

```

1 #pragma once
2 #include <stdio.h>
3 #include "grid3D.h"
4 #include "FuncionesOpenGL.h"
5 #include "Class_Vector.h"
6
7
8
9 double xx0,yy0,zz0,xx1,yy1,zz1,xx2,yy2,zz2;
10 int xxi;
11 int Dibuja3puntos=false;
12 int QuienGeneraPoligonos;
13
14 #define sqr(x) ((x)*(x))
15
16
17 extern void ZGlobal(double*v);
18 extern int NumON;
19 extern float NumEscala;
20 extern int Modo_DibujaEdges,ModoDibujaNormales;
21 extern int ModoDibujaInterior;
22 extern int ModoDibujaFrontera;
23 extern int Modo_DibujaCentroCaras;
24 extern int Modo_DibujaCentroBloques;
25 extern double Dominio_Rint,Dominio_Rmax,Dominio_Hsup;
26
27
28
29 double ppunto(R3 a,R3 b) {
30     return(a.x*b.x+a.y*b.y+a.z*b.z);
31 }
32
33 double ppuntodiff(R3 N,R3 b,R3 c) {
34     return(N.x*(b.x-c.x)+N.y*(b.y-c.y)+N.z*(b.z-c.z));
35 }
36
37 grid3D::grid3D(void)
38 {
39 }
40
41 grid3D::~grid3D(void)
42 {
43 }
44
45
46 void grid3D::draw_caraGL(int ii[4])
47 {
48
49     int li;
50     double x[4],y[4],z[4],v[3],xg,yg,zg,lambda=0.0,nx,ny,nz,nxx,nyy,nzz,nnn;
51     GLdouble winX1,winY1,winX2,winY2,winX3,winY3,winZ,winZ2;
52     xg=0;yg=0;zg=0;
53
54     for (li=0;li<4;li++) {
55         x[li]=v3D[ii[li]].x;
56         y[li]=v3D[ii[li]].y;
57         z[li]=v3D[ii[li]].z;
58         xg+=x[li]/4;         yg+=y[li]/4;         zg+=z[li]/4;
59     }
60     for (li=0;li<4;li++) {
61         x[li]+=(xg-x[li])*lambda;
62         y[li]+=(yg-y[li])*lambda;
63         z[li]+=(zg-z[li])*lambda;
64     }
65
66     nxx=nx=(y[2]-y[0])*(z[3]-z[1])-(z[2]-z[0])*(y[3]-y[1]);
67     nyy=ny=(z[2]-z[0])*(x[3]-x[1])-(x[2]-x[0])*(z[3]-z[1]);
68     nzz=nz=(x[2]-x[0])*(y[3]-y[1])-(y[2]-y[0])*(x[3]-x[1]);
69     nnn=sqrt(sqr(nxx)+sqr(nyy)+sqr(nzz));
70
71     // ZGlobal(v);
72     FuncionesOpenGL::modelview_calculado=false;
73     /*
74     FuncionesOpenGL::World2Win(xg,yg,zg,&winX,&winY,&winZ);
75     FuncionesOpenGL::World2Win(xg+nxx,yg+nyy,zg+nzz,&winX,&winY,&winZ2);
76     if (winZ2>winZ){
77         nxx=-nxx;nyy=-nyy;nzz=-nzz;
78     }
79     */
80     FuncionesOpenGL::World2Win(x[0],y[0],z[0],&winX1,&winY1,&winZ);
81     FuncionesOpenGL::World2Win(x[1],y[1],z[1],&winX2,&winY2,&winZ);
82     FuncionesOpenGL::World2Win(x[2],y[2],z[2],&winX3,&winY3,&winZ);
83     if ((winX2-winX1)*(winY3-winY1)-(winY2-winY1)*(winX3-winX1) <0 ){
84         nxx=-nxx;nyy=-nyy;nzz=-nzz;
85     }
86
87     /*
88     if (nxx*v[0]+nyy*v[1]+nzz*v[2]<0){
89         nxx=-nxx;nyy=-nyy;nzz=-nzz;
90     }
91     */
92
93     glEnable(GL_NORMALIZE);
94     glBegin(GL_QUADS);
95     for (li=0;li<4;li++) {
96         if (x[li]<xg)
97             nx=nnx+(x[li]-xg)*((x[li]-xg))/0.51;
98         else

```

```

98         nx=nxx-(x[li]-xg)*((x[li]-xg))/0.51;
99         if (y[li]<yg)
100             ny=nyy+(y[li]-yg)*((y[li]-yg))/0.51;
101         else
102             ny=nyy-(y[li]-yg)*((y[li]-yg))/0.51;
103         if (z[li]<zg)
104             nz=nzz+(z[li]-zg)*((z[li]-zg))/0.51;
105         else
106             nz=nzz-(z[li]-zg)*((z[li]-zg))/0.51;
107
108
109         //glNormal3d(nxx,nyy,nzz);
110         glNormal3d(nx,ny,nz);
111         glVertex3d(x[li], y[li],z[li]);
112     }
113     glEnd();
114
115     if(ModoDibujaNormales) {
116         glPushAttrib( GL_LIGHTING_BIT );
117         glDisable( GL_LIGHTING );
118
119         glBegin(GL_LINES);
120         glColor3f(1,0,0);
121         glVertex3d(xg,yg,zg);
122         glVertex3d(xg+nxx/nnn/10,yg+nyy/nnn/10,zg+nzz/nnn/10);
123         glEnd();
124         glPopAttrib();
125     }
126 }
127
128
129
130 void grid3D::draw_caraGL(vector<double>F,double minF,double maxF,int ii[4])
131 {
132
133     int li;
134     double x[4],y[4],z[4],v[3],xg,yg,zg,lambda=0.0,nx,ny,nz,nxx,nyy,nzz,nnn;
135     GLdouble winX1,winY1,winX2,winY2,winX3,winY3,winZ,winZ2;
136     xg=0;yg=0;zg=0;
137
138     for (li=0;li<4;li++) {
139         x[li]=v3D[ii[li]].x;
140         y[li]=v3D[ii[li]].y;
141         z[li]=v3D[ii[li]].z;
142         xg+=x[li]/4;         yg+=y[li]/4;         zg+=z[li]/4;
143     }
144     for (li=0;li<4;li++) {
145         x[li]+=(xg-x[li])*lambda;
146         y[li]+=(yg-y[li])*lambda;
147         z[li]+=(zg-z[li])*lambda;
148     }
149
150     nxx=nx=(y[2]-y[0])*(z[3]-z[1])-(z[2]-z[0])*(y[3]-y[1]);
151     nyy=ny=(z[2]-z[0])*(x[3]-x[1])-(x[2]-x[0])*(z[3]-z[1]);
152     nzz=nz=(x[2]-x[0])*(y[3]-y[1])-(y[2]-y[0])*(x[3]-x[1]);
153     nnn=sqrt(sqr(nxx)+sqr(nyy)+sqr(nzz));
154
155     ZGlobal(v);
156
157     glEnable(GL_NORMALIZE);
158     int s;
159     glBegin(GL_QUADS);
160     if ( nxx*v[0]+nyy*v[1]+nzz*v[2]<0) s=-1; // glNormal3d(-nx,-ny,-nz);
161     else s=1; //glNormal3d(nx,ny,nz);
162     for (li=0;li<4;li++) {
163         FuncionesOpenGL::ColorF(minF,maxF,F[ii[li]]);
164         if (x[li]<xg)
165             nx=nxx+(x[li]-xg)*((x[li]-xg))/10.51;
166         else
167             nx=nxx-(x[li]-xg)*((x[li]-xg))/10.51;
168         if (y[li]<yg)
169             ny=nyy+(y[li]-yg)*((y[li]-yg))/10.51;
170         else
171             ny=nyy-(y[li]-yg)*((y[li]-yg))/10.51;
172         if (z[li]<zg)
173             nz=nzz+(z[li]-zg)*((z[li]-zg))/10.51;
174         else
175             nz=nzz-(z[li]-zg)*((z[li]-zg))/10.51;
176
177         //glNormal3d(nxx,nyy,nzz);
178         glNormal3d(s*nx,s*ny,s*nz);
179         glVertex3d(x[li], y[li],z[li]);
180     }
181     glEnd();
182
183
184 }
185
186
187 void Hexa3D::draw_caraGL(int i0,int i1,int i2,int i3)
188 {
189     int li;
190     double x[4],y[4],z[4],v[3],xg,yg,zg,lambda=0.0,nx,ny,nz,nxx,nyy,nzz;
191     x[0]=papa->v3D[i0].x; x[1]=papa->v3D[i1].x; x[2]=papa->v3D[i2].x; x[3]=papa->v3D[i3].x;
192     y[0]=papa->v3D[i0].y; y[1]=papa->v3D[i1].y; y[2]=papa->v3D[i2].y; y[3]=papa->v3D[i3].y;
193     z[0]=papa->v3D[i0].z; z[1]=papa->v3D[i1].z; z[2]=papa->v3D[i2].z; z[3]=papa->v3D[i3].z;

```

```

195  xg=0;yg=0;zg=0;
196  for (li=0;li<4;li++) {
197      xg+=x[li]/4;          yg+=y[li]/4;          zg+=z[li]/4;
198  }
199  for (li=0;li<4;li++) {
200      x[li]+=(xg-x[li])*lambda;
201      y[li]+=(yg-y[li])*lambda;
202      z[li]+=(zg-z[li])*lambda;
203  }
204
205  nxx=nx=(y[2]-y[0])*(z[3]-z[1])-(z[2]-z[0])*(y[3]-y[1]);
206  nyy=ny=(z[2]-z[0])*(x[3]-x[1])-(x[2]-x[0])*(z[3]-z[1]);
207  nzz=nz=(x[2]-x[0])*(y[3]-y[1])-(y[2]-y[0])*(x[3]-x[1]);
208
209  ZGlobal(v);
210  glEnable(GL_NORMALIZE);
211  glBegin(GL_QUADS);
212  for (li=0;li<4;li++) {
213      if (x[li]<xg)
214          nx=-nxx+(x[li]-xg)*((x[li]-xg))/0.51;
215      else
216          nx=-nxx-(x[li]-xg)*((x[li]-xg))/0.51;
217      if (y[li]<yg)
218          ny=-nyy+(y[li]-yg)*((y[li]-yg))/0.51;
219      else
220          ny=-nyy-(y[li]-yg)*((y[li]-yg))/0.51;
221      if (z[li]<zg)
222          nz=-nzz+(z[li]-zg)*((z[li]-zg))/0.51;
223      else
224          nz=-nzz-(z[li]-zg)*((z[li]-zg))/0.51;
225
226
227      if (nxx*v[0]+nyy*v[1]+nzz*v[2]<0)
228          glNormal3d(nx,ny,nz);
229      else
230          glNormal3d(-nx,-ny,-nz);
231
232
233      //glNormal3d(nx,ny,nz);
234      glVertex3d(x[li], y[li],z[li]);
235  }
236  glEnd();
237
238 }
239
240 void Hexa3D::draw_caraGL(vector<double>F,double minF,double maxF, int i0,int i1,int i2,int i3)
241 {
242     int li;
243     double x[4],y[4],z[4],v[3],nx,ny,nz,nxx,nyy,nzz;
244     double lF[4];
245     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;
246     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;
247     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;
248     lF[0]=F[iv[i0]];            lF[1]=F[iv[i1]];            lF[2]=F[iv[i2]];            lF[3]=F[iv[i3]];
249
250     nxx=nx=(y[2]-y[0])*(z[3]-z[1])-(z[2]-z[0])*(y[3]-y[1]);
251     nyy=ny=(z[2]-z[0])*(x[3]-x[1])-(x[2]-x[0])*(z[3]-z[1]);
252     nzz=nz=(x[2]-x[0])*(y[3]-y[1])-(y[2]-y[0])*(x[3]-x[1]);
253
254     ZGlobal(v);
255     glEnable(GL_NORMALIZE);
256     {
257         glBegin(GL_TRIANGLES );
258         if ( nxx*v[0]+nyy*v[1]+nzz*v[2]<0)    glNormal3d(-nx,-ny,-nz);
259         else                                glNormal3d(nx,ny,nz);
260
261         FuncionesOpenGL::CoLorF(minF,maxF,lF[0]);glVertex3d(x[0], y[0],z[0]);
262         // cout<<"Hexa3D::draw_caraGL minF,MaxF,lF[0]="<<minF<<" "<<maxF<<" "<<lF[0]<<endl;
263         FuncionesOpenGL::CoLorF(minF,maxF,lF[1]);glVertex3d(x[1], y[1],z[1]);
264         FuncionesOpenGL::CoLorF(minF,maxF,lF[2]);glVertex3d(x[2], y[2],z[2]);
265         glEnd();
266     }
267
268     {
269         glBegin(GL_TRIANGLES );
270         if ( nxx*v[0]+nyy*v[1]+nzz*v[2]<0)    glNormal3d(-nx,-ny,-nz);
271         else                                glNormal3d(nx,ny,nz);
272
273         FuncionesOpenGL::CoLorF(minF,maxF,lF[0]);glVertex3d(x[0], y[0],z[0]);
274         FuncionesOpenGL::CoLorF(minF,maxF,lF[2]);glVertex3d(x[2], y[2],z[2]);
275         FuncionesOpenGL::CoLorF(minF,maxF,lF[3]);glVertex3d(x[3], y[3],z[3]);
276         glEnd();
277     }
278
279 }
280
281 void Hexa3D::draw_edgeGL(int i0,int i1)
282 {
283     int li;
284     vector<Vertex3D> *lv3D;
285     lv3D=&(papa->v3D);
286
287     glBegin(GL_LINES);
288     glVertex3d((papa->v3D)[iv[i0]].x, (papa->v3D)[iv[i0]].y, (papa->v3D)[iv[i0]].z);
289     glVertex3d((papa->v3D)[iv[i1]].x, (papa->v3D)[iv[i1]].y, (papa->v3D)[iv[i1]].z);
290     glEnd();
291 }

```

```

292 }
293
294 int grid3D::AddCara(int ib,int i0,int i1,int i2,int i3)
295 {
296     int i;
297     int is,is2,is3,ip,iip,iis,iis2,iis3;
298     is=i0+i1+i2+i3;
299     is2=sqr(i0)+sqr(i1)+sqr(i2)+sqr(i3);
300     ip=(i0+1)*(i1+1)*(i2+1)*(i3+1);
301     is3=ip/(i0+1)+ip/(i1+1)+ip/(i2+1)+ip/(i3+1);
302     for (i=0;i<nCaras;i++) {
303         iis=Cara[i].iv[0]+Cara[i].iv[1]+Cara[i].iv[2]+Cara[i].iv[3];
304         if (iis != is) continue;
305         iis2=sqr(Cara[i].iv[0])+sqr(Cara[i].iv[1])+sqr(Cara[i].iv[2])+sqr(Cara[i].iv[3]);
306         if (iis2 != is2) continue;
307         iip=(Cara[i].iv[0]+1)*(Cara[i].iv[1]+1)*(Cara[i].iv[2]+1)*(Cara[i].iv[3]+1);
308         if (iip != ip) continue;
309         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);
310         if (iis3 != is3) continue;
311         break;
312     }
313     if (i<nCaras) { //Cara existente
314         Cara[i].ih[Cara[i].nh]=ib;
315         Cara[i].nh++;
316     }
317     return(i);
318 }
319 Cara.push_back(Cara3D());
320 Cara[nCaras].nv=4;
321 Cara[nCaras].nh=1;
322 Cara[nCaras].iv[0]=i0;
323 Cara[nCaras].iv[1]=i1;
324 Cara[nCaras].iv[2]=i2;
325 Cara[nCaras].iv[3]=i3;
326 Cara[nCaras].ih[0]=ib;
327 nCaras++;
328 return (nCaras-1);
329
330
331 }
332 void GeneraPoligonoInicial(R3 a, R3 b, PoligonoPlano &P)
333 {
334     double d1x,d1y,d1z,dab;
335     double d2x,d2y,d2z,dd;
336     double d3x,d3y,d3z;
337     double lambda=1.5;
338     d1x=(b.x-a.x); d1y=(b.y-a.y); d1z=(b.z-a.z);
339     dab=sqrt(sqr(d1x)+sqr(d1y)+sqr(d1z));
340     if (fabs(d1x)>dab/4) {
341         d2x=-d1y; d2y=d1x; d2z=0;
342     } else if (fabs(d1y)>dab/4) {
343         d2x=-d1y; d2y=d1x; d2z=0;
344     } if (fabs(d1z)>dab/4) {
345         d2x=-d1z; d2y=0; d2z=d1x;
346     }
347     d3x=(d1y*d2z-d1z*d2y); d3y=(d1z*d2x-d1x*d2z); d3z=(d1x*d2y-d1y*d2x);
348
349     dd=sqrt(sqr(d2x)+sqr(d2y)+sqr(d2z)); d2x /=dd; d2y /=dd; d2z /=dd;
350     dd=sqrt(sqr(d3x)+sqr(d3y)+sqr(d3z)); d3x /=dd; d3y /=dd; d3z /=dd;
351
352     P.Dab=dab;
353     P.normal.x=d1x/dab;
354     P.normal.y=d1y/dab;
355     P.normal.z=d1z/dab;
356     P.centro.x=(a.x+b.x)/2;
357     P.centro.y=(a.y+b.y)/2;
358     P.centro.z=(a.z+b.z)/2;
359     P.punto.clear();
360     P.punto.resize(4);
361     P.punto[0].x = (a.x+b.x)/2 + lambda*d2x + lambda*d3x;
362     P.punto[0].y = (a.y+b.y)/2 + lambda*d2y + lambda*d3y;
363     P.punto[0].z = (a.z+b.z)/2 + lambda*d2z + lambda*d3z;
364
365     P.punto[1].x = (a.x+b.x)/2 - lambda*d2x + lambda*d3x;
366     P.punto[1].y = (a.y+b.y)/2 - lambda*d2y + lambda*d3y;
367     P.punto[1].z = (a.z+b.z)/2 - lambda*d2z + lambda*d3z;
368
369     P.punto[2].x = (a.x+b.x)/2 - lambda*d2x - lambda*d3x;
370     P.punto[2].y = (a.y+b.y)/2 - lambda*d2y - lambda*d3y;
371     P.punto[2].z = (a.z+b.z)/2 - lambda*d2z - lambda*d3z;
372
373     P.punto[3].x = (a.x+b.x)/2 + lambda*d2x - lambda*d3x;
374     P.punto[3].y = (a.y+b.y)/2 + lambda*d2y - lambda*d3y;
375     P.punto[3].z = (a.z+b.z)/2 + lambda*d2z - lambda*d3z;
376
377 }
378
379 void RecortaPoligono(R3 a, R3 b, PoligonoPlano &P)
380 {
381     double a2,b2,KP,KQ,xpn,ypn,zpn,xnp,ynp,znp;
382     int icj,ibB,k,kk,km1,kneg_in1,kneg_fin,nerase;
383     a2=sqr(a.x)+sqr(a.y)+sqr(a.z);
384     b2=sqr(b.x)+sqr(b.y)+sqr(b.z);
385     xx2=b.x;
386     yy2=b.y;
387     zz2=b.z;
388     //busco k tal que KQ(k)>0 (vertice dentro de la zona)

```

```

389 for(k=0 ; k<P.punto.size() ; k++) {
390     KQ= 2*(a.x-b.x)*P.punto[k].x+
391         2*(a.y-b.y)*P.punto[k].y+
392         2*(a.z-b.z)*P.punto[k].z + b2-a2;
393     if (KQ>0) break;
394 }
395 if (k>=P.punto.size()) {
396     P.punto.clear();
397     return; //Poligono fuera de la zona de interes
398 }
399 //hay un punto con KP>0....(OK)
400 for(kk=0 ; kk < P.punto.size() ; kk++) {
401     km1=k;
402     k++;if ( k>=P.punto.size() ) k=0;
403     KP=KQ;
404     KQ= 2*(a.x-b.x)*P.punto[k].x+
405         2*(a.y-b.y)*P.punto[k].y+
406         2*(a.z-b.z)*P.punto[k].z + b2-a2;
407     if (KQ<0) break;
408 }
409 if (kk>=P.punto.size()) return; //Todoel poligono es positivo
410 //Encontre KQ(k) negativo
411 kneg_ini=k;
412 xpn=( KP*P.punto[k].x - KQ*P.punto[km1].x ) / (KP-KQ);
413 ypn=( KP*P.punto[k].y - KQ*P.punto[km1].y ) / (KP-KQ);
414 zpn=( KP*P.punto[k].z - KQ*P.punto[km1].z ) / (KP-KQ);
415 nerase=0;
416 for(kk=0 ; kk < P.punto.size() ; kk++) {
417     km1=k;
418     k++;if ( k>=P.punto.size() ) k=0;
419     KP=KQ;
420     KQ= 2*(a.x-b.x)*P.punto[k].x+
421         2*(a.y-b.y)*P.punto[k].y+
422         2*(a.z-b.z)*P.punto[k].z + b2-a2;
423     if (kk>1) nerase++;
424     if (KQ>0) break;
425 }
426 kneg_fin=km1;
427 xnp=( KP*P.punto[k].x - KQ*P.punto[km1].x ) / (KP-KQ);
428 ynp=( KP*P.punto[k].y - KQ*P.punto[km1].y ) / (KP-KQ);
429 znp=( KP*P.punto[k].z - KQ*P.punto[km1].z ) / (KP-KQ);
430 if (kneg_fin==kneg_ini) {
431     P.punto.insert( P.punto.begin()+kneg_ini , R3() );
432     P.punto[kneg_ini ].x=xpn;
433     P.punto[kneg_ini ].y=ypn;
434     P.punto[kneg_ini ].z=zpn;
435     P.punto[kneg_ini+1].x=xnp;
436     P.punto[kneg_ini+1].y=ypn;
437     P.punto[kneg_ini+1].z=znp;
438 } else if (nerase==0) {
439     P.punto[kneg_ini].x=xpn;
440     P.punto[kneg_ini].y=ypn;
441     P.punto[kneg_ini].z=zpn;
442     P.punto[kneg_fin].x=xnp;
443     P.punto[kneg_fin].y=ypn;
444     P.punto[kneg_fin].z=znp;
445 } else {
446     P.punto[kneg_ini].x=xpn;
447     P.punto[kneg_ini].y=ypn;
448     P.punto[kneg_ini].z=zpn;
449     P.punto[kneg_fin].x=xnp;
450     P.punto[kneg_fin].y=ypn;
451     P.punto[kneg_fin].z=znp;
452     if (kneg_ini+nerase < P.punto.size() ) {
453         P.punto.erase( P.punto.begin()+kneg_ini+1 , P.punto.begin()+kneg_ini+nerase+1 );
454     } else {
455         int n1=P.punto.size()-kneg_ini-1;
456         if (n1>0) P.punto.erase( P.punto.begin()+kneg_ini+1 , P.punto.begin()+kneg_ini+n1+1 );
457         int n2=nerase-n1;
458         P.punto.erase( P.punto.begin() , P.punto.begin()+n2 );
459     }
460 }
461 }
462
463 void grid3D::CentroCarasBloques()
464 {
465     int i,j,k,l,iCuantosPoligonos,icj,ibb,jj;
466     double xg,yg,zg,x0,y0,z0,x1,y1,z1,d1x,d1y,d1z,d2x,d2y,d2z,d3x,d3y,d3z,dij,lambada;
467     double Ax,Ay,Az,AA;
468     for(i=0;i<nH3D;i++) {
469         xg=0;yg=0;zg=0;
470         for(j=0;j<8;j++) {
471             xg+=v3D[h3D[i].iv[j]].x; yg+=v3D[h3D[i].iv[j]].y; zg+=v3D[h3D[i].iv[j]].z;
472         }
473         h3D[i].centro.x=xg/8; h3D[i].centro.y=yg/8; h3D[i].centro.z=zg/8;
474     }
475     for(i=0;i<nCaras;i++) {
476         xg=0;yg=0;zg=0;
477         Cara[i].nVertexPolig=0;
478         for(j=0;j<4;j++) {
479             xg+=v3D[Cara[i].iv[j]].x; yg+=v3D[Cara[i].iv[j]].y; zg+=v3D[Cara[i].iv[j]].z;
480         }
481         Cara[i].centro.x=xg/4; Cara[i].centro.y=yg/4; Cara[i].centro.z=zg/4;
482         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)
483             - (v3D[Cara[i].iv[1]].z - v3D[Cara[i].iv[0]].z) * (v3D[Cara[i].iv[2]].y - v3D[Cara[i].iv[0]].y);
484         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)
485             - (v3D[Cara[i].iv[1]].x - v3D[Cara[i].iv[0]].x) * (v3D[Cara[i].iv[2]].z - v3D[Cara[i].iv[0]].z);

```

```

486     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)
487     - (v3D[Cara[i].iv[1]].y - v3D[Cara[i].iv[0]].y) * (v3D[Cara[i].iv[2]].x - v3D[Cara[i].iv[0]].x);
488     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)
489     - (v3D[Cara[i].iv[2]].z - v3D[Cara[i].iv[0]].z) * (v3D[Cara[i].iv[3]].y - v3D[Cara[i].iv[0]].y);
490     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)
491     - (v3D[Cara[i].iv[2]].x - v3D[Cara[i].iv[0]].x) * (v3D[Cara[i].iv[3]].z - v3D[Cara[i].iv[0]].z);
492     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)
493     - (v3D[Cara[i].iv[2]].y - v3D[Cara[i].iv[0]].y) * (v3D[Cara[i].iv[3]].x - v3D[Cara[i].iv[0]].x);
494
495     double Norma2=sqrt(sqr(Ax)+sqr(Ay)+sqr(Az));
496     Cara[i].Area = Norma2/2;
497     Cara[i].normal.x = Ax/Norma2;
498     Cara[i].normal.y = Ay/Norma2;
499     Cara[i].normal.z = Az/Norma2;
500     Cara[i].normal.L = 1.0;
501 }
502 #ifdef JSM_NO_HACER_ESTO
503 for(i=0;i<nH3D;i++) {
504     xg=0;yg=0;zg=0;AA=0;
505     for(j=0;j<6;j++) {
506         int ic=h3D[i].icara[j];
507         xg+=Cara[ic].centro.x * Cara[ic].Area ;
508         yg+=Cara[ic].centro.y * Cara[ic].Area;
509         zg+=Cara[ic].centro.z * Cara[ic].Area;
510         AA+= Cara[ic].Area;
511     }
512     // h3D[i].centro.x=xg/AA; h3D[i].centro.y=yg/AA; h3D[i].centro.z=zg/AA;
513 }
514 #endif
515 }
516
517 void grid3D::generaPoligonos2(int CuantosPoligonos)
518 {
519     int i,j,k,l,iCuantosPoligonos,icj,ibb,jj;
520     double xg,yg,zg,x0,y0,z0,x1,y1,z1,d1x,d1y,d1z,d2x,d2y,d2z,d3x,d3y,d3z,dij,lambdas;
521     double Ax,Ay,Az,AA;
522     QuienGeneraPoligonos=2;
523
524     //Primer nivel de vecinos
525     for(i=0;i<nH3D;i++) {
526         h3D[i].vecino.resize(6);
527         h3D[i].tipo_vecino.resize(6);
528         for (j=0;j<6;j++) {
529             icj=h3D[i].icara[j];
530             if (Cara[icj].nh>1) {
531                 h3D[i].tipo_vecino[j]=ES_BLOQUE;
532                 ibb=Cara[icj].ih[0]; if (ibb==i) ibb=Cara[icj].ih[1];
533                 h3D[i].vecino[j]=ibb;
534             } else {
535                 h3D[i].tipo_vecino[j]=ES_CARA;;
536                 h3D[i].vecino[j]=icj;
537             }
538         }
539     }
540     //segundo nivel de vecinos (los vecinos de mis BLOQUES vecinos)
541     for(i=0;i<nH3D;i++) {
542         int nvecinos_level1=h3D[i].vecino.size();
543         for (j=0 ; j<nvecinos_level1 ;j++) {
544             if (h3D[i].tipo_vecino[j]==ES_CARA) continue;
545             ibb=h3D[i].vecino[j];
546             for (k=0 ; k<h3D[ibb].vecino.size() ;k++) {
547                 if (h3D[ibb].tipo_vecino[k]==ES_CARA) continue;
548                 if (h3D[ibb].vecino[k]==i) continue;
549                 for (l=0 ; l<h3D[i].vecino.size() ;l++) {
550                     if (h3D[i].tipo_vecino[l]==ES_CARA) continue;
551                     if (h3D[i].vecino[l]==h3D[ibb].vecino[k]) break;
552                 }
553                 if (l==h3D[i].vecino.size()) { //es un nuevo vecino
554                     h3D[i].vecino.push_back(0);
555                     h3D[i].tipo_vecino.push_back(0);
556                     h3D[i].vecino[l]=h3D[ibb].vecino[k];
557                     h3D[i].tipo_vecino[l]=h3D[ibb].tipo_vecino[k];
558                 }
559             }
560         }
561     }
562     for(i=0;i<nH3D;i++) {
563         R3 VA,VB,NB;
564         VA=h3D[i].centro;
565         for (j=0 ; j< h3D[i].vecino.size() ; j++) {
566             if (h3D[i].tipo_vecino[j]==ES_CARA) {
567                 VB=Cara[ h3D[i].vecino[j] ].centro;
568                 NB=Cara[ h3D[i].vecino[j] ].normal;
569                 if (Cara[ h3D[i].vecino[j] ].iBC ==1||Cara[ h3D[i].vecino[j] ].iBC >=10) {
570                     double lambda=2*ppuntodiff(NB,VB,VA);
571                     Cara[ h3D[i].vecino[j] ].centro.x = VA.x+lambda*NB.x;
572                     Cara[ h3D[i].vecino[j] ].centro.y = VA.y+lambda*NB.y;
573                     Cara[ h3D[i].vecino[j] ].centro.z = VA.z+lambda*NB.z;
574                     VB=Cara[ h3D[i].vecino[j] ].centro;
575                 }
576             }
577         }
578     }
579
580     int navance=0;
581     for(i=0;i<nH3D;i++) {
582         if (i>navance*nH3D/20) {

```

```

583         printf("%.4d",navance);fflush(stdout);
584         navance++;
585     }
586     h3D[i].Poligono.resize( h3D[i].vecino.size() );
587     R3 VA,VB;
588     VA=h3D[i].centro;
589     for (j=0; j< h3D[i].vecino.size(); j++) {
590         if (h3D[i].tipo_vecino[j]==ES_BLOQUE) {
591             VB=h3D[ h3D[i].vecino[j] ].centro;
592         }
593         if (h3D[i].tipo_vecino[j]==ES_CARA) {
594             VB=Cara[ h3D[i].vecino[j] ].centro;
595         }
596         GeneraPoligonoInicial(VA,VB, h3D[i].Poligono[j]);
597         for (jj=0; jj< h3D[i].vecino.size(); jj++) {
598             if (jj==j) continue;
599             if (h3D[i].tipo_vecino[jj]==ES_BLOQUE) {
600                 VB=h3D[ h3D[i].vecino[jj] ].centro;
601             }
602             if (h3D[i].tipo_vecino[jj]==ES_CARA) {
603                 VB=Cara[ h3D[i].vecino[jj] ].centro;
604             }
605             RecortaPoligono(VA,VB, h3D[i].Poligono[j]);
606             if (h3D[i].Poligono[j].punto.size() == 0) {
607                 h3D[i].Poligono.erase(h3D[i].Poligono.begin()+j);
608                 h3D[i].vecino.erase(h3D[i].vecino.begin()+j);
609                 h3D[i].tipo_vecino.erase(h3D[i].tipo_vecino.begin()+j);
610                 j--;
611                 break;
612             }
613         }
614     }
615 }
616 printf("\n");
617 //calculos de areas y longitudes de trazos
618 for (i=0;i<nH3D;i++) {
619     for (k=0; k<h3D[i].Poligono.size(); k++) {
620         vector<R3> *Pto;
621         Pto= &(h3D[i].Poligono[k].punto);
622         double Area,Ax,Ay,Az;
623         Area=0;
624         for (j=1; j<Pto->size(); j++) {
625             (*Pto)[j].L = sqrt( sqrt( (*Pto)[j].x- (*Pto)[j-1].x )
626                                 + sqrt( (*Pto)[j].y- (*Pto)[j-1].y )
627                                 + sqrt( (*Pto)[j].z- (*Pto)[j-1].z ) );
628             if ((*Pto)[j].L <1e-5) {
629                 (*Pto).erase( (*Pto).begin() + j);
630                 j--;
631                 continue;
632             }
633             if (j<2) continue;
634             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);
635             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);
636             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);
637             Area+=sqrt(sqr(Ax)+sqr(Ay)+sqr(Az))/2;
638         }
639         h3D[i].Poligono[k].Area=Area;
640         if (Area <1e-10) {
641             h3D[i].Poligono.erase(h3D[i].Poligono.begin()+k);
642             h3D[i].vecino.erase(h3D[i].vecino.begin()+k);
643             h3D[i].tipo_vecino.erase(h3D[i].tipo_vecino.begin()+k);
644             k--;
645         }
646     }
647 }
648 }
649
650 void grid3D::generaPoligonos(int CuantosPoligonos)
651 {
652     int i,j,iCuantosPoligonos;
653     double xg,yg,zg,x0,y0,z0,x1,y1,z1,d1x,d1y,d1z,d2x,d2y,d2z,d3x,d3y,d3z,dij,lambda;
654     QuienGeneraPoligonos=1;
655     for(i=0;i<nH3D;i++) {
656         xg=0;yg=0;zg=0;
657         for(j=0;j<8;j++) {
658             xg+=v3D[h3D[i].iv[j]].x; yg+=v3D[h3D[i].iv[j]].y; zg+=v3D[h3D[i].iv[j]].z;
659         }
660         h3D[i].centro.x=xg/8; h3D[i].centro.y=yg/8; h3D[i].centro.z=zg/8;
661     }
662     for(i=0;i<nCaras;i++) {
663         xg=0;yg=0;zg=0;
664         Cara[i].nVertexPolig=0;
665         for(j=0;j<4;j++) {
666             xg+=v3D[Cara[i].iv[j]].x; yg+=v3D[Cara[i].iv[j]].y; zg+=v3D[Cara[i].iv[j]].z;
667         }
668         Cara[i].centro.x=xg/4; Cara[i].centro.y=yg/4; Cara[i].centro.z=zg/4;
669     }
670     iCuantosPoligonos=0;
671     for(i=0;i<nCaras;i++) {
672         //nn++;
673         x0=h3D[Cara[i].ih[0]].centro.x; y0=h3D[Cara[i].ih[0]].centro.y; z0=h3D[Cara[i].ih[0]].centro.z;
674         if (Cara[i].nh==2) {
675             x1=h3D[Cara[i].ih[1]].centro.x; y1=h3D[Cara[i].ih[1]].centro.y; z1=h3D[Cara[i].ih[1]].centro.z;
676         } else {
677             x1=Cara[i].centro.x; y1=Cara[i].centro.y; z1=Cara[i].centro.z;
678         }
679     }

```

```

680 //      Cara[i].centro.x=(x0+x1)/2; Cara[i].centro.y=(y0+y1)/2; Cara[i].centro.z=(z0+z1)/2;
681      d1x=(x1-x0); d1y=(y1-y0); d1z=(z1-z0);
682      dij=sqrt(sqr(d1x)+sqr(d1y)+sqr(d1z));
683      if (fabs(d1x)>dij/4) {
684          d2x=-d1y; d2y=d1x; d2z=0;
685      } else if (fabs(d1y)>dij/4) {
686          d2x=-d1y; d2y=d1x; d2z=0;
687      } if (fabs(d1z)>dij/4) {
688          d2x=-d1z; d2y=0; d2z=d1x;
689      }
690      d3x=(d1y*d2z-d1z*d2y); d3y=(d1z*d2x-d1x*d2z); d3z=(d1x*d2y-d1y*d2x);
691
692      Cara[i].Dij=dij;
693      Cara[i].nx=d1x/dij; Cara[i].ny=d1y/dij; Cara[i].nz=d1z/dij;
694
695      dij=sqrt(sqr(d2x)+sqr(d2y)+sqr(d2z)); d2x /=dij; d2y /=dij; d2z /=dij;
696      dij=sqrt(sqr(d3x)+sqr(d3y)+sqr(d3z)); d3x /=dij; d3y /=dij; d3z /=dij;
697      lambda=1.5;
698
699      Cara[i].vPolig.punto.clear();
700      Cara[i].vPolig.punto.resize(4);
701      Cara[i].vPolig.punto[0].x=x0+d1x/2+lambda*d2x+lambda*d3x;
702      Cara[i].vPolig.punto[0].y=y0+d1y/2+lambda*d2y+lambda*d3y;
703      Cara[i].vPolig.punto[0].z=z0+d1z/2+lambda*d2z+lambda*d3z;
704
705      Cara[i].vPolig.punto[1].x=x0+d1x/2-lambda*d2x+lambda*d3x;
706      Cara[i].vPolig.punto[1].y=y0+d1y/2-lambda*d2y+lambda*d3y;
707      Cara[i].vPolig.punto[1].z=z0+d1z/2-lambda*d2z+lambda*d3z;
708
709      Cara[i].vPolig.punto[2].x=x0+d1x/2-lambda*d2x-lambda*d3x;
710      Cara[i].vPolig.punto[2].y=y0+d1y/2-lambda*d2y-lambda*d3y;
711      Cara[i].vPolig.punto[2].z=z0+d1z/2-lambda*d2z-lambda*d3z;
712
713      Cara[i].vPolig.punto[3].x=x0+d1x/2+lambda*d2x-lambda*d3x;
714      Cara[i].vPolig.punto[3].y=y0+d1y/2+lambda*d2y-lambda*d3y;
715      Cara[i].vPolig.punto[3].z=z0+d1z/2+lambda*d2z-lambda*d3z;
716
717      Cara[i].nVertexPolig=4;
718
719      //      x0=h3D[Cara[i].ih[0]].centro.x; y0=h3D[Cara[i].ih[0]].centro.y; z0=h3D[Cara[i].ih[0]].centro.z;
720      //      x1=h3D[Cara[i].ih[1]].centro.x; y1=h3D[Cara[i].ih[1]].centro.y; z1=h3D[Cara[i].ih[1]].centro.z;
721
722      xx0=x0;yy0=y0;zz0=z0;
723      xx1=x1;yy1=y1;zz1=z1;
724      xx2=x1;yy2=y1;zz2=z1;
725      xx1=i;
726      Dibuja3puntos=2;
727      if(++iCuantosPoligonos>CuantosPoligonos && CuantosPoligonos>0) {
728          return;
729      }
730
731      for (int iv=0;iv<Cara[i].nh;iv++) {
732          int ibA=Cara[i].ih[iv];
733          int icj,ibB;
734          for (j=0;j<6;j++) {
735              icj=h3D[ibA].icara[j];
736              if (icj==i) continue;
737              if (Cara[icj].nh>1) {
738                  ibB=Cara[icj].ih[0]; if (ibB==ibA) ibB=Cara[icj].ih[1];
739                  RecortaPoligono(h3D[ibA].centro, h3D[ibB].centro, Cara[i].vPolig);
740                  Cara[i].nVertexPolig = Cara[i].vPolig.punto.size();
741              }
742              if (Cara[icj].nh==1) {
743                  RecortaPoligono(h3D[ibA].centro, Cara[icj].centro, Cara[i].vPolig);
744                  Cara[i].nVertexPolig = Cara[i].vPolig.punto.size();
745              }
746              if(++iCuantosPoligonos>CuantosPoligonos && CuantosPoligonos>0) {
747                  return;
748              }
749          }
750      }
751  }
752  }
753
754 }
755
756
757
758 void grid3D::GeneraCaras(int inicia)
759 {
760     int ib,i;
761     static int cuantas=0;
762     cout<<"grid3D::GeneraCaras: inicia="<<inicia<<endl;
763     if (inicia<0) {
764         cuantas=0;
765     }
766
767     if (cuantas==0) {
768         nCaras=0;
769     }
770     for (i=0;i<inicia;i++){
771         if (cuantas<nH3D) {
772             //      2-----3
773             //      /|      /|
774             //      6-|----7 |
775             //      | |      | |
776             //      | 0----|-1

```



```

777         //  | /      | /
778         // 4-----5
779
780         ib=cuantas; cuantas++;
781         if(1==0) cout<<"ib="<<ib<<"\tincia="<<inicia<<endl;
782         h3D[ib].icara[0]=AddCara(ib,h3D[ib].iv[0],h3D[ib].iv[2],h3D[ib].iv[3],h3D[ib].iv[1]);
783         h3D[ib].icara[1]=AddCara(ib,h3D[ib].iv[0],h3D[ib].iv[4],h3D[ib].iv[6],h3D[ib].iv[2]);
784         h3D[ib].icara[2]=AddCara(ib,h3D[ib].iv[0],h3D[ib].iv[1],h3D[ib].iv[5],h3D[ib].iv[4]);
785         h3D[ib].icara[3]=AddCara(ib,h3D[ib].iv[4],h3D[ib].iv[5],h3D[ib].iv[7],h3D[ib].iv[6]);
786         h3D[ib].icara[4]=AddCara(ib,h3D[ib].iv[1],h3D[ib].iv[3],h3D[ib].iv[7],h3D[ib].iv[5]);
787         h3D[ib].icara[5]=AddCara(ib,h3D[ib].iv[2],h3D[ib].iv[6],h3D[ib].iv[7],h3D[ib].iv[3]);
788     }
789 }
790 }
791
792
793 int nParticulas=200;
794 double ThetaMax,ThetaMin,dTheta_med;
795 static Vector<double> Particulas[maxpasadas+1][3];
796 int primerdrawVelGL=1;
797
798
799 int npasadas=1;
800
801 void PosINI(int i)
802 {
803     double x,y,z,t;
804     int j,k;
805
806     x=( (double)rand() / ((double)(RAND_MAX)+(double)(1)) ) *Dominio_Rmax;
807     y=( (double)rand() / ((double)(RAND_MAX)+(double)(1)) ) *Dominio_Rmax;
808     z=( (double)rand() / ((double)(RAND_MAX)+(double)(1)) ) *Dominio_Hsup;
809     t=ThetaMax*( (double)(1.0*rand()) / ((double)(RAND_MAX)+(double)(1)) );
810     Particulas[0][0][i]= 1.5 *Dominio_Rint*cos(t);
811     Particulas[0][1][i]= 1.5 *Dominio_Rint*sin(t);
812 // Particulas[0][0][i]= x ;
813 // Particulas[0][1][i]= y ;
814 Particulas[0][2][i]= z ;
815     for (j=0;j<3;j++) {
816         for (k=0;k<maxpasadas;k++) Particulas[k+1][j][i]=Particulas[0][j][i];
817     }
818 }
819
820 void grid3D::drawVelGL(vector<double> U,vector<double> V,vector<double> W)
821 {
822 // cout<<"grid3D::drawVelGL"<<"primerdrawVelGL="<<primerdrawVelGL<<"nParticulas="<<nParticulas<<endl;
823 static double dt=0.1*sqr(Dominio_Rmax);
824 static int ipasadas=0;
825 double d2,d2min,x,y,z,dx,dy,dz,t;
826 int i,j,k,jmin;
827 if (primerdrawVelGL) {
828     for (k=0;k<maxpasadas;k++) {
829         for (j=0;j<3;j++) {
830             Particulas[k][j].init(nParticulas);
831         }
832     }
833     for (i=0;i<nParticulas;i++) {
834         PosINI(i);
835     }
836 }
837
838 primerdrawVelGL=0;
839 ipasadas++;
840
841 for (i=0;i<nParticulas;i++) {
842     x=Particulas[0][0][i];
843     y=Particulas[0][1][i];
844     z=Particulas[0][2][i];
845     d2min=1e10;
846     for (j=0;j<nV3D;j++) {
847         d2=sqr(x-v3D[j].x)+sqr(y-v3D[j].y)+sqr(z-v3D[j].z);
848         if (d2<d2min) {
849             d2min=d2;
850             jmin=j;
851         }
852     }
853 //cout<<"factorV="<<factorV<<endl;
854 double UUU=sqrt(sqr(U[jmin])+sqr(V[jmin])+sqr(W[jmin]));
855 dx=U[jmin]*dt/npasadas*factorV/UUU*factorVh;
856 dy=V[jmin]*dt/npasadas*factorV/UUU*factorVh;
857 dz=W[jmin]*dt/npasadas*factorV/UUU;
858
859 if (ipasadas>npasadas) {
860     for (k=maxpasadas-1;k>=0;k--)
861         for (j=0;j<3;j++)
862             Particulas[k+1][j][i]=Particulas[k][j][i];
863 }
864
865 x=Particulas[0][0][i]+=dx;
866 y=Particulas[0][1][i]+=dy;
867 z=Particulas[0][2][i]+=dz;
868
869 if ( sqrt(sqr(x-2 *Dominio_Rmax)+sqr(y)) < Dominio_Rint || x<xmin || x>xmax || y<ymin || y>ymax || z<zmin || z>zmax ) {
870     PosINI(i);
871 }
872 }
873

```

```

874 for (i=0;i<nParticulas;i++) {
875     for (k=0;k<maxpasadas;k++) {
876         glBegin(GL_LINES);
877         glVertex3d(Particulas[k][0][i],Particulas[k][1][i], Particulas[k][2][i]);
878         glVertex3d(Particulas[k+1][0][i],Particulas[k+1][1][i], Particulas[k+1][2][i]);
879         glEnd();
880     }
881     glPushMatrix();
882     glTranslated(Particulas[0][0][i],Particulas[0][1][i], Particulas[0][2][i]);
883 //     cout<<"esfera"<<endl;
884     FuncionesOpenGL::esfera(0.5*Dominio_Rint,4);
885     glPopMatrix();
886 }
887 if (ipasadas>npasadas) ipasadas=0;
888
889 }
890 }
891
892
893 void grid3D::drawVelGL2(vector<double> UU,vector<double> VV,vector<double> WW)
894 {
895     double x,y,z;
896     int i;
897     glPushAttrib( GL_LIGHTING_BIT );
898     glDisable( GL_LIGHTING );
899
900     for (i=0;i<nH3D;i++) {
901         x=h3D[i].centro.x;
902         y=h3D[i].centro.y;
903         z=h3D[i].centro.z;
904
905         glBegin(GL_LINES);
906         glColor3f(1,0,0);
907         glVertex3d(x,y,z);
908         glVertex3d(x+UU[i]*factorV,y +VV[i]*factorV,z+WW[i]*factorV);
909         glEnd();
910     }
911     glPopAttrib();
912
913 }
914 }
915
916
917 void grid3D::drawVoronoi()
918 {
919     int i,j,k;
920     static int xxip=0;
921     GLdouble winX1,winY1,winZ,winX2,winY2,winX3,winY3,nx,ny,nz;
922
923     FuncionesOpenGL::modelview_calculado=false;
924
925     glEnable(GL_NORMALIZE);
926     if (QuienGeneraPoligonos==1) {
927         for (i=0;i<nCaras;i++) {
928             nx=Cara[i].nx; ny=Cara[i].ny;nz=Cara[i].nz;
929             if (Cara[i].nVertexPolig >2) {
930                 FuncionesOpenGL::World2Win(Cara[i].vPolig.punto[0].x , Cara[i].vPolig.punto[0].y , Cara[i].vPolig.punto[0].z,&winX1,&winY1,&winZ);
931                 FuncionesOpenGL::World2Win(Cara[i].vPolig.punto[1].x , Cara[i].vPolig.punto[1].y , Cara[i].vPolig.punto[1].z,&winX2,&winY2,&winZ);
932                 FuncionesOpenGL::World2Win(Cara[i].vPolig.punto[Cara[i].nVertexPolig-1].x , Cara[i].vPolig.punto[Cara[i].nVertexPolig-1].y ,
933                     Cara[i].vPolig.punto[Cara[i].nVertexPolig-1].z,&winX3,&winY3,&winZ);
934                 if ((winX2-winX1)*(winY3-winY1)-(winY2-winY1)*(winX3-winX1) <0 ){
935                     nx = -nx; ny = -ny; nz = -nz; ;
936                 }
937             }
938             if (Dibuja3puntos && xxi==i && xxi != xxip) {
939                 xxip=xxi;
940                 for (j=0;j<Cara[i].nVertexPolig;j++)
941                     printf("Cara[%d].vPolig.punto[%d]={%6.2f , %6.2f ,
942 %6.2f}\n",i,j,Cara[i].vPolig.punto[j].x,Cara[i].vPolig.punto[j].y,Cara[i].vPolig.punto[j].z);
943             }
944             for (j=2;j<Cara[i].nVertexPolig;j++) {
945
946                 if (Dibuja3puntos && xxi==i) {
947                     FuncionesOpenGL::material(3); }
948                 else {FuncionesOpenGL::material(2);}
949
950
951                 glBegin(GL_TRIANGLES );
952                 glNormal3d(nx , ny , nz );
953                 glVertex3d(Cara[i].vPolig.punto[0].x,Cara[i].vPolig.punto[0].y,Cara[i].vPolig.punto[0].z);
954                 glVertex3d(Cara[i].vPolig.punto[j-1].x,Cara[i].vPolig.punto[j-1].y,Cara[i].vPolig.punto[j-1].z);
955                 glVertex3d(Cara[i].vPolig.punto[j].x,Cara[i].vPolig.punto[j].y,Cara[i].vPolig.punto[j].z);
956                 glEnd();
957             }
958         }
959     }
960     else if (QuienGeneraPoligonos==2) {
961         for (i=0;i<nH3D;i++) {
962             h3D[i].dibujado.assign( h3D[i].Poligono.size(),0);
963         }
964         for (i=0;i<nH3D;i++) {
965             for (k=0; k<h3D[i].Poligono.size(); k++) {
966                 if (h3D[i].dibujado[k] == 1) continue;
967                 vector<R3> *Pto;
968                 Pto= &(h3D[i].Poligono[k].punto);
969                 nx=h3D[i].Poligono[k].normal.x; ny=h3D[i].Poligono[k].normal.y;nz=h3D[i].Poligono[k].normal.z;

```

```

970     if (Pto->size() >2) {
971         FuncionesOpenGL::World2Win( (*Pto)[0].x , (*Pto)[0].y , (*Pto)[0].z,&winX1,&winY1,&winZ);
972         FuncionesOpenGL::World2Win( (*Pto)[Pto->size()/2].x , (*Pto)[Pto->size()/2].y , (*Pto)[Pto->size()/2].z,&winX2,&winY2,&winZ);
973         FuncionesOpenGL::World2Win( (*Pto)[Pto->size()-1].x , (*Pto)[Pto->size()-1].y ,
974             (*Pto)[Pto->size()-1].z,&winX3,&winY3,&winZ);
975         if ((winX2-winX1)*(winY3-winY1)-(winY2-winY1)*(winX3-winX1) <0 ){
976             nx = -nx; ny = -ny; nz = -nz; ;
977         }
978     }
979     if (Dibuja3puntos && xxi==i) {
980         FuncionesOpenGL::material(2); }
981     else {FuncionesOpenGL::material(3);}
982     for (j=2; j<Pto->size(); j++) {
983         glBegin(GL_TRIANGLES);
984         glNormal3d(nx , ny , nz );
985         glVertex3d( (*Pto)[ 0].x, (*Pto)[ 0].y, (*Pto)[0].z);
986         glVertex3d( (*Pto)[j-1].x, (*Pto)[j-1].y, (*Pto)[j-1].z);
987         glVertex3d( (*Pto)[ j].x, (*Pto)[ j].y, (*Pto)[j].z);
988         glEnd();
989     }
990 }
991 if(ModoDibujaNormales) {
992     double xg,yg,zg,nnn;
993     glPushAttrib( GL_LIGHTING_BIT );
994     glDisable( GL_LIGHTING );
995
996     glBegin(GL_LINES);
997     glColor3f(1,0,0);
998     xg=h3D[i].Poligono[k].centro.x;
999     yg=h3D[i].Poligono[k].centro.y;
1000     zg=h3D[i].Poligono[k].centro.z;
1001     glVertex3d(xg,yg,zg);
1002     nnn=sqrt(sqr(nx)+sqr(ny)+sqr(nz));
1003     glVertex3d(xg+nx/nnn/10,yg+ny/nnn/10,zg+nz/nnn/10);
1004     glEnd();
1005     glPopAttrib();
1006 }
1007
1008 h3D[i].dibujado[k]=1;
1009 if (h3D[i].tipo_vecino[k]==ES_BLOQUE) {
1010     for (j=0; j<h3D[ h3D[i].tipo_vecino[k] ].dibujado.size(); j++) {
1011         if ( h3D[ h3D[i].vecino[k] ].tipo_vecino[j] ==ES_BLOQUE
1012             && h3D[ h3D[i].vecino[k] ].vecino[j] == i) {
1013
1014             h3D[ h3D[i].vecino[k] ].dibujado[j]=1;
1015             break;
1016         }
1017     }
1018 }
1019 }
1020 }
1021
1022 }
1023 if (Dibuja3puntos) {
1024     glTranslatef(xx0,yy0,zz0);
1025     FuncionesOpenGL::material(1); FuncionesOpenGL::esfera(0.02,3);
1026     glTranslatef(xx1-xx0,yy1-yy0,zz1-zz0);
1027     FuncionesOpenGL::material(2); FuncionesOpenGL::esfera(0.02,3);
1028     glTranslatef(xx2-xx1,yy2-yy1,zz2-zz1);
1029     FuncionesOpenGL::material(2); FuncionesOpenGL::esfera(0.02,3);
1030     glTranslatef(-xx2,-yy2,-zz2);
1031 }
1032 }
1033 }
1034
1035
1036 void grid3D::minmax()
1037 {
1038     int i;
1039     double x,y,z;
1040     xmin=xmax=v3D[0].x;
1041     ymin=ymax=v3D[0].y;
1042     zmin=zmax=v3D[0].z;
1043     for (i=1;i<nV3D;i++) {
1044         x=v3D[i].x;
1045         y=v3D[i].y;
1046         z=v3D[i].z;
1047         if (x < xmin) xmin=x;
1048         if (x > xmax) xmax=x;
1049         if (y < ymin) ymin=y;
1050         if (y > ymax) ymax=y;
1051         if (z < zmin) zmin=z;
1052         if (z > zmax) zmax=z;
1053     }
1054 }
1055
1056 void grid3D::drawGL()
1057 {
1058     int i;
1059     char s[100];
1060     static int version=1;//0:dibuja los cubos, 1:dibuja las caras
1061
1062     FuncionesOpenGL::ObtieneMatrices();
1063     if (version==0) {
1064         for (i=0;i<nH3D;i++) {
1065             // 2-----3

```

```

1067         // 6-|----7 |
1068         // | | | |
1069         // | 0----|-1
1070         // | | | |
1071         // 4-----5
1072
1073         h3D[i].draw_caraGL(0,1,3,2);
1074         h3D[i].draw_caraGL(0,2,6,4);
1075         h3D[i].draw_caraGL(0,4,5,1);
1076     }
1077 } else if (version ==1) {
1078     if (Modo_DibujaCentroBloques) {
1079         for (i=0;i<nH3D;i++) {
1080             glTranslatef(h3D[i].centro.x,h3D[i].centro.y,h3D[i].centro.z);
1081             FuncionesOpenGL::material(1); FuncionesOpenGL::esfera(0.01/Escala,3);
1082             glTranslatef(-h3D[i].centro.x,-h3D[i].centro.y,-h3D[i].centro.z);
1083         }
1084     }
1085     for (i=0;i<nCaras;i++) {
1086         if (0*Modo_DibujaCentroCaras) {
1087             glTranslatef(Cara[i].centro.x,Cara[i].centro.y,Cara[i].centro.z);
1088             FuncionesOpenGL::material(0); FuncionesOpenGL::esfera(0.04/Escala,3);
1089             glTranslatef(-Cara[i].centro.x,-Cara[i].centro.y,-Cara[i].centro.z);
1090         }
1091         if (ModoDibujaFrontera && Cara[i].nh ==1) {
1092             FuncionesOpenGL::material(Cara[i].iBC+10);draw_caraGL(Cara[i].iv);
1093         }
1094         if (ModoDibujaInterior && Cara[i].nh >1) {
1095             FuncionesOpenGL::material(2);draw_caraGL(Cara[i].iv);
1096             if (0*Modo_DibujaCentroCaras) {
1097                 glTranslatef(Cara[i].centro.x,Cara[i].centro.y,Cara[i].centro.z);
1098                 FuncionesOpenGL::material(1); FuncionesOpenGL::esfera(0.04,3);
1099                 glTranslatef(-Cara[i].centro.x,-Cara[i].centro.y,-Cara[i].centro.z);
1100             }
1101         }
1102     }
1103 }
1104 }
1105
1106 if (Modo_DibujaEdges) {
1107     glPushAttrib( GL_LIGHTING_BIT );
1108     glDisable( GL_LIGHTING );
1109     for (i=0;i<nH3D;i++) {
1110         // 2-----3
1111         // 6-|----7 |
1112         // | | | |
1113         // | 0----|-1
1114         // | | | |
1115         // 4-----5
1116
1117         h3D[i].draw_edgeGL(6,7);
1118         h3D[i].draw_edgeGL(3,7);
1119         h3D[i].draw_edgeGL(5,7);
1120     }
1121 }
1122 glPopAttrib();
1123 }
1124
1125 if (MODO_Numerah){
1126     FuncionesOpenGL::material(0);
1127
1128     for (i=0;i<nH3D;i++) {
1129         char *p;
1130         glPushMatrix();
1131
1132         glTranslatef(h3D[i].centro.x,h3D[i].centro.y,h3D[i].centro.z);
1133         //glTranslatef(0,0,.5);
1134
1135         glmultMatrixf((GLfloat *)MatrizRotacionGlobalINV);
1136         glScalef(NumEscala,NumEscala,NumEscala);
1137         float y0=4.0/Escala,dy=1.7/Escala;
1138         glRasterPos3f(-3/Escala, y0=dy ,0);
1139         sprintf(s,"%d",i);
1140         for(p = s; *p; p++) {
1141             glutBitmapCharacter(GLUT_BITMAP_8_BY_13, *p);
1142         }
1143         if (MODO_NumerahFF) {
1144             glRasterPos3f(-3/Escala, y0=dy ,0);
1145             sprintf(s,"FF=%.3f",FF[i]);
1146             for(p = s; *p; p++) {
1147                 glutBitmapCharacter(GLUT_BITMAP_8_BY_13, *p);
1148             }
1149             glRasterPos3f(-3/Escala, y0=dy ,0);
1150             sprintf(s,"U=%.3f",sqrt(sqr(UU[i])+sqr(VV[i])+sqr(WW[i])));
1151             for(p = s; *p; p++) {
1152                 glutBitmapCharacter(GLUT_BITMAP_8_BY_13, *p);
1153             }
1154             glRasterPos3f(-3/Escala, y0=dy ,0);
1155             sprintf(s,"d=%.3f",sqrt(sqr(h3D[i].centro.x)+sqr(h3D[i].centro.y)));
1156             for(p = s; *p; p++) {
1157                 glutBitmapCharacter(GLUT_BITMAP_8_BY_13, *p);
1158             }
1159         }
1160     }
1161 }
1162

```

```

1164         }
1165         /*
1166         glRasterPos3f(-3/Escala, y0-=dy ,0);
1167         sprintf(s,"VV=%.3f",VV[i]);
1168         for(p = s; *p; p++) {
1169             glutBitmapCharacter(GLUT_BITMAP_8_BY_13, *p);
1170         }
1171         glRasterPos3f(-3/Escala, y0-=dy ,0);
1172         sprintf(s,"WW=%.3f",WW[i]);
1173         for(p = s; *p; p++) {
1174             glutBitmapCharacter(GLUT_BITMAP_8_BY_13, *p);
1175         }
1176         */
1177     }
1178
1179     // FuncionesOpenGL::Print(s);
1180     glPopMatrix();
1181 }
1182 }
1183 }
1184
1185
1186 void grid3D::drawGL(vector<double> F)
1187 {
1188     int i;
1189     char s[100];
1190     double minF,maxF;
1191
1192     maxF=minF=F[0];
1193     for (i=0;i<nV3D;i++) {
1194         if (minF>F[i]) minF=F[i];
1195         if (maxF<F[i]) maxF=F[i];
1196     }
1197     glPushAttrib( GL_LIGHTING_BIT );
1198     // glEnable(GL_COLOR_MATERIAL);
1199     // glDisable( GL_LIGHTING );
1200
1201     for (i=0;i<nCaras;i++) {
1202         // 2-----3
1203         // /|      /|
1204         // 6-|----7 |
1205         // | |    | |
1206         // | 0----|-1
1207         // /|      /|
1208         // 4-----5
1209
1210         //TODO
1211
1212         if (ModoDibujaFrontera && Cara[i].nh ==1) {
1213             FuncionesOpenGL::material(Cara[i].IBC+10);draw_caraGL(F,minF,maxF,Cara[i].iv);
1214         }
1215         if (ModoDibujaInterior && Cara[i].nh >1) {
1216             FuncionesOpenGL::material(2);draw_caraGL(F,minF,maxF,Cara[i].iv);
1217             if (0*Modo_DibujaCentroCaras) {
1218                 glTranslatef(Cara[i].centro.x,Cara[i].centro.y,Cara[i].centro.z);
1219                 FuncionesOpenGL::material(1); FuncionesOpenGL::esfera(0.04,3);
1220                 glTranslatef(-Cara[i].centro.x,-Cara[i].centro.y,-Cara[i].centro.z);
1221             }
1222         }
1223     }
1224 }
1225
1226
1227 if (Modo_DibujaEdges) {
1228     for (i=0;i<nH3D;i++) {
1229         // 2-----3
1230         // /|      /|
1231         // 6-|----7 |
1232         // | |    | |
1233         // | 0----|-1
1234         // /|      /|
1235         // 4-----5
1236
1237         h3D[i].draw_edgeGL(6,7);
1238         h3D[i].draw_edgeGL(3,7);
1239         h3D[i].draw_edgeGL(5,7);
1240     }
1241 }
1242
1243 glPopAttrib();
1244
1245 if (NumON){
1246     FuncionesOpenGL::material(0);
1247
1248     for (i=0;i<nV3D;i++) {
1249         glPushMatrix();
1250
1251         glTranslatef(v3D[i].x,v3D[i].y,v3D[i].z);
1252         glMultMatrixf((GLfloat *)MatrizRotacionGlobalINV);
1253         glScalef(NumEscala,NumEscala,NumEscala);
1254         glTranslatef(0,0,.5);
1255         sprintf(s,"%d",i);
1256         FuncionesOpenGL::Print(s);
1257         glPopMatrix();
1258     }
1259 }
1260 }

```

```

1261 }
1262
1263
1264 void grid3D::cubo(int ix,int iy,int iz,float Lx,float Ly,float Lz)
1265 {
1266     int i,j,k;
1267     nH3D=ix*iy*iz;
1268     nV3D=(ix+1)*(iy+1)*(iz+1);
1269     v3D.resize(nV3D,Vertex3D());
1270     h3D.resize(nH3D,Hexa3D());
1271     for (i=0;i<=ix;i++) {
1272         for (j=0;j<=iy;j++) {
1273             for (k=0;k<=iz;k++) {
1274                 if ((i<ix)&&(j<iy)&&(k<iz)) {
1275
1276                     h3D[i*(iy*iz)+j*(iz)+k].papa=this;
1277                     h3D[i*(iy*iz)+j*(iz)+k].no=i*(iy*iz)+j*(iz)+k;
1278                     h3D[i*(iy*iz)+j*(iz)+k].iv[0]=i*(iy+1)*(iz+1)+ j *(iz+1)+ k;
1279                     h3D[i*(iy*iz)+j*(iz)+k].iv[1]=i*(iy+1)*(iz+1)+(j+1)*(iz+1)+ k;
1280                     h3D[i*(iy*iz)+j*(iz)+k].iv[2]=i*(iy+1)*(iz+1)+ j *(iz+1)+(k+1);
1281                     h3D[i*(iy*iz)+j*(iz)+k].iv[3]=i*(iy+1)*(iz+1)+(j+1)*(iz+1)+(k+1);
1282                     h3D[i*(iy*iz)+j*(iz)+k].iv[4]=(i+1)*(iy+1)*(iz+1)+ j *(iz+1)+ k;
1283                     h3D[i*(iy*iz)+j*(iz)+k].iv[5]=(i+1)*(iy+1)*(iz+1)+(j+1)*(iz+1)+ k;
1284                     h3D[i*(iy*iz)+j*(iz)+k].iv[6]=(i+1)*(iy+1)*(iz+1)+ j *(iz+1)+(k+1);
1285                     h3D[i*(iy*iz)+j*(iz)+k].iv[7]=(i+1)*(iy+1)*(iz+1)+(j+1)*(iz+1)+(k+1);
1286                 }
1287                 v3D[i*(iy+1)*(iz+1)+ j *(iz+1)+ k].no=i*(iy+1)*(iz+1)+ j *(iz+1)+ k;
1288                 v3D[i*(iy+1)*(iz+1)+ j *(iz+1)+ k].papa=this;
1289                 v3D[i*(iy+1)*(iz+1)+ j *(iz+1)+ k].x=Lx/ix*i;
1290                 v3D[i*(iy+1)*(iz+1)+ j *(iz+1)+ k].y=Ly/iy*j;
1291                 v3D[i*(iy+1)*(iz+1)+ j *(iz+1)+ k].z=Lz/iz*k;
1292             }
1293         }
1294     }
1295 }
1296
1297 void grid3D::Junta(grid3D g1,grid3D g2)
1298 {
1299     int i,j;
1300     double DIJ;
1301     vector<int> i2enUnion(g2.nV3D);
1302     const double eps=1e-10;
1303     v3D=g1.v3D;
1304     nV3D=g1.nV3D;
1305     h3D=g1.h3D;
1306     nH3D=g1.nH3D+g2.nH3D;
1307     for (i=0;i<g2.nV3D;i++) {
1308         for (j=0;j<g1.nV3D;j++) {
1309             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);
1310             if (DIJ<eps) break;
1311         }
1312         if (j<g1.nV3D){
1313             i2enUnion[i]=j;
1314         } else {
1315             i2enUnion[i]=nV3D;
1316             v3D.push_back(g2.v3D[i]);
1317             nV3D++;
1318         }
1319     }
1320     for (i=0;i<g2.nH3D;i++) {
1321         h3D.push_back(g2.h3D[i]);
1322         for (j=0;j<8;j++) {
1323             h3D[g1.nH3D+i].iv[j]=i2enUnion[g2.h3D[i].iv[j]];
1324         }
1325     }
1326     for (i=0;i<nV3D;i++) {
1327         v3D[i].no=i;
1328         v3D[i].papa=this;
1329     }
1330     for (i=0;i<nH3D;i++) {
1331         h3D[i].no=i;
1332         h3D[i].papa=this;
1333     }
1334 }
1335
1336 }
1337
1338 void grid3D::Junta(grid3D g2)
1339 {
1340     int i,j,nv1,nH1;
1341     double DIJ;
1342     vector<int> i2enUnion(g2.nV3D);
1343     const double eps=1e-10;
1344     nv1=nV3D;
1345     nH1=nH3D;
1346     nH3D+=g2.nH3D;
1347     for (i=0;i<g2.nV3D;i++) {
1348         for (j=0;j<nv1;j++) {
1349             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);
1350             if (DIJ<eps) break;
1351         }
1352         if (j<nv1){
1353             i2enUnion[i]=j;
1354         } else {
1355             i2enUnion[i]=nV3D;
1356             v3D.push_back(g2.v3D[i]);
1357             nv1++;

```

```

1358     }
1359 }
1360 for (i=0;i<g2.nH3D;i++) {
1361     h3D.push_back(g2.h3D[i]);
1362     for (j=0;j<8;j++) {
1363         h3D[nH1+i].iv[j]=i2enUnion[g2.h3D[i].iv[j]];
1364     }
1365 }
1366 for (i=0;i<nV3D;i++) {
1367     v3D[i].no=i;
1368     v3D[i].papa=this;
1369 }
1370 for (i=0;i<nH3D;i++) {
1371     h3D[i].no=i;
1372     h3D[i].papa=this;
1373 }
1374
1375
1376 }
1377
1378 void grid3D::Rota90Z()
1379 {
1380     int i;
1381     double xL,yL;
1382     for (i=0;i<nV3D;i++){
1383         xL=v3D[i].x;
1384         yL=v3D[i].y;
1385         v3D[i].x = -yL;
1386         v3D[i].y = xL;
1387     }
1388 }
1389
1390 void grid3D::Traslada(double dx,double dy,double dz)
1391 {
1392     int i;
1393     for (i=0;i<nV3D;i++){
1394         v3D[i].x += dx;
1395         v3D[i].y += dy;
1396         v3D[i].z += dz;
1397     }
1398 }
1399
1400

```