d2z /=dij;
            dij=sqrt(sqr(d3x)+sqr(d3y)+sqr(d3z)); d3x /=dij; d3y /=dij; d3z /=dij;
            lambda=1.5;

            Cara[i].vPolig.punto.clear();
            Cara[i].vPolig.punto.resize(4);
            Cara[i].vPolig.punto[0].x=x0+d1x/2+lambda*d2x+lambda*d3x;
            Cara[i].vPolig.punto[0].y=y0+d1y/2+lambda*d2y+lambda*d3y;
            Cara[i].vPolig.punto[0].z=z0+d1z/2+lambda*d2z+lambda*d3z;

            Cara[i].vPolig.punto[1].x=x0+d1x/2-lambda*d2x+lambda*d3x;
            Cara[i].vPolig.punto[1].y=y0+d1y/2-lambda*d2y+lambda*d3y;
            Cara[i].vPolig.punto[1].z=z0+d1z/2-lambda*d2z+lambda*d3z;

            Cara[i].vPolig.punto[2].x=x0+d1x/2-lambda*d2x-lambda*d3x;
            Cara[i].vPolig.punto[2].y=y0+d1y/2-lambda*d2y-lambda*d3y;
            Cara[i].vPolig.punto[2].z=z0+d1z/2-lambda*d2z-lambda*d3z;

            Cara[i].vPolig.punto[3].x=x0+d1x/2+lambda*d2x-lambda*d3x;
            Cara[i].vPolig.punto[3].y=y0+d1y/2+lambda*d2y-lambda*d3y;
            Cara[i].vPolig.punto[3].z=z0+d1z/2+lambda*d2z-lambda*d3z;

            Cara[i].nVertexPolig=4;

            //          x0=h3D[Cara[i].ih[0]].centro.x; y0=h3D[Cara[i].ih[0]].centro.y; z0=h3D[Cara[i].ih[0]].centro.z;
            //          x1=h3D[Cara[i].ih[1]].centro.x; y1=h3D[Cara[i].ih[1]].centro.y; z1=h3D[Cara[i].ih[1]].centro.z;

            xx0=x0;yy0=y0;zz0=z0;
            xx1=x1;yy1=y1;zz1=z1;
            xx2=x1;yy2=y1;zz2=z1;
                        xxi=i;
            Dibuja3puntos=2;
            if(++iCuantosPoligonos>CuantosPoligonos && CuantosPoligonos>0) {
                return;
            }

            for (int iv=0;iv<Cara[i].nh;iv++) {
                int ibA=Cara[i].ih[iv];
                int icj,ibB;
                for (j=0;j<6;j++) {
                    icj=h3D[ibA].icara[j];
                    if (icj==i) continue;
                    if (Cara[icj].nh>1) {
                        ibB=Cara[icj].ih[0]; if (ibB==ibA) ibB=Cara[icj].ih[1];
                        RecortaPoligono(h3D[ibA].centro, h3D[ibB].centro, Cara[i].vPolig);
                        Cara[i].nVertexPolig = Cara[i].vPolig.punto.size();
                    }
                    if (Cara[icj].nh==1) {
                        RecortaPoligono(h3D[ibA].centro, Cara[icj].centro, Cara[i].vPolig);
                        Cara[i].nVertexPolig = Cara[i].vPolig.punto.size();
                    }
                    if(++iCuantosPoligonos>CuantosPoligonos && CuantosPoligonos>0) {
                        return;
                    }

                }
            }
        }

}



void grid3D::GeneraCaras(int inicia)
{
    int ib,i;
    static int cuantas=0;
    cout<<"grid3D::GeneraCaras: inicia="<<inicia<<endl;
    if (inicia<0) {
        cuantas=0;
    }

    if (cuantas==0) {
        nCaras=0;
    }
    for (i=0;i<inicia;i++){
        if (cuantas<nH3D) {
            //    2------3
            //   /|     /|
            //  6-|----7 |
            //  | |    | |
            //  | 0----|-1
            //  |/     |/
            //  4------5

            ib=cuantas; cuantas++;
            if(1==0) cout<<"ib="<<ib<<"\tinicia="<<inicia<<endl;
            h3D[ib].icara[0]=AddCara(ib,h3D[ib].iv[0],h3D[ib].iv[2],h3D[ib].iv[3],h3D[ib].iv[1]);
            h3D[ib].icara[1]=AddCara(ib,h3D[ib].iv[0],h3D[ib].iv[4],h3D[ib].iv[6],h3D[ib].iv[2]);
            h3D[ib].icara[2]=AddCara(ib,h3D[ib].iv[0],h3D[ib].iv[1],h3D[ib].iv[5],h3D[ib].iv[4]);
            h3D[ib].icara[3]=AddCara(ib,h3D[ib].iv[4],h3D[ib].iv[5],h3D[ib].iv[7],h3D[ib].iv[6]);
            h3D[ib].icara[4]=AddCara(ib,h3D[ib].iv[1],h3D[ib].iv[3],h3D[ib].iv[7],h3D[ib].iv[5]);
            h3D[ib].icara[5]=AddCara(ib,h3D[ib].iv[2],h3D[ib].iv[6],h3D[ib].iv[7],h3D[ib].iv[3]);
        }
    }
}


int nParticulas=200;
double ThetaMax,ThetaMin,dTheta_med;
static Vector<double> Particulas[maxpasadas+1][3];
int primerdrawVelGL=1;


int npasadas=1;

void PosINI(int i)
{
    double x,y,z,t;
    int j,k;

    x=(   (double)rand() / ((double)(RAND_MAX)+(double)(1)) )*Dominio_Rmax;
    y=(   (double)rand() / ((double)(RAND_MAX)+(double)(1)) )*Dominio_Rmax;
    z=(   (double)rand() / ((double)(RAND_MAX)+(double)(1)) )*Dominio_Hsup;
    t=ThetaMax*(   (double)(1.0*rand()) / ((double)(RAND_MAX)+(double)(1)) );
    Particulas[0][0][i]= 1.5*Dominio_Rint*cos(t);
    Particulas[0][1][i]= 1.5*Dominio_Rint*sin(t);
//  Particulas[0][0][i]= x ;
//  Particulas[0][1][i]= y ;
    Particulas[0][2][i]= z ;
    for (j=0;j<3;j++) {
        for (k=0;k<maxpasadas;k++) Particulas[k+1][j][i]=Particulas[0][j][i];
    }
}

void grid3D::drawVelGL(vector<double> U,vector<double> V,vector<double> W)
{
//  cout<<"grid3D::drawVelGL"<<"primerdrawVelGL="<<primerdrawVelGL<<"nParticulas="<<nParticulas<<endl;
    static double dt=0.1*sqr(Dominio_Rmax);
    static int ipasadas=0;
    double d2,d2min,x,y,z,dx,dy,dz,t;
    int i,j,k,jmin;
    if (primerdrawVelGL) {
        for (k=0;k<=maxpasadas;k++) {
            for (j=0;j<3;j++) {
                Particulas[k][j].init(nParticulas);
            }
        }
        for (i=0;i<nParticulas;i++) {
            PosINI(i);
        }
    }

    primerdrawVelGL=0;
    ipasadas++;

    for (i=0;i<nParticulas;i++) {
        x=Particulas[0][0][i];
        y=Particulas[0][1][i];
        z=Particulas[0][2][i];
        d2min=1e10;
        for (j=0;j<nV3D;j++) {
            d2=sqr(x-v3D[j].x)+sqr(y-v3D[j].y)+sqr(z-v3D[j].z);
            if (d2<d2min) {
                d2min=d2;
                jmin=j;
            }
        }
        //cout<<"factorV="<<factorV<<endl;
        double UUU=sqrt(sqr(U[jmin])+sqr(V[jmin])+sqr(W[jmin]));
        dx=U[jmin]*dt/npasadas*factorV/UUU*factorVh;
        dy=V[jmin]*dt/npasadas*factorV/UUU*factorVh;
        dz=W[jmin]*dt/npasadas*factorV/UUU;

        if (ipasadas>npasadas) {
            for (k=maxpasadas-1;k>=0;k--)
                for (j=0;j<3;j++)
                    Particulas[k+1][j][i]=Particulas[k][j][i];
        }

        x=Particulas[0][0][i]+=dx;
        y=Particulas[0][1][i]+=dy;
        z=Particulas[0][2][i]+=dz;

        if ( sqrt(sqr(x-2*Dominio_Rmax)+sqr(y)) < Dominio_Rint || x<xmin || x>xmax || y<ymin || y>ymax || z<zmin || z>zmax ) {
            PosINI(i);
        }
    }

```

```cpp
    for (i=0;i<nParticulas;i++) {
        for (k=0;k<maxpasadas;k++) {
            glBegin(GL_LINES);
            glVertex3d(Particulas[k][0][i],Particulas[k][1][i], Particulas[k][2][i]);
            glVertex3d(Particulas[k+1][0][i],Particulas[k+1][1][i], Particulas[k+1][2][i]);
            glEnd();
        }
        glPushMatrix();
        glTranslated(Particulas[0][0][i],Particulas[0][1][i], Particulas[0][2][i]);
//      cout<<"esfera"<<endl;
        FuncionesOpenGL::esfera(0.5*Dominio_Rint,4);
        glPopMatrix();
    }
    if (ipasadas>npasadas) ipasadas=0;


}


void grid3D::drawVelGL2(vector<double> UU,vector<double> VV,vector<double> WW)
{
    double x,y,z;
    int i;
    glPushAttrib( GL_LIGHTING_BIT );
    glDisable( GL_LIGHTING );

    for (i=0;i<nH3D;i++) {
        x=h3D[i].centro.x;
        y=h3D[i].centro.y;
        z=h3D[i].centro.z;

        glBegin(GL_LINES);
        glColor3f(1,0,0);
        glVertex3d(x,y,z);
        glVertex3d(x+UU[i]*factorV,y +VV[i]*factorV,z+WW[i]*factorV);
        glEnd();
    }
    glPopAttrib();


}


void grid3D::drawVoronoi()
{
    int i,j,k;
    static int xxip=0;
    GLdouble winX1,winY1,winZ,winX2,winY2,winX3,winY3,nx,ny,nz;

    FuncionesOpenGL::modelview_calculado=false;

    glEnable(GL_NORMALIZE);
    if (QuienGeneraPoligonos==1) {
        for (i=0;i<nCaras;i++) {
            nx=Cara[i].nx; ny=Cara[i].ny;nz=Cara[i].nz;
            if (Cara[i].nVertexPolig >2) {
                FuncionesOpenGL::World2Win(Cara[i].vPolig.punto[0].x , Cara[i].vPolig.punto[0].y , Cara[i].vPolig.punto[0].z,&winX1,&winY1,&winZ);
                FuncionesOpenGL::World2Win(Cara[i].vPolig.punto[1].x , Cara[i].vPolig.punto[1].y , Cara[i].vPolig.punto[1].z,&winX2,&winY2,&winZ);
                FuncionesOpenGL::World2Win(Cara[i].vPolig.punto[Cara[i].nVertexPolig-1].x , Cara[i].vPolig.punto[Cara[i].nVertexPolig-1].y ,
                    Cara[i].vPolig.punto[Cara[i].nVertexPolig-1].z,&winX3,&winY3,&winZ);
                if ((winX2-winX1)*(winY3-winY1)-(winY2-winY1)*(winX3-winX1) <0 ){
                    nx = -nx; ny = -ny; nz = -nz; ;
                }
            }
            if (Dibuja3puntos && xxi==i && xxi != xxip) {
                xxip=xxi;
                for (j=0;j<Cara[i].nVertexPolig;j++)
                    printf("Cara[%d].vPolig.punto[%d]={%6.2f , %6.2f , %6.2f}\n",i,j,Cara[i].vPolig.punto[j].x,Cara[i].vPolig.punto[j].y,Cara[i].vPolig.punto[j].z);
            }
            for (j=2;j<Cara[i].nVertexPolig;j++) {

                if (Dibuja3puntos && xxi==i) {
                    FuncionesOpenGL::material(3);   }
                else  {FuncionesOpenGL::material(2);}



                glBegin(GL_TRIANGLES );
                glNormal3d(nx , ny , nz );
                glVertex3d(Cara[i].vPolig.punto[0].x,Cara[i].vPolig.punto[0].y,Cara[i].vPolig.punto[0].z);
                glVertex3d(Cara[i].vPolig.punto[j-1].x,Cara[i].vPolig.punto[j-1].y,Cara[i].vPolig.punto[j-1].z);
                glVertex3d(Cara[i].vPolig.punto[j].x,Cara[i].vPolig.punto[j].y,Cara[i].vPolig.punto[j].z);
                glEnd();

            }
        }
    } else if (QuienGeneraPoligonos==2) {
        for (i=0;i<nH3D;i++) {
            h3D[i].dibujado.assign( h3D[i].Poligono.size(),0);
        }
        for (i=0;i<nH3D;i++) {
            for (k=0; k<h3D[i].Poligono.size(); k++) {
                if (h3D[i].dibujado[k] == 1) continue;
                vector<R3> *Pto;
                Pto= &(h3D[i].Poligono[k].punto);
                nx=h3D[i].Poligono[k].normal.x; ny=h3D[i].Poligono[k].normal.y;nz=h3D[i].Poligono[k].normal.z;
                if (Pto->size() >2) {
                    FuncionesOpenGL::World2Win( (*Pto)[0].x , (*Pto)[0].y , (*Pto)[0].z,&winX1,&winY1,&winZ);
                    FuncionesOpenGL::World2Win( (*Pto)[Pto->size()/2].x , (*Pto)[Pto->size()/2].y ,  (*Pto)[Pto->size()/2].z,&winX2,&winY2,&winZ);
                    FuncionesOpenGL::World2Win( (*Pto)[Pto->size()-1].x , (*Pto)[Pto->size()-1].y ,
                        (*Pto)[Pto->size()-1].z,&winX3,&winY3,&winZ);
                    if ((winX2-winX1)*(winY3-winY1)-(winY2-winY1)*(winX3-winX1) <0 ){
                        nx = -nx; ny = -ny; nz = -nz; ;
                    }
                }
                if (Dibuja3puntos && xxi==i) {
                    FuncionesOpenGL::material(2);   }
                else  {FuncionesOpenGL::material(3);}
                for (j=2; j<Pto->size(); j++) {
                    glBegin(GL_TRIANGLES );
                    glNormal3d(nx , ny , nz );
                    glVertex3d( (*Pto)[  0].x, (*Pto)[  0].y, (*Pto)[0].z);
                    glVertex3d( (*Pto)[j-1].x, (*Pto)[j-1].y, (*Pto)[j-1].z);
                    glVertex3d( (*Pto)[  j].x, (*Pto)[  j].y, (*Pto)[j].z);
                    glEnd();

                }
                if(ModoDibujaNormales) {
                    double xg,yg,zg,nnn;
                    glPushAttrib( GL_LIGHTING_BIT );
                    glDisable( GL_LIGHTING );

                    glBegin(GL_LINES);
                    glColor3f(1,0,0);
                    xg=h3D[i].Poligono[k].centro.x;
                    yg=h3D[i].Poligono[k].centro.y;
                    zg=h3D[i].Poligono[k].centro.z;
                    glVertex3d(xg,yg,zg);
                    nnn=sqrt(sqr(nx)+sqr(ny)+sqr(nz));
                    glVertex3d(xg+nx/nnn/10,yg+ny/nnn/10,zg+nz/nnn/10);
                    glEnd();
                    glPopAttrib();
                }

                h3D[i].dibujado[k]=1;
                if (h3D[i].tipo_vecino[k]==ES_BLOQUE) {
                    for (j=0; j<h3D[ h3D[i].tipo_vecino[k] ].dibujado.size(); j++) {
                        if (   h3D[ h3D[i].vecino[k] ].tipo_vecino[j] ==ES_BLOQUE
                            && h3D[ h3D[i].vecino[k] ].vecino[j] == i) {

                            h3D[ h3D[i].vecino[k] ].dibujado[j]=1;
                            break;
                        }

                    }
                }
            }

        }
        if (Dibuja3puntos) {
            glTranslatef(xx0,yy0,zz0);
            FuncionesOpenGL::material(1);   FuncionesOpenGL::esfera(0.02,3);
            glTranslatef(xx1-xx0,yy1-yy0,zz1-zz0);
            FuncionesOpenGL::material(2);   FuncionesOpenGL::esfera(0.02,3);
            glTranslatef(xx2-xx1,yy2-yy1,zz2-zz1);
            FuncionesOpenGL::material(2);   FuncionesOpenGL::esfera(0.02,3);
            glTranslatef(-xx2,-yy2,-zz2);
        }
    }
}


void grid3D::minmax()
{
    int i;
    double x,y,z;
    xmin=xmax=v3D[0].x;
    ymin=ymax=v3D[0].y;
    zmin=zmax=v3D[0].z;
    for (i=1;i<nV3D;i++) {
        x=v3D[i].x;
        y=v3D[i].y;
        z=v3D[i].z;
        if (x < xmin) xmin=x;
        if (x > xmax) xmax=x;
        if (y < ymin) ymin=y;
        if (y > ymax) ymax=y;
        if (z < zmin) zmin=z;
        if (z > zmax) zmax=z;
    }
}

void grid3D::drawGL()
{
    int i;
    char s[100];
    static int version=1;//0:dibuja los cubos, 1:dibuja las caras


    FuncionesOpenGL::ObtieneMatrices();
    if (version==0) {
        for (i=0;i<nH3D;i++) {
            //    2------3
            //   /|     /|
            //  6-|----7 |
            //  | |    | |
            //  | 0----|-1
            //  |/     |/
            //  4------5

            h3D[i].draw_caraGL(0,1,3,2);
            h3D[i].draw_caraGL(0,2,6,4);
            h3D[i].draw_caraGL(0,4,5,1);
        }
    } else if (version ==1) {
        if (Modo_DibujaCentroBloques) {
            for (i=0;i<nH3D;i++) {
                glTranslatef(h3D[i].centro.x,h3D[i].centro.y,h3D[i].centro.z);
                FuncionesOpenGL::material(1);   FuncionesOpenGL::esfera(0.01/Escala,3);
                glTranslatef(-h3D[i].centro.x,-h3D[i].centro.y,-h3D[i].centro.z);
            }
        }
        for (i=0;i<nCaras;i++) {
            if (0*Modo_DibujaCentroCaras) {
                glTranslatef(Cara[i].centro.x,Cara[i].centro.y,Cara[i].centro.z);
                FuncionesOpenGL::material(0);   FuncionesOpenGL::esfera(0.04/Escala,3);
                glTranslatef(-Cara[i].centro.x,-Cara[i].centro.y,-Cara[i].centro.z);
            }
            if (ModoDibujaFrontera && Cara[i].nh ==1) {
                FuncionesOpenGL::material(Cara[i].iBC+10);draw_caraGL(Cara[i].iv);
            }
            if (ModoDibujaInterior && Cara[i].nh >1) {
                FuncionesOpenGL::material(2);draw_caraGL(Cara[i].iv);
                if (0*Modo_DibujaCentroCaras) {
                    glTranslatef(Cara[i].centro.x,Cara[i].centro.y,Cara[i].centro.z);
                    FuncionesOpenGL::material(1);   FuncionesOpenGL::esfera(0.04,3);
                    glTranslatef(-Cara[i].centro.x,-Cara[i].centro.y,-Cara[i].centro.z);
                }
            }
        }
    }


    if (Modo_DibujaEdges) {
        glPushAttrib( GL_LIGHTING_BIT );
        glDisable( GL_LIGHTING );
        for (i=0;i<nH3D;i++) {
            //    2------3
            //   /|     /|
            //  6-|----7 |
            //  | |    | |
            //  | 0----|-1
            //  |/     |/
            //  4------5

            h3D[i].draw_edgeGL(6,7);
            h3D[i].draw_edgeGL(3,7);
            h3D[i].draw_edgeGL(5,7);
        }

        glPopAttrib();
    }



    if (MODO_NumeraH){

        FuncionesOpenGL::material(0);

        for (i=0;i<nH3D;i++) {
            char *p;
            glPushMatrix();

            glTranslatef(h3D[i].centro.x,h3D[i].centro.y,h3D[i].centro.z);
            //glTranslatef(0,0,.5);


            glMultMatrixf((GLfloat *)MatrizRotacionGlobalINV);
            glScalef(NumEscala,NumEscala,NumEscala);
            float y0=4.0/Escala,dy=1.7/Escala;
            glRasterPos3f(-3/Escala, y0-=dy ,0);
            sprintf(s,"%d",i);
            for(p = s; *p; p++) {
                glutBitmapCharacter(GLUT_BITMAP_8_BY_13, *p);
            }
            if (MODO_NumeraFF)  {
                glRasterPos3f(-3/Escala, y0-=dy ,0);
                sprintf(s,"FF=%.3f",FF[i]);
                for(p = s; *p; p++) {
                    glutBitmapCharacter(GLUT_BITMAP_8_BY_13, *p);
                }
                glRasterPos3f(-3/Escala, y0-=dy ,0);
                sprintf(s,"U=%.3f",sqrt(sqr(UU[i])+sqr(VV[i])+sqr(WW[i])));
                for(p = s; *p; p++) {
                    glutBitmapCharacter(GLUT_BITMAP_8_BY_13, *p);
                }
                glRasterPos3f(-3/Escala, y0-=dy ,0);
                sprintf(s,"d=%.3f",sqrt(sqr(h3D[i].centro.x)+sqr(h3D[i].centro.y)));
                for(p = s; *p; p++) {
                    glutBitmapCharacter(GLUT_BITMAP_8_BY_13, *p);
                }
                /*
                glRasterPos3f(-3/Escala, y0-=dy ,0);
                sprintf(s,"VV=%.3f",VV[i]);
                for(p = s; *p; p++) {
                    glutBitmapCharacter(GLUT_BITMAP_8_BY_13, *p);
                }
                glRasterPos3f(-3/Escala, y0-=dy ,0);
                sprintf(s,"WW=%.3f",WW[i]);
                for(p = s; *p; p++) {
                    glutBitmapCharacter(GLUT_BITMAP_8_BY_13, *p);
                }
                */
            }

            //          FuncionesOpenGL::Print(s);
            glPopMatrix();
        }
    }
}


void grid3D::drawGL(vector<double> F)
{
    int i;
    char s[100];
    double minF,maxF;

    maxF=minF=F[0];
    for (i=0;i<nV3D;i++) {
        if (minF>F[i]) minF=F[i];
        if (maxF<F[i]) maxF=F[i];
    }
    glPushAttrib( GL_LIGHTING_BIT );
    //  glEnable(GL_COLOR_MATERIAL);
    //  glDisable( GL_LIGHTING );

    for (i=0;i<nCaras;i++) {
        //    2------3
        //   /|     /|
        //  6-|----7 |
        //  | |    | |
        //  | 0----|-1
        //  |/     |/
        //  4------5

        //TODO

        if (ModoDibujaFrontera && Cara[i].nh ==1) {
            FuncionesOpenGL::material(Cara[i].iBC+10);draw_caraGL(F,minF,maxF,Cara[i].iv);
        }
        if (ModoDibujaInterior && Cara[i].nh >1) {
            FuncionesOpenGL::material(2);draw_caraGL(F,minF,maxF,Cara[i].iv);
            if (0*Modo_DibujaCentroCaras) {
                glTranslatef(Cara[i].centro.x,Cara[i].centro.y,Cara[i].centro.z);
                FuncionesOpenGL::material(1);   FuncionesOpenGL::esfera(0.04,3);
                glTranslatef(-Cara[i].centro.x,-Cara[i].centro.y,-Cara[i].centro.z);
            }
        }
    }



    if (Modo_DibujaEdges) {
        for (i=0;i<nH3D;i++) {
            //    2------3
            //   /|     /|
            //  6-|----7 |
            //  | |    | |
            //  | 0----|-1
            //  |/     |/
            //  4------5

            h3D[i].draw_edgeGL(6,7);
            h3D[i].draw_edgeGL(3,7);
            h3D[i].draw_edgeGL(5,7);
        }
    }

    glPopAttrib();

    if (NumON){

        FuncionesOpenGL::material(0);

        for (i=0;i<nV3D;i++) {
            glPushMatrix();

            glTranslatef(v3D[i].x,v3D[i].y,v3D[i].z);
            glMultMatrixf((GLfloat *)MatrizRotacionGlobalINV);
            glScalef(NumEscala,NumEscala,NumEscala);
            glTranslatef(0,0,.5);
            sprintf(s,"%d",i);
            FuncionesOpenGL::Print(s);
            glPopMatrix();
        }
    }
```

```cpp
}


void grid3D::cubo(int ix,int iy,int iz,float Lx,float Ly,float Lz)
{
    int i,j,k;
    nH3D=ix*iy*iz;
    nV3D=(ix+1)*(iy+1)*(iz+1);
    v3D.resize(nV3D,Vertex3D());
    h3D.resize(nH3D,Hexa3D());
    for (i=0;i<=ix;i++) {
        for (j=0;j<=iy;j++) {
            for (k=0;k<=iz;k++) {
                if ((i<ix)&&(j<iy)&&(k<iz)) {

                    h3D[i*(iy*iz)+j*(iz)+k].papa=this;
                    h3D[i*(iy*iz)+j*(iz)+k].no=i*(iy*iz)+j*(iz)+k;
                    h3D[i*(iy*iz)+j*(iz)+k].iv[0]=i*(iy+1)*(iz+1)+  j  *(iz+1)+  k;
                    h3D[i*(iy*iz)+j*(iz)+k].iv[1]=i*(iy+1)*(iz+1)+(j+1)*(iz+1)+  k;
                    h3D[i*(iy*iz)+j*(iz)+k].iv[2]=i*(iy+1)*(iz+1)+  j  *(iz+1)+(k+1);
                    h3D[i*(iy*iz)+j*(iz)+k].iv[3]=i*(iy+1)*(iz+1)+(j+1)*(iz+1)+(k+1);
                    h3D[i*(iy*iz)+j*(iz)+k].iv[4]=(i+1)*(iy+1)*(iz+1)+  j  *(iz+1)+  k;
                    h3D[i*(iy*iz)+j*(iz)+k].iv[5]=(i+1)*(iy+1)*(iz+1)+(j+1)*(iz+1)+  k;
                    h3D[i*(iy*iz)+j*(iz)+k].iv[6]=(i+1)*(iy+1)*(iz+1)+  j  *(iz+1)+(k+1);
                    h3D[i*(iy*iz)+j*(iz)+k].iv[7]=(i+1)*(iy+1)*(iz+1)+(j+1)*(iz+1)+(k+1);
                }
                v3D[i*(iy+1)*(iz+1)+  j  *(iz+1)+  k].no=i*(iy+1)*(iz+1)+  j  *(iz+1)+  k;
                v3D[i*(iy+1)*(iz+1)+  j  *(iz+1)+  k].papa=this;
                v3D[i*(iy+1)*(iz+1)+  j  *(iz+1)+  k].x=Lx/ix*i;
                v3D[i*(iy+1)*(iz+1)+  j  *(iz+1)+  k].y=Ly/iy*j;
                v3D[i*(iy+1)*(iz+1)+  j  *(iz+1)+  k].z=Lz/iz*k;
            }
        }
    }
}

void grid3D::Junta(grid3D g1,grid3D g2)
{
    int i,j;
    double DIJ;
    vector<int> i2enUnion(g2.nV3D);
    const double eps=1e-10;
    v3D=g1.v3D;
    nV3D=g1.nV3D;
    h3D=g1.h3D;
    nH3D=g1.nH3D+g2.nH3D;
    for (i=0;i<g2.nV3D;i++) {
        for (j=0;j<g1.nV3D;j++) {
            DIJ=sqr(g1.v3D[j].x-g2.v3D[i].x)+sqr(g1.v3D[j].y-g2.v3D[i].y)+sqr(g1.v3D[j].z-g2.v3D[i].z);
            if (DIJ<eps) break;
        }
        if (j<g1.nV3D){
            i2enUnion[i]=j;
        } else {
            i2enUnion[i]=nV3D;
            v3D.push_back(g2.v3D[i]);
            nV3D++;
        }
    }
    for (i=0;i<g2.nH3D;i++) {
        h3D.push_back(g2.h3D[i]);
        for (j=0;j<8;j++) {
            h3D[g1.nH3D+i].iv[j]=i2enUnion[g2.h3D[i].iv[j]];
        }
    }
    for (i=0;i<nV3D;i++) {
        v3D[i].no=i;
        v3D[i].papa=this;
    }
    for (i=0;i<nH3D;i++) {
        h3D[i].no=i;
        h3D[i].papa=this;
    }


}

void grid3D::Junta(grid3D g2)
{
    int i,j,nv1,nH1;
    double DIJ;
    vector<int> i2enUnion(g2.nV3D);
    const double eps=1e-10;
    nv1=nV3D;
    nH1=nH3D;
    nH3D+=g2.nH3D;
    for (i=0;i<g2.nV3D;i++) {
        for (j=0;j<nv1;j++) {
            DIJ=sqr(v3D[j].x-g2.v3D[i].x)+sqr(v3D[j].y-g2.v3D[i].y)+sqr(v3D[j].z-g2.v3D[i].z);
            if (DIJ<eps) break;
        }
        if (j<nv1){
            i2enUnion[i]=j;
        } else {
            i2enUnion[i]=nV3D;
            v3D.push_back(g2.v3D[i]);
            nV3D++;
        }
    }
    for (i=0;i<g2.nH3D;i++) {
        h3D.push_back(g2.h3D[i]);
        for (j=0;j<8;j++) {
            h3D[nH1+i].iv[j]=i2enUnion[g2.h3D[i].iv[j]];
        }
    }
    for (i=0;i<nV3D;i++) {
        v3D[i].no=i;
        v3D[i].papa=this;
    }
    for (i=0;i<nH3D;i++) {
        h3D[i].no=i;
        h3D[i].papa=this;
    }


}

void grid3D::Rota90Z()
{
    int i;
    double xL,yL;
    for (i=0;i<nV3D ;i++){
        xL=v3D[i].x;
        yL=v3D[i].y;
        v3D[i].x = -yL;
        v3D[i].y = xL;
    }
}

void grid3D::Traslada(double dx,double dy,double dz)
{
    int i;
    for (i=0;i<nV3D ;i++){
        v3D[i].x += dx;
        v3D[i].y += dy;
        v3D[i].z += dz;
    }
}


```