```
#pragma once

#include <sstream>

extern double ThetaMax;

void SaveOrRead(char *ifile_name,int iSaveReadMode);

void AddMensaje(char* newMensaje)
{
    int i;
    for (i=0;i<NMensajes-1;i++) {
        strcpy(Mensaje[i],Mensaje[i+1]);
    }
    strcpy(Mensaje[NMensajes-1],newMensaje);
}


void print_text(int x, int y, char* s)
{
    int lines;
    char* p;
    glDisable(GL_DEPTH_TEST);
    glMatrixMode(GL_PROJECTION);
    glPushMatrix();
    glLoadIdentity();
    glOrtho(0, width,
            0, height, -1, 1);
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
    glLoadIdentity();
    glColor3ub(128, 0, 255);
    glRasterPos2i(x, y);
    for(p = s; *p; p++) {
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, *p);
    }
    glMatrixMode(GL_PROJECTION);
    glPopMatrix();
    glMatrixMode(GL_MODELVIEW);
    glPopMatrix();
    glEnable(GL_DEPTH_TEST);
}

void TrasponeMatriz() {

    MatrizRotacionGlobalINV[0]=MatrizRotacionGlobal[0];
    MatrizRotacionGlobalINV[1]=MatrizRotacionGlobal[4];
    MatrizRotacionGlobalINV[2]=MatrizRotacionGlobal[8];
    MatrizRotacionGlobalINV[3]=MatrizRotacionGlobal[12];

    MatrizRotacionGlobalINV[4]=MatrizRotacionGlobal[1];
    MatrizRotacionGlobalINV[5]=MatrizRotacionGlobal[5];
    MatrizRotacionGlobalINV[6]=MatrizRotacionGlobal[9];
    MatrizRotacionGlobalINV[7]=MatrizRotacionGlobal[13];

    MatrizRotacionGlobalINV[8]=MatrizRotacionGlobal[2];
    MatrizRotacionGlobalINV[9]=MatrizRotacionGlobal[6];
    MatrizRotacionGlobalINV[10]=MatrizRotacionGlobal[10];
    MatrizRotacionGlobalINV[11]=MatrizRotacionGlobal[14];

    MatrizRotacionGlobalINV[12]=MatrizRotacionGlobal[3];
    MatrizRotacionGlobalINV[13]=MatrizRotacionGlobal[7];
    MatrizRotacionGlobalINV[14]=MatrizRotacionGlobal[11];
    MatrizRotacionGlobalINV[15]=MatrizRotacionGlobal[15];
}

void CopiaBloquesAVertices(vector<double> FFB,vector<double> &FV,double *minF,double *maxF,int iBC) {
    int i,j;
    vector<int> cuantos(FV.size());
    for (i=0;i<gtotal->nV3D;i++) {
        cuantos[i]=0;
```

```
72          FV[i]=0;
73      }
74      for (i=0; i<gtotal->nH3D ; i++) {
75          for( j=0 ; j<8 ; j++) {
76              int iv=gtotal->h3D[i].iv[j];
77              FV[iv] += FFB[i];
78              cuantos[iv] ++;
79          }
80      }
81
82      if (iBC>0) {
83
84          for (i=0; i<gtotal->nCaras ; i++) {
85              if (gtotal->Cara[i].iBC ==2 || gtotal->Cara[i].iBC ==3) {
86                  for (j=0; j<4 ; j++) {
87                      FV[gtotal->Cara[i].iv[j]]=0;
88                      cuantos[gtotal->Cara[i].iv[j]]=0;
89                  }
90              }
91          }
92          for (i=0; i<gtotal->nCaras ; i++) {
93              if (gtotal->Cara[i].iBC ==2 || gtotal->Cara[i].iBC ==3) {
94                  for (j=0; j<4 ; j++) {
95                      switch (iBC)
96                      {
97                      case 1:
98                          FV[gtotal->Cara[i].iv[j]] += gtotal->Cara[i].BC;
99                          cuantos[gtotal->Cara[i].iv[j]]++;
100                         break;
101                     case 2:
102                         FV[gtotal->Cara[i].iv[j]] += gtotal->Cara[i].BC2;
103                         cuantos[gtotal->Cara[i].iv[j]]++;
104                         if(1==0)
105                         cout<<"g.Cara[i].BC2="<<gtotal->Cara[i].BC2
106                                 <<"\tcuantos[g.Cara[i].iv[j]]="<<cuantos[gtotal->Cara[i].iv[j]]
107                                                                 <<"g.Cara[i].iv[j]="<<gtotal->Cara[i].iv[j]
108                                                                 <<endl;
109                         break;
110                     };
111                 }
112             }
113         }
114     }
115
116     *maxF=*minF=FV[0]/cuantos[0];
117     for (i=0;i<gtotal->nV3D;i++) {
118         FV[i] /= cuantos[i];
119         if (*minF>FV[i]) {
120             *minF=FV[i];
121             if(1==0) cout<<"FV[i]="<<FV[i]<<"\tcuantos[i]="<<cuantos[i]<<"\ti="<<i<<endl;
122         }
123         if (*maxF<FV[i]) *maxF=FV[i];
124     }
125
126 }
127
128 void DrawGraphics()
129 {
130     if (DBG) cout<<"DrawGraphics()"<<endl;
131     cout<<"DrawGraphics()"<<endl;
132     static int contador=0;
133     static int nframes2=0;
134     char linea[100];
135     int i,j;
136
137     if (!DrawYES) return;
138     DrawYES=0;
139
140     if (DBG) cout<<"DrawGraphics()138"<<endl;
141     glClear(GL_COLOR_BUFFER_BIT );
142     glClear(GL_DEPTH_BUFFER_BIT);
```

```
143
144     glDisable(GL_CLIP_PLANE0);
145     if (DBG) cout<<"DrawGraphics()143"<<endl;
146
147
148
149
150     if (0) {
151         glPushAttrib( GL_LIGHTING_BIT );
152         glDisable( GL_LIGHTING );
153
154
155         glMatrixMode (GL_MODELVIEW); glPushMatrix (); glLoadIdentity ();
156         glMatrixMode (GL_PROJECTION); glPushMatrix (); glLoadIdentity ();
157
158         glBegin (GL_QUADS);
159         glColor3d(.1, 0, .1);
160         glVertex3f (-1, -1, -1.5); glVertex3f (1, -1, -1.5); glVertex3f (1, 1, -1.5); glVertex3f (-1, 1, -1.5);
161         glEnd ();
162         glPopMatrix (); glMatrixMode (GL_MODELVIEW); glPopMatrix ();
163
164
165         glPopAttrib();
166     }
167     glClear(GL_DEPTH_BUFFER_BIT);
168     if (DBG) cout<<"DrawGraphics()166"<<endl;
169
170     /*
171     //Test Ortogonal
172     cout<<"MatrizRotacionGlobal="<<endl;
173     cout<< MatrizRotacionGlobal[0]<<"\t"<<MatrizRotacionGlobal[1]<<"\t"
174             <<MatrizRotacionGlobal[2]<<"\t"<<MatrizRotacionGlobal[3]<<endl;
175     cout<< MatrizRotacionGlobal[4]<<"\t"<<MatrizRotacionGlobal[5]<<"\t"
176             <<MatrizRotacionGlobal[6]<<"\t"<<MatrizRotacionGlobal[7]<<endl;
177     cout<< MatrizRotacionGlobal[8]<<"\t"<<MatrizRotacionGlobal[9]<<"\t"
178             <<MatrizRotacionGlobal[10]<<"\t"<<MatrizRotacionGlobal[11]<<endl;
179     cout<< MatrizRotacionGlobal[12]<<"\t"<<MatrizRotacionGlobal[13]<<"\t"
180             <<MatrizRotacionGlobal[14]<<"\t"<<MatrizRotacionGlobal[15]<<endl;
181
182     cout<<"MatrizRotacionGlobalINV="<<endl;
183
184     cout<< MatrizRotacionGlobalINV[0]<<"\t"<<MatrizRotacionGlobalINV[1]<<"\t"
185             <<MatrizRotacionGlobalINV[2]<<"\t"<<MatrizRotacionGlobalINV[3]<<endl;
186     cout<< MatrizRotacionGlobalINV[4]<<"\t"<<MatrizRotacionGlobalINV[5]<<"\t"
187             <<MatrizRotacionGlobalINV[6]<<"\t"<<MatrizRotacionGlobalINV[7]<<endl;
188     cout<< MatrizRotacionGlobalINV[8]<<"\t"<<MatrizRotacionGlobalINV[9]<<"\t"
189             <<MatrizRotacionGlobalINV[10]<<"\t"<<MatrizRotacionGlobalINV[11]<<endl;
190     cout<< MatrizRotacionGlobalINV[12]<<"\t"<<MatrizRotacionGlobalINV[13]<<"\t"
191             <<MatrizRotacionGlobalINV[14]<<"\t"<<MatrizRotacionGlobalINV[15]<<endl;
192
193      */
194
195     // Set location in front of camera
196     glLoadIdentity();
197     glTranslated( 0, 0, -5);
198
199     FuncionesOpenGL::ActivaLuz0();
200     glShadeModel(GL_SMOOTH);
201
202     glTranslatef(vecUEsfera[0], vecUEsfera[1],vecUEsfera[2]);
203
204
205     glScalef(Escala, Escala, Escala);
206     glMultMatrixf((GLfloat *)MatrizRotacionGlobal);
207     //Cursor centro
208     FuncionesOpenGL::material(0);   FuncionesOpenGL::esfera(0.002,20);
209     glTranslatef(-vecXEsfera[0],-vecXEsfera[1],-vecXEsfera[2]);
210
211     if (DBG) cout<<"DrawGraphics()209"<<endl;
212
213     if (MODO_Ejes) {
```

```
214         FuncionesOpenGL::ejes();
215     }
216
217     if (ClippingON) {
218         double eq[4];
219         eq[0]=-Ax;  eq[1]=-By;  eq[2]=-Cz;  eq[3]=DD;
220         glClipPlane(GL_CLIP_PLANE0, eq);
221         glEnable(GL_CLIP_PLANE0);
222     }
223     if (DBG) cout<<"DrawGraphics()221"
224              <<"\nColorON="<<ColorON
225              <<"\n gtotal="<<gtotal
226              <<endl;
227
228
229     //glEnable(GL_CULL_FACE);
230     switch (ColorON)
231     {
232     case 0:  //Caso Normal
233         FuncionesOpenGL::material(1);gtotal->drawGL();
234         break;
235     case 1:  //Dibuja Campo 1
236         gtotal->drawGL(F);
237         break;
238     case 2:  //Dibuja Campo 2
239         gtotal->drawGL(F2Nodos);
240         break;
241     case 3:
242         FuncionesOpenGL::material(1);gtotal->drawGL();
243         FuncionesOpenGL::material(2);gtotal->drawVoronoi();
244         break;
245     };
246     if (DBG) cout<<"DrawGraphics()244"<<endl;
247
248     if (MODO_CampoVelocidades) {
249         FuncionesOpenGL::material(0);   gtotal->drawVelGL(U,V,W);
250     }
251     if (DBG) cout<<"DrawGraphics()249"<<endl;
252     if (MODO_CampoVelocidades2) {
253         FuncionesOpenGL::material(1);   gtotal->drawVelGL2(UU,VV,WW);
254     }
255     if (DBG) cout<<"DrawGraphics()253"<<endl;
256
257     {/////////////////bloque que imprime texto......
258         glDisable(GL_CLIP_PLANE0);
259         glPushAttrib( GL_LIGHTING_BIT );
260         glDisable( GL_LIGHTING );
261
262         if (DBG) cout<<"DrawGraphics()260"
263                  <<"\n MODO_MenuMENSAJES="<<MODO_MenuMENSAJES
264                  <<endl;
265         if (MODO_MenuMENSAJES) DrawMensajes();
266         {
267             clock2F=clockF=clock ();
268             nframes++;nframes2++;
269             if (DBG) cout<<"DrawGraphics()267"<<endl;
270
271             FPS=(nframes*1.0)*CLOCKS_PER_SEC/(clockF-clock0);
272             FPS2=(nframes2*1.0)*CLOCKS_PER_SEC/(clock2F-clock2);
273             if (clockF-clock0>CLOCKS_PER_SEC) {
274                 glColor3d(.8,.9, 1);
275                 glRasterPos3f(-2.07*aspect,1.95,5);
276                 ///modificado GLUT
277                 sprintf(s,"FPS=%.2f (%.2f)\r",FPS2,FPS);//fflush(stdout);
278                 //sprintf(s,"%.0f F/S",FPS);FuncionesOpenGL::Print(s,1);
279                 clock0=clockF;
280                 nframes=0;
281                 glui_FPS->set_name(s);
282                 //                  printf("%s",s);
283             }
284             //          print_text(width-150,5,s);
```

```
285
286         }
287         glPopAttrib();
288     }
289
290     glutSwapBuffers();
291
292     DrawYES=1;
293     if (DBG) cout<<"DrawGraphics():END"<<endl;
294
295 }
296
297 save(vector<double> &F,ofstream &myfile) {
298     int i;
299     myfile  <<F.size()<<endl;
300     for (i=0;i<     F.size();i++) { myfile<<     F[i]<<" "; } myfile<<endl;
301
302 }
303
304 read(vector<double> &F,ifstream &myfile) {
305     int i,sizeL;
306     myfile  >> sizeL; F.resize(sizeL);
307     for (i=0;i<F.size();i++) { myfile>>F[i]; }
308
309 }
310
311 void DrawMensajes()
312 {
313     int i;
314     glPushAttrib( GL_LIGHTING_BIT );
315
316     glDisable( GL_LIGHTING );
317
318
319     glColor3d(1,1, 0.7);
320
321     for (i=0;i<NMensajes;i++){
322         print_text(3,height-17*(NMensajes-i),Mensaje[i]);
323     }
324
325
326
327
328     switch (mode)
329     {
330     case modeT :
331         glPushMatrix();
332         glColor3d(.8,.9, 1);
333
334         glTranslatef(-2.07*aspect,-2,5);
335         glRasterPos2f(0,0);
336
337         glListBase(baseBIT);
338         glCallLists(12, GL_UNSIGNED_BYTE, "(T)raslación");
339
340         glPopMatrix();
341         break;
342     case modeR:
343         glPushMatrix();
344         glColor3d(.8,.9, 1);
345
346         glTranslatef(-2.07*aspect,-2,5);
347         glRasterPos2f(0,0);
348
349         glListBase(baseBIT);
350         glCallLists(10, GL_UNSIGNED_BYTE, "(R)otación");
351
352         glPopMatrix();
353         break;
354     }
355     glPopAttrib();
```

```
356
357 }
358
359
360 void MatrizXvector4(GLfloat *mat,GLfloat *vec,GLfloat *resultado) {
361     resultado[0]=mat[0]*vec[0] + mat[4]*vec[1] + mat[ 8]*vec[2] + mat[12]*vec[3];
362     resultado[1]=mat[1]*vec[0] + mat[5]*vec[1] + mat[ 9]*vec[2] + mat[13]*vec[3];
363     resultado[2]=mat[2]*vec[0] + mat[6]*vec[1] + mat[10]*vec[2] + mat[14]*vec[3];
364     resultado[3]=mat[3]*vec[0] + mat[7]*vec[1] + mat[11]*vec[2] + mat[15]*vec[3];
365 }
366
367
368 void ZGlobal(double*v) {
369     v[0]=MatrizRotacionGlobalINV[8];
370     v[1]=MatrizRotacionGlobalINV[9];
371     v[2]=MatrizRotacionGlobalINV[10];
372 }
373
374 void inicializacionGL()
375 {
376     if (DBG) cout<<"inicializacionGL()"<<endl;
377     glEnable(GL_DEPTH_TEST);
378     glMatrixMode(GL_PROJECTION);
379     glLoadIdentity();
380
381     gluPerspective( 0.0 , 640.0/480.0, 0.001, 100.0);
382     glMatrixMode(GL_MODELVIEW);
383
384     glLoadIdentity();
385     glRotatef(-60,1,0,0);
386     glRotatef(-135,0,0,1);
387     glGetFloatv(GL_MODELVIEW_MATRIX, (GLfloat*)MatrizRotacionGlobal);
388     glLoadIdentity();
389     glRotatef(135,0,0,1);
390     glRotatef(60,1,0,0);
391     glGetFloatv(GL_MODELVIEW_MATRIX, (GLfloat*)MatrizRotacionGlobalINV);
392     FuncionesOpenGL::modelview_calculado=false;
393     if (DBG) cout<<"inicializacionGL():END"<<endl;
394 }
395 void inicializacion()
396 {
397     int i,j,iL,malla;
398     double x1,x2,y1,y2,xL,yL,R,xF,yF,RF,th,pi=4*atan(1.0);
399
400     for (i=0;i<NMensajes;i++) {
401         Mensaje[i]=(char *)(new char[100]);
402         strcpy(Mensaje[i],"");
403     }
404
405     clock2=clock0=clock ();
406     glClearColor(.9, .9, 1.0, 1.0f);
407     glClearDepth(1.0);
408
409     malla=0;
410     gtotal->cubo(5,2,2);
411     g1.cubo(5,2,2);
412     g2.cubo(5,2,2);
413     switch (malla)
414     {
415     case 1:
416         gtotal->cubo(2,2,2);
417         g2.cubo(1,2,2);
418         x1=0.1;x2=1;
419         y1=0.1;y2=1;
420         for (i=0;i<gtotal->nV3D ;i++){
421             xL=gtotal->v3D[i].x ;
422             yL=gtotal->v3D[i].y ;
423             gtotal->v3D[i].x = x1+xL*(x2-x1);
424             gtotal->v3D[i].y = yL*(y1+xL*(y2-y1));
425         }
426         x1=x2;x2=1.5;
```

```
427         for (i=0;i<g2.nV3D ;i++){
428             xL=g2.v3D[i].x ;
429             g2.v3D[i].x = x1+xL*(x2-x1);
430         }
431         gtotal->Junta(g2);
432         g2=*gtotal;
433 
434         for (i=0;i<g2.nV3D ;i++){
435             g2.v3D[i].y *= -1;
436         }
437         for (i=0;i<g2.nH3D ;i++){
438             for (j=0;j<4;j++) {
439                 iL=g2.h3D[i].iv[j];
440                 g2.h3D[i].iv[j]=g2.h3D[i].iv[7-j];
441                 g2.h3D[i].iv[7-j]=iL;
442             }
443         }
444         gtotal->Junta(g2);
445 
446         g2=*gtotal;
447         g2.Rota90Z();   gtotal->Junta(g2);
448         g2.Rota90Z();   gtotal->Junta(g2);
449         g2.Rota90Z();   gtotal->Junta(g2);
450         g2.cubo(1,1,2);
451         x1=1;x2=1.5;
452         for (i=0;i<g2.nV3D ;i++){
453             xL=g2.v3D[i].x ;
454             yL=g2.v3D[i].y ;
455             g2.v3D[i].x = x1+xL*(x2-x1);
456             g2.v3D[i].y = x1+yL*(x2-x1);
457         }
458         gtotal->Junta(g2);
459         g2.Rota90Z();   gtotal->Junta(g2);
460         g2.Rota90Z();   gtotal->Junta(g2);
461         g2.Rota90Z();   gtotal->Junta(g2);
462         break;
463     case 2:
464         gtotal->cubo(2,2,2);g2=*gtotal;
465         x1=0.05;x2=1;y1=0.1;y2=1;
466         for (i=0;i<gtotal->nV3D;i++) {
467             xL=gtotal->v3D[i].x; yL=gtotal->v3D[i].y;
468             R=x1+(x2-x1)*xL;
469             th=pi/4*yL;
470             xF=R; yF=R*yL;
471             xF=xF*xL+(1-xL)*R*cos(th);
472             yF=yF*xL+(1-xL)*R*sin(th);
473             gtotal->v3D[i].x=xF;
474             gtotal->v3D[i].y=yF;
475         }
476         for (i=0;i<g2.nV3D;i++) {
477             xL=g2.v3D[i].x; yL=g2.v3D[i].y;
478             R=x1+(x2-x1)*yL; th=pi/4*xL;
479             yF=R; xF=R*xL;
480             xF=xF*yL+(1-yL)*R*sin(th);
481             yF=yF*yL+(1-yL)*R*cos(th);
482             g2.v3D[i].x=xF;
483             g2.v3D[i].y=yF;
484         }
485         gtotal->Junta(g2);
486 
487         g2=*gtotal;g2.Rota90Z();gtotal->Junta(g2);g2.Rota90Z();gtotal->Junta(g2);g2.Rota90Z();gtotal->Junta(g2);
488         g2=*gtotal;
489         g2.Traslada(2,0,0);gtotal->Junta(g2);
490         g2.Traslada(0,2,0);gtotal->Junta(g2);
491         g2.Traslada(-2,0,0);gtotal->Junta(g2);
492 
493 
494     }
495 
496     F.resize(gtotal->nV3D);
497     U.resize(gtotal->nV3D);
```

```
498     V.resize(gtotal->nV3D);
499     W.resize(gtotal->nV3D);
500 
501     for (i=0;i<gtotal->nV3D;i++) {
502         float x,y,z;
503         x=gtotal->v3D[i].x;
504         y=gtotal->v3D[i].y;
505         z=gtotal->v3D[i].z;
506         F[i]=sqrt(x*x+z*z);
507         U[i]=-y;
508         V[i]=x;
509         W[i]=x/20;
510     }
511     inicializacionGL();
512 }
513 
514 ///Cambiado para GLUT de () a void ResizeGraphics(int lwidth, int lheight)
515 void ResizeGraphics(int lwidth, int lheight)
516 {
517     // Get new window size
518 
519     if (DBG) cout<<"ResizeGraphics()"<<endl;
520     int tx, ty, tw, th;
521     tx=0; ty=0; tw= lwidth; th = lheight;
522 
523 #if defined(GLUI_GLUI_H)
524     if (!glui_hide) {
525         GLUI_Master.get_viewport_area( &tx, &ty, &tw, &th );
526     }
527 #endif
528     width = tw;
529     height = th;
530 
531     aspect = (GLfloat) width / height;
532 
533     // Adjust graphics to window size
534     glViewport(tx, ty, width, height);
535     glMatrixMode(GL_PROJECTION);
536     glLoadIdentity();
537     gluPerspective((GLdouble)45.0, aspect, (GLdouble)1, (GLdouble)100.0);
538     //    glTranslated( 0, 0, -10);
539     glMatrixMode(GL_MODELVIEW);
540     FuncionesOpenGL::projection_calculado=false;
541     FuncionesOpenGL::viewport_calculado=false;
542     if (DBG) cout<<"ResizeGraphics():END"<<endl;
543 }
544 
545 void CB_mouse_whell(int button, int state, int x, int y)
546 {
547 
548 //  printf("CB_mouse_whell: button=%d, state=%d, x=%d, y=%d\n",button,state,x,y);   cout<<endl;
549     switch (state)
550     {
551     case 1:
552 
553         Escala *= 1.2;
554         FuncionesOpenGL::modelview_calculado=false;
555         break;
556     case -1:
557 
558         Escala /= 1.2;
559         FuncionesOpenGL::modelview_calculado=false;
560         break;
561     }
562     return;
563 }
564 void CB_mouse(int button, int state, int x, int y)
565 {
566     xPos0 = x;
567     yPos0 = y;
568 
```

```
569     //  printf("CB_mouse: button=%d, state=%d, x=%d, y=%d\n",button,state,x,y);
570     KeyControlAltShift = glutGetModifiers();
571     //  printf("glutGetModifiers=%d\n",KeyControlAltShift); cout<<endl;
572
573
574     if (state==GLUT_UP) {iPush=0;}
575     else {
576         switch (button)
577         {
578         case GLUT_LEFT_BUTTON:
579             if (MueveCentro)
580             {
581                 //Convertir coordenadas de la pantalla a OpenGL.....
582
583
584                 //float x=xPos0,y=yPos0;
585                 GLint viewport[4];
586                 GLdouble modelview[16];
587                 GLdouble projection[16];
588                 GLfloat winX, winY, winZ;
589                 GLdouble posX, posY, posZ;
590
591                 /*
592                 glGetDoublev( GL_MODELVIEW_MATRIX, modelview );
593                 glGetDoublev( GL_PROJECTION_MATRIX, projection );
594                 glGetIntegerv( GL_VIEWPORT, viewport );
595                  */
596                 FuncionesOpenGL::ObtieneMatrices();
597
598                 winX = (float)x;
599                 winY = (float)FuncionesOpenGL::viewport[3] - (float)y;
600                 glReadPixels( x, int(winY), 1, 1, GL_DEPTH_COMPONENT, GL_FLOAT, &winZ );
601
602                 FuncionesOpenGL::Win2World(winX, winY, winZ, &posX, &posY, &posZ);
603                 //gluUnProject( winX, winY, winZ, modelview, projection, viewport, &posX, &posY, &posZ);
604
605                 printf("%f %f %f\n",posX,posY,posZ);
606                 vecDXEsfera[0]=posX-vecXEsfera[0];
607                 vecDXEsfera[1]=posY-vecXEsfera[1];
608                 vecDXEsfera[2]=posZ-vecXEsfera[2];
609                 vecXEsfera[0]=posX;
610                 vecXEsfera[1]=posY;
611                 vecXEsfera[2]=posZ;
612                 //          vecXEsfera[3]=1;
613                 MatrizXvector4(MatrizRotacionGlobal,vecDXEsfera,vecDUEsfera);
614                 vecUEsfera[0]+=Escala*vecDUEsfera[0];
615                 vecUEsfera[1]+=Escala*vecDUEsfera[1];
616                 vecUEsfera[2]+=Escala*vecDUEsfera[2];
617             }
618             else {
619                 if (MODO_Rotacion2==0) iPush=1;
620                 else iPush=3;
621             }
622
623             break;
624
625         case GLUT_RIGHT_BUTTON:
626             iPush=3;
627             break;
628         case GLUT_MIDDLE_BUTTON:
629
630             iPush=2;
631
632         }
633     }
634
635 }
636
637 void CB_motion(int x, int y)
638 {
639     //  printf("CB_motion: iPush=%d, x=%d, y=%d",iPush,x,y);cout<<endl;
```

```
640     float dx,dy;
641     GLdouble winX,winY,winZ;
642     xPos = x;
643     yPos = y;
644
645     int lMODO_de_mover=MODO_de_mover;
646
647     if (KeyControlAltShift==GLUT_ACTIVE_CTRL)
648         lMODO_de_mover=0;
649     if (iPush==2) lMODO_de_mover=3;
650     if (iPush==3) lMODO_de_mover=0;
651     switch (lMODO_de_mover)
652     {
653     case 0: //Trasladar
654         dx=(xPos-xPos0+0.0)/width*10;
655         dy=-(yPos-yPos0+0.0)/height*10;
656
657         vecUEsfera[0] +=dx;
658         vecUEsfera[1] +=dy;
659         FuncionesOpenGL::modelview_calculado=false;
660         break;
661     case 1: //Rotar
662         wAngleX=(yPos-yPos0);
663         wAngleY=(xPos-xPos0);
664         glLoadIdentity();
665         glRotatef(wAngleX, 1.0f, 0.0f, 0.0f);
666         glRotatef(wAngleY, 0.0f, 1.0f, 0.0f);
667         glMultMatrixf((GLfloat *)MatrizRotacionGlobal);
668         glGetFloatv(GL_MODELVIEW_MATRIX, (GLfloat*)MatrizRotacionGlobal);
669         glLoadIdentity();
670         glMultMatrixf((GLfloat *)MatrizRotacionGlobalINV);
671         glRotatef(-wAngleY, 0.0f, 1.0f, 0.0f);
672         glRotatef(-wAngleX, 1.0f, 0.0f, 0.0f);
673         glGetFloatv(GL_MODELVIEW_MATRIX, (GLfloat*)MatrizRotacionGlobalINV);
674         FuncionesOpenGL::modelview_calculado=false;
675         break;
676     case 2: //Spin en torno a Z
677         int dx1,dy1,x1,y1;
678         dx1=xPos-xPos0;dy1=yPos-yPos0;
679         x1=xPos0-width/2;y1=yPos0-height/2;
680
681         FuncionesOpenGL::World2Win(vecXEsfera[0],vecXEsfera[1],vecXEsfera[2], &winX, &winY, &winZ);
682         printf("xPos0=%d,winX=%f,yPos0=%d,winY=%f\n",xPos0,winX,yPos0,winY);
683         x1=xPos0-winX;y1=yPos0-(height-winY);
684
685         printf("x1=%d,y1=%d\n",x1,y1);
686
687         wAngleZ=-(dx1*(-y1)+dy1*x1+0.0)/(x1*x1+y1*y1+1.0)*180/3.14;
688         glLoadIdentity();
689         glRotatef(wAngleZ, 0.0f, 0.0f, 1.0f);
690         glMultMatrixf((GLfloat *)MatrizRotacionGlobal);
691         glGetFloatv(GL_MODELVIEW_MATRIX, (GLfloat*)MatrizRotacionGlobal);
692         glLoadIdentity();
693         glMultMatrixf((GLfloat *)MatrizRotacionGlobalINV);
694         glRotatef(-wAngleZ, 0.0f, 0.0f, 1.0f);
695         glGetFloatv(GL_MODELVIEW_MATRIX, (GLfloat*)MatrizRotacionGlobalINV);
696         FuncionesOpenGL::modelview_calculado=false;
697         break;
698
699     case 3:
700         Escala *= 1-2*(yPos-yPos0+0.0)/height;
701         FuncionesOpenGL::modelview_calculado=false;
702         break;
703
704     }
705     yPos0=yPos;
706     xPos0=xPos;
707     return;
708 }
709
710 void CB_keyboardSpecial( int key, int x, int y )
```

```
711 {
712     // manejo de teclas especiales
713     printf("CB_keyboardSpecial: char=%d, x=%d, y=%d\n",key,x,y);
714     glutPostRedisplay();
715 }
716
717
718 void idleevent()
719 {
720     static int generador=0;
721     if(DBG)cout<<"idleevent()"<<endl;
722
723     if ( glutGetWindow() != main_window )
724         glutSetWindow(main_window);
725
726
727     if(Falta_llamar_etapaS){
728         Calculo_EtapaS(0);
729     }
730     if (CalculoContinuo) {
731         if (Etapa==0){
732                 Calculo_EtapaS(); //Lanzar la primera etapa
733
734                 if (nRLC>0) nR=nRLC;
735                 if (nThLC>0) nTh=nThLC;
736
737                 time_t _tm =time(NULL );
738
739                 struct tm * curtime = localtime ( &_tm );
740                 sprintf(nombre0,"Soluciones/Sal_%02d.%02d_%02d.%02d.%02d_Malla=%d_nR=%d_nTg=%d_NDivZ=%d_",
741                         curtime->tm_mon,curtime->tm_mday,
742                         curtime->tm_hour,curtime->tm_min,curtime->tm_sec,CualMalla,nR,nTh,NDivZ);
743             }
744
745
746         Calculo_EtapaS();
747
748         char nombre[100];
749         sprintf(nombre,"%s_Size=%d_Etapa=%d.dat",nombre0,gtotal->nH3D,Etapa);
750         SaveOrRead(nombre,1);
751     } else  if (Etapa==0){
752         Calculo_EtapaS(); //Lanzar la primera etapa
753     }
754
755
756     generador++;
757     if (generador >0) {
758         generador=0;
759 //      g.GeneraCaras(g.nH3D);
760         //      g.GeneraCaras();
761
762     }
763     glutPostRedisplay();
764
765     if(DBG)cout<<"idleevent():END"<<endl;
766 }
767
768 void LecturaArchivoDeDatos() {
769     ifstream myfile;
770     ofstream myfile2;
771     if(DBG) {
772         myfile2.open("dondeestoy.txt");
773         myfile2<<"hola"<<endl;
774         myfile2.close();
775     }
776     myfile.open (FDatos);
777     string linea;
778
779     cout<<"LecturaArchivoDeDatos():"<<endl;
780     cout<<"FDatos="<<FDatos<<endl;
781
```

```
782     while (getline(myfile,linea)) {
783         if(DBG) cout<<"linea="<<linea<<endl;
784         istringstream trozo( linea );
785         string snombre,svalor;
786         if (linea[0]=='#') continue;
787         if (!getline( trozo, snombre, '=' )) continue;
788         if (!getline( trozo, svalor, '\n' )) continue;
789         double ff=atof(svalor.c_str());
790 
791         if      (snombre=="Dominio_Rmax"    ) {Dominio_Rmax     =ff;}
792         else if (snombre=="Dominio_Hmax"    ) {Dominio_Hmax     =ff;}
793         else if (snombre=="Dominio_Hsup"    ) {Dominio_Hsup     =ff;}
794         else if (snombre=="rhof"            ) {Datos_rhof               =ff;}
795         else if (snombre=="rhos"            ) {Datos_rhos               =ff;}
796         else if (snombre=="phi"             ) {Datos_phi                =ff;}
797         else if (snombre=="cf"              ) {Datos_cf             =ff;}
798         else if (snombre=="Tinyeccion"      ) {Datos_Tinyeccion     =ff;}
799         else if (snombre=="Tambiente"       ) {Datos_Tambiente      =ff;}
800         else if (snombre=="hc"              ) {Datos_hc             =ff;}
801         else if (snombre=="KTermofilm"      ) {Datos_KTermofilm     =ff;}
802         else if (snombre=="eTermofilm"      ) {Datos_eTermofilm     =ff;}
803         else if (snombre=="VinyeccionLtsHr" ) {VinyeccionLtsHr  =ff;}
804         else if (snombre=="kf"              ) {Datos_kf             =ff;}
805         else if (snombre=="ks"              ) {Datos_ks             =ff;}
806         else {
807             cout <<"snombre="<< snombre<<"\t svalor="<<svalor<<"\t ff="<<ff<<endl;
808         }
809 
810     }
811 
812     Datos_rho=Datos_rhof;    //TODO buscar
813     Datos_KtEt=Datos_KTermofilm/Datos_eTermofilm;
814     Datos_htilde=(Datos_hc*Datos_KtEt)/(Datos_hc+Datos_KtEt);
815     Datos_Vinyeccion=VinyeccionLtsHr/1000/3600; //% m^3/m^2/s
816     Datos_km=Datos_phi*Datos_kf+(1-Datos_phi)*Datos_ks;
817 
818 
819 
820     K_difusion=1/Dominio_Hsup; //TODO buscar donde se usa
821 
822     myfile.close();
823 
824 }
825 void SaveOrRead(char *ifile_name,int iSaveReadMode) {
826 
827 if (iSaveReadMode==1) { //Save
828 
829       ofstream myfile;
830       myfile.open (ifile_name);
831 
832       myfile.precision(17);
833 
834       gtotal->save(myfile);
835       myfile<<Etapa<<endl;
836       save(F,myfile);
837       save(FF,myfile);
838       save(FF2,myfile);
839       save(F2Nodos,myfile);
840       save(PotencialVInfiltracion,myfile);
841       save(TempPilaBloques,myfile);
842       save(U,myfile);
843       save(V,myfile);
844       save(W,myfile);
845       myfile.close();
846 } else {
847     char leyendo[100];
848     char leyendo2[100];
849     int num1,num2;
850     sscanf(ifile_name,"%[A-Za-z_\.0-9]=%d%s",leyendo,&num1,leyendo2);
851     cout <<"ifile_name="<<ifile_name<<endl;
852     cout <<"leyendo="<<leyendo<<endl;
```

```
853     cout <<"num1="<<num1<<endl;
854 //  cout <<"num2="<<num1<<endl;
855     cout <<"leyendo2="<<leyendo2<<endl;
856 
857 
858     ifstream myfile;
859       myfile.open (ifile_name);
860       gtotal->read(myfile);
861       cout<<"Malla Ok"<<endl;
862       myfile>>Etapa;
863       read(F,myfile);
864       cout<<"F Ok"<<endl;
865       read(FF,myfile);
866       cout<<"FF Ok"<<endl;
867       read(FF2,myfile);
868       cout<<"FF2 Ok"<<endl;
869       read(F2Nodos,myfile);
870       cout<<"F2Nodos Ok"<<endl;
871       read(PotencialVInfiltracion,myfile);
872       cout<<"PotencialVInfiltracion Ok"<<endl;
873       read(TempPilaBloques,myfile);
874       cout<<"TempPilaBloques Ok"<<endl;
875       read(U,myfile);
876       cout<<"U Ok"<<endl;
877       read(V,myfile);
878       cout<<"V Ok"<<endl;
879       read(W,myfile);
880       cout<<"W Ok"<<endl;
881       myfile.close();
882 
883         CualMalla=num1;
884         InicioEtapa=Etapa;
885         err0=1e-8;
886         switch (CualMalla) {
887         case 1:
888             ThetaMax=atan(1)*2;
889             if (Etapa==7) InicioEtapa=Etapa-1;
890             break;
891         }
892 }
893 }
894 
895 
896 void visible(int vis)
897 {
898     cout<<"visible()"<<endl;
899 
900     if (vis == GLUT_VISIBLE) {
901 
902         glutIdleFunc(idleevent);
903 
904     } else {
905         glutIdleFunc(NULL);
906     }
907     cout<<"visible():END"<<endl;
908 }
909 
910 double CalculaLaplacianoCero(vector<double> &F,vector<double> &U,vector<double> &V,
911          vector<double> &W,grid3D g,int niteraciones,double err0)
912 {
913     int iiter,i,j,iC;
914     double SumaCoeffDif_F,SumaCoeffDif,Coeff_Difusion,err,errG,nF,SumaVolumen;
915     if (F.size() ==0) F.resize(g.nH3D);
916     //Resuelvo la ecuación varias veces
917     niteraciones=g.nH3D*50;
918     if (niteraciones<30000) niteraciones=30000;
919     cout <<"niteraciones="<<niteraciones<<endl;
920     for (iiter=0 ; iiter<niteraciones ; iiter++) {
921         errG=0;
922         int i0=g.nH3D-1,istep=-1;
923         for (i=i0 ; i<g.nH3D && i>= 0 ; i+=istep) {
```

```
924             //Escribo la ecuación para cada hexahedro [i]
925             SumaCoeffDif_F=0;SumaCoeffDif=0;SumaVolumen=0;
926             for (j=0 ; j< g.h3D[i].vecino.size() ;j++) {
927                 //recorro cada vecino==cara [i]--[j]
928 
929                 Coeff_Difusion = K_difusion*g.h3D[i].Poligono[j].AreaPoligono / g.h3D[i].Poligono[j].Dab;
930                 SumaVolumen += g.h3D[i].Poligono[j].AreaPoligono * g.h3D[i].Poligono[j].Dab/6;
931                 if (g.h3D[i].tipo_vecino[j] == ES_BLOQUE) {
932                     SumaCoeffDif_F += F[ g.h3D[i].vecino[j] ] * Coeff_Difusion;
933                     SumaCoeffDif += Coeff_Difusion;
934                 } else {
935                     iC=g.h3D[i].vecino[j];
936                     if (g.Cara[iC].iBC ==2 ||g.Cara[iC].iBC ==3 ) { //1== Derivada normal igual a cero
937                         //                      if (iiter==0 ) cout<<"g.Cara[iC].iBC="<<g.Cara[iC].iBC<<endl;
938                         SumaCoeffDif_F += g.Cara[iC].BC * Coeff_Difusion;
939                         SumaCoeffDif += Coeff_Difusion;
940                     }
941                     if (g.Cara[iC].iBC ==10 ) {
942                         SumaCoeffDif_F -= Datos_rho*Datos_Vinyeccion*g.h3D[i].Poligono[j].AreaPoligono;
943                     }
944 
945                 }
946             }
947             nF = SumaCoeffDif_F/SumaCoeffDif;
948             err=fabs(nF-F[i])/(fabs(nF)+1e-20);
949             if (err > errG) errG=err;
950             F[i]=nF;
951             //          if (F[i]==0) F[i]=0.5;
952 
953         }
954         if (errG<err0) {
955             cout<<"break: errG<"<<err0<<", iiter="<<iiter<<endl;
956             break;
957         }
958     }
959     for (i=0 ; i<g.nH3D ; i++) {
960         SumaCoeffDif_F=0;SumaCoeffDif=0;
961         double A,AA,AB,AC,RA;
962         double BA,BB,BC,RB;
963         double CA,CB,CC,RC;
964         double nx,ny,nz,DET;
965         AA=0;AB=0;AC=0;RA=0;
966         BA=0;BB=0;BC=0;RB=0;
967         CA=0;CB=0;CC=0;RC=0;
968         for (j=0 ; j< g.h3D[i].vecino.size() ;j++) {
969             if (g.h3D[i].tipo_vecino[j] == ES_BLOQUE) {
970                 Coeff_Difusion = ( F[ g.h3D[i].vecino[j] ] - F[i] ) / g.h3D[i].Poligono[j].Dab;
971             } else {
972                 iC=g.h3D[i].vecino[j];
973                 Coeff_Difusion=0;
974                 if (g.Cara[iC].iBC ==2 ||g.Cara[iC].iBC ==3) {
975                     Coeff_Difusion = ( g.Cara[iC].BC - F[i] ) / g.h3D[i].Poligono[j].Dab;
976 
977                 }
978                 if (g.Cara[iC].iBC ==10) {
979                     Coeff_Difusion = -Datos_rho*Datos_Vinyeccion*Dominio_Hsup;
980 
981                 }
982                 if (g.Cara[iC].iBC ==110) {
983                     Coeff_Difusion = Datos_rho*Datos_Vinyeccion*Dominio_Hsup;
984 
985                 }
986             }
987             A=g.h3D[i].Poligono[j].AreaPoligono;
988             nx=g.h3D[i].Poligono[j].normal.x ;
989             ny=g.h3D[i].Poligono[j].normal.y ;
990             nz=g.h3D[i].Poligono[j].normal.z ;
991             AA += nx*nx*A; AB+= nx*ny*A; AC+= nx*nz*A; RA += -nx*Coeff_Difusion*A;
992             BA += ny*nx*A; BB+= ny*ny*A; BC+= ny*nz*A; RB += -ny*Coeff_Difusion*A;
993             CA += nz*nx*A; CB+= nz*ny*A; CC+= nz*nz*A; RC += -nz*Coeff_Difusion*A;
994         }
```

```
 995         // AA AB AC --> RA
 996         // BA BB BC --> RB
 997         // CA CB CC --> RC
 998         DET= AA*(BB*CC-BC*CB)+BA*(CB*AC-AB*CC)+CA*(AB*BC-BB*AC) ;
 999         U[i]= ( RA*(BB*CC-BC*CB)+RB*(CB*AC-AB*CC)+RC*(AB*BC-BB*AC) )/DET;
1000         V[i]=-( RA*(BA*CC-BC*CA)+RB*(CA*AC-AA*CC)+RC*(AA*BC-BA*AC) )/DET;
1001         W[i]= ( RA*(BA*CB-BB*CA)+RB*(CA*AB-AA*CB)+RC*(AA*BB-BA*AB) )/DET;
1002     }
1003     return(errG);
1004 }
1005
1006
1007 double CalculaLaplacianoCero2(vector<double> &F2,vector<double> &PotencialV,vector<double> &U,
1008         vector<double> &V,vector<double> &W,grid3D g,int niteraciones,double err0)
1009 {
1010     int iiter,i,j,iC;
1011     double SumaCoeffDif_F2,SumaCoeffDif,Coeff_Difusion,err,errG,nF,SumaVolumen;
1012     if (F2.size() ==0) F2.resize(g.nH3D);
1013     //Resuelvo la ecuación varias veces
1014     cout <<"niteraciones="<<niteraciones<<endl;
1015     for (iiter=0 ; iiter<niteraciones*0.5 ; iiter++) {
1016         errG=0;
1017         for (i=0 ; i<g.nH3D ; i++) {
1018             //Escribo la ecuación para cada hexahedro [i]
1019             SumaCoeffDif_F2=0;SumaCoeffDif=0;SumaVolumen=0;
1020             double PotencialV_i=PotencialV[i],PotencialV_j;
1021             for (j=0 ; j< g.h3D[i].vecino.size() ;j++) {
1022                 if(1==0) cout <<"CalculaLaplacianoCero2: iiter,i,j="<<iiter<<" "<<i<<" "<<j
1023                         << "\tPotencialV.size=" << PotencialV.size()
1024                         <<"\tg.h3D[i].vecino[j]="<<g.h3D[i].vecino[j]
1025                         <<"\tg.h3D[i].tipo_vecino[j]="<<g.h3D[i].tipo_vecino[j]
1026                         <<endl;
1027                 //recorro cada vecino==cara [i]--[j]
1028
1029                 if (g.h3D[i].tipo_vecino[j] == ES_BLOQUE) {
1030                     PotencialV_j=PotencialV[ g.h3D[i].vecino[j] ];
1031                     if (PotencialV_j>PotencialV_i) Coeff_Difusion = K_difusion*g.h3D[i].Poligono[j].AreaPoligono /
    g.h3D[i].Poligono[j].Dab
1032                             *(PotencialV_j-PotencialV_i)*Datos_cf;
1033                     else Coeff_Difusion=0;
1034                     SumaCoeffDif_F2 += F2[ g.h3D[i].vecino[j] ] * Coeff_Difusion;
1035                     SumaCoeffDif += Coeff_Difusion;
1036                 } else {
1037                     iC=g.h3D[i].vecino[j];
1038                     if (g.Cara[iC].iBC ==2 ||g.Cara[iC].iBC ==3 ) { //1== Derivada normal igual a cero
1039                         PotencialV_j=g.Cara[iC].BC;
1040                         if (PotencialV_j>PotencialV_i) Coeff_Difusion = K_difusion*g.h3D[i].Poligono[j].AreaPoligono
    / g.h3D[i].Poligono[j].Dab
1041                                 *(PotencialV_j-PotencialV_i)*Datos_cf;
1042
1043                         //                      if (iiter==0 ) cout<<"g.Cara[iC].iBC="<<g.Cara[iC].iBC<<endl;
1044                         if (iiter==-10 ) cout<<"g.Cara[iC].BC2="<<g.Cara[iC].BC2
1045                                 <<"\t.BC="<<g.Cara[iC].BC
1046                                 <<"\tg.Cara[iC].iBC="<<g.Cara[iC].iBC
1047                                 <<"\tiC="<<iC
1048                                 <<"\tCoeff_Difusion="<<Coeff_Difusion
1049                                 <<"\tPotencialV_i="<<PotencialV_i<<"PotencialV_j="<<PotencialV_j
1050                                 <<endl;
1051                         SumaCoeffDif_F2 += g.Cara[iC].BC2 * Coeff_Difusion;
1052                         SumaCoeffDif += Coeff_Difusion;
1053                     }
1054                     if (g.Cara[iC].iBC ==11 ) {
1055                         Coeff_Difusion=Datos_htilde*g.h3D[i].Poligono[j].AreaPoligono;;
1056                         SumaCoeffDif_F2 += Coeff_Difusion*Datos_Tambiente;
1057                         SumaCoeffDif += Coeff_Difusion;
1058                     }
1059
1060                 }
1061             }
1062             nF = SumaCoeffDif_F2/SumaCoeffDif;
1063             err=fabs(nF-F2[i])/(fabs(nF)+1e-20);
```

```
1064            if (err > errG) errG=err;
1065            F2[i]=nF;
1066            //          if (F[i]==0) F[i]=0.5;
1067        }
1068        if (errG<err0) {
1069            cout<<"break: errG<"<<err0<<", iiter="<<iiter<<endl;
1070            break;
1071        }
1072    }
1073    for (i=0 ; i<g.nH3D ; i++) {
1074        double A,AA,AB,AC,RA;
1075        double BA,BB,BC,RB;
1076        double CA,CB,CC,RC;
1077        double nx,ny,nz,DET;
1078        AA=0;AB=0;AC=0;RA=0;
1079        BA=0;BB=0;BC=0;RB=0;
1080        CA=0;CB=0;CC=0;RC=0;
1081        for (j=0 ; j< g.h3D[i].vecino.size() ;j++) {
1082            if (g.h3D[i].tipo_vecino[j] == ES_BLOQUE) {
1083                Coeff_Difusion = ( F2[ g.h3D[i].vecino[j] ] - F2[i] ) / g.h3D[i].Poligono[j].Dab;
1084            } else {
1085                iC=g.h3D[i].vecino[j];
1086                Coeff_Difusion=0;
1087                if (g.Cara[iC].iBC ==2 ||g.Cara[iC].iBC ==3) {
1088                    Coeff_Difusion = ( g.Cara[iC].BC2 - F2[i] ) / g.h3D[i].Poligono[j].Dab;
1089
1090                }
1091            }
1092            A=g.h3D[i].Poligono[j].AreaPoligono;
1093            nx=g.h3D[i].Poligono[j].normal.x ;
1094            ny=g.h3D[i].Poligono[j].normal.y ;
1095            nz=g.h3D[i].Poligono[j].normal.z ;
1096            AA += nx*nx*A; AB+= nx*ny*A; AC+= nx*nz*A; RA += -nx*Coeff_Difusion*A;
1097            BA += ny*nx*A; BB+= ny*ny*A; BC+= ny*nz*A; RB += -ny*Coeff_Difusion*A;
1098            CA += nz*nx*A; CB+= nz*ny*A; CC+= nz*nz*A; RC += -nz*Coeff_Difusion*A;
1099        }
1100        // AA AB AC --> RA
1101        // BA BB BC --> RB
1102        // CA CB CC --> RC
1103        DET= AA*(BB*CC-BC*CB)+BA*(CB*AC-AB*CC)+CA*(AB*BC-BB*AC) ;
1104        U[i]= ( RA*(BB*CC-BC*CB)+RB*(CB*AC-AB*CC)+RC*(AB*BC-BB*AC) )/DET;
1105        V[i]=-( RA*(BA*CC-BC*CA)+RB*(CA*AC-AA*CC)+RC*(AA*BC-BA*AC) )/DET;
1106        W[i]= ( RA*(BA*CB-BB*CA)+RB*(CA*AB-AA*CB)+RC*(AA*BB-BA*AB) )/DET;
1107    }
1108    return(errG);
1109 }
1110
1111
1112 double CalculaTemperaturaEquilibrioPila(vector<double> &F2,vector<double> &PotencialV,vector<double> &U,
1113         vector<double> &V,vector<double> &W,grid3D g,int niteraciones,double err0)
1114 {
1115     int iiter,i,j,iC;
1116     double SumaCoeffDif_F2,SumaCoeffDif,Coeff_Difusion,err,errG,nF,SumaVolumen;
1117     if (F2.size() ==0) F2.resize(g.nH3D);
1118     //Resuelvo la ecuación varias veces
1119     cout <<"niteraciones="<<niteraciones<<endl;
1120     for (iiter=0 ; iiter<niteraciones*5 ; iiter++) {
1121         errG=0;
1122         for (i=0 ; i<g.nH3D ; i++) {
1123             //Escribo la ecuación para cada hexahedro [i]
1124             SumaCoeffDif_F2=0;SumaCoeffDif=0;SumaVolumen=0;
1125             double PotencialV_i=PotencialV[i],PotencialV_j;
1126             for (j=0 ; j< g.h3D[i].vecino.size() ;j++) {
1127                 if(1==0) cout <<"CalculaLaplacianoCero2: iiter,i,j="<<iiter<<" "<<i<<" "<<j
1128                         << "\tPotencialV_.size=" << PotencialV.size()
1129                         <<"\tg.h3D[i].vecino[j]="<<g.h3D[i].vecino[j]
1130                         <<"\tg.h3D[i].tipo_vecino[j]="<<g.h3D[i].tipo_vecino[j]
1131                         <<endl;
1132                 //recorro cada vecino==cara [i]--[j]
1133
1134                 if (g.h3D[i].tipo_vecino[j] == ES_BLOQUE) {
```

```
1135                    PotencialV_j=PotencialV[ g.h3D[i].vecino[j] ];
1136                    if (PotencialV_j>PotencialV_i)
1137                        Coeff_Difusion = Datos_rhof * Datos_cf *(PotencialV_j-PotencialV_i) *
1138                        g.h3D[i].Poligono[j].AreaPoligono / g.h3D[i].Poligono[j].Dab;
1139                    else Coeff_Difusion=0;
1140
1141                    Coeff_Difusion += Datos_km * g.h3D[i].Poligono[j].AreaPoligono / g.h3D[i].Poligono[j].Dab;
1142
1143                    SumaCoeffDif_F2 += F2[ g.h3D[i].vecino[j] ] * Coeff_Difusion;
1144                    SumaCoeffDif += Coeff_Difusion;
1145                } else {
1146                    iC=g.h3D[i].vecino[j];
1147                    if (g.Cara[iC].iBC ==2 ||g.Cara[iC].iBC ==3 ) { //1== Derivada normal igual a cero
1148                        PotencialV_j=g.Cara[iC].BC;
1149                        if (PotencialV_j>PotencialV_i)
1150                            Coeff_Difusion = Datos_rhof * Datos_cf *(PotencialV_j-PotencialV_i) *
1151                            g.h3D[i].Poligono[j].AreaPoligono / g.h3D[i].Poligono[j].Dab;
1152                        else Coeff_Difusion=0;
1153
1154                        Coeff_Difusion += Datos_km * g.h3D[i].Poligono[j].AreaPoligono / g.h3D[i].Poligono[j].Dab;
1155
1156                        //if (iiter==0 ) cout<<"g.Cara[iC].iBC="<<g.Cara[iC].iBC<<endl;
1157                        if (iiter==-10 ) cout<<"g.Cara[iC].BC2="<<g.Cara[iC].BC2
1158                                <<"\t.BC="<<g.Cara[iC].BC
1159                                <<"\tg.Cara[iC].iBC="<<g.Cara[iC].iBC
1160                                <<"\tiC="<<iC
1161                                <<"\tCoeff_Difusion="<<Coeff_Difusion
1162                                <<"\tPotencialV_i="<<PotencialV_i<<"PotencialV_j="<<PotencialV_j
1163                                <<endl;
1164                        SumaCoeffDif_F2 += g.Cara[iC].BC2 * Coeff_Difusion;
1165                        SumaCoeffDif += Coeff_Difusion;
1166                    }
1167                    if (g.Cara[iC].iBC ==11 ) {
1168                        Coeff_Difusion=Datos_htilde*g.h3D[i].Poligono[j].AreaPoligono;;
1169                        SumaCoeffDif_F2 += Coeff_Difusion*Datos_Tambiente;
1170                        SumaCoeffDif += Coeff_Difusion;
1171                    }
1172
1173                }
1174            }
1175            nF = SumaCoeffDif_F2/SumaCoeffDif;
1176            err=fabs(nF-F2[i])/(fabs(nF)+1e-20);
1177            if (err > errG) errG=err;
1178            F2[i]=nF;
1179            //          if (F[i]==0) F[i]=0.5;
1180        }
1181        if (errG<err0) {
1182            cout<<"break: errG<"<<err0<<", iiter="<<iiter<<endl;
1183            break;
1184        }
1185    }
1186    for (i=0 ; i<g.nH3D ; i++) {
1187        double A,AA,AB,AC,RA;
1188        double BA,BB,BC,RB;
1189        double CA,CB,CC,RC;
1190        double nx,ny,nz,DET;
1191        AA=0;AB=0;AC=0;RA=0;
1192        BA=0;BB=0;BC=0;RB=0;
1193        CA=0;CB=0;CC=0;RC=0;
1194        for (j=0 ; j< g.h3D[i].vecino.size() ;j++) {
1195            if (g.h3D[i].tipo_vecino[j] == ES_BLOQUE) {
1196                Coeff_Difusion = ( F2[ g.h3D[i].vecino[j] ] - F2[i] ) / g.h3D[i].Poligono[j].Dab;
1197            } else {
1198                iC=g.h3D[i].vecino[j];
1199                Coeff_Difusion=0;
1200                if (g.Cara[iC].iBC ==2 ||g.Cara[iC].iBC ==3) {
1201                    Coeff_Difusion = ( g.Cara[iC].BC2 - F2[i] ) / g.h3D[i].Poligono[j].Dab;
1202
1203                }
1204            }
1205            A=g.h3D[i].Poligono[j].AreaPoligono;
```

```
1206            nx=g.h3D[i].Poligono[j].normal.x ;
1207            ny=g.h3D[i].Poligono[j].normal.y ;
1208            nz=g.h3D[i].Poligono[j].normal.z ;
1209            AA += nx*nx*A; AB+= nx*ny*A; AC+= nx*nz*A; RA += -nx*Coeff_Difusion*A;
1210            BA += ny*nx*A; BB+= ny*ny*A; BC+= ny*nz*A; RB += -ny*Coeff_Difusion*A;
1211            CA += nz*nx*A; CB+= nz*ny*A; CC+= nz*nz*A; RC += -nz*Coeff_Difusion*A;
1212        }
1213        // AA AB AC --> RA
1214        // BA BB BC --> RB
1215        // CA CB CC --> RC
1216        DET= AA*(BB*CC-BC*CB)+BA*(CB*AC-AB*CC)+CA*(AB*BC-BB*AC) ;
1217        U[i]= ( RA*(BB*CC-BC*CB)+RB*(CB*AC-AB*CC)+RC*(AB*BC-BB*AC) )/DET;
1218        V[i]=-( RA*(BA*CC-BC*CA)+RB*(CA*AC-AA*CC)+RC*(AA*BC-BA*AC) )/DET;
1219        W[i]= ( RA*(BA*CB-BB*CA)+RB*(CA*AB-AA*CB)+RC*(AA*BB-BA*AB) )/DET;
1220    }
1221    return(errG);
1222 }
1223
1224
1225 void CB_keyboard(unsigned char key, int x, int y)
1226 {
1227
1228    cout<<"CB_keyboard()"<<endl;
1229    double gLeft=-.1,gRight=.1, gBottom=-.1, gTop=.1, gNear=0, gFar=1000000;
1230    printf("CB_keyboard: char=%d(%c), x=%d, y=%d",key,key,x,y);
1231    cout<<endl;
1232
1233    switch(key) {
1234    case 27: /* ESC */
1235        exit(0);
1236        break;
1237
1238    case 32: /* Espacio */
1239        Calculo_EtapaS();
1240        break;
1241    case 8: /* BackSpace */
1242        Etapa=InicioEtapa-1;
1243        Calculo_EtapaS();
1244        break;
1245    case '1':
1246        Calculo_EtapaS(true);
1247        break;
1248    case 'v':
1249    case 'V':
1250        gluiHelp->set_int_val(!MODO_MenuMENSAJES);MODO_MenuMENSAJES=gluiHelp->get_int_val();
1251        break;
1252    case 'n':
1253    case 'N':
1254        gluiNormales->set_int_val(!gluiNormales->int_val);
1255        break;
1256    case 'b':
1257    case 'B':
1258        gluiBordes->set_int_val(!ModoDibujaFrontera);ModoDibujaFrontera=gluiBordes->get_int_val();
1259        break;
1260    case 'i':
1261    case 'I':
1262        gluiInterior->set_int_val(!ModoDibujaInterior);ModoDibujaInterior=gluiInterior->get_int_val();
1263        break;
1264
1265
1266 #if defined(GLUI_GLUI_H)
1267    case 'Z':
1268    case 'z':
1269        if (glui_hide)
1270            glui->show();
1271        else
1272            glui->hide();
1273        glui_hide=!glui_hide;
1274        ResizeGraphics(glutGet(GLUT_WINDOW_WIDTH),glutGet(GLUT_WINDOW_HEIGHT));
1275        break;
1276 #endif
```

```
1277    case 'T':
1278    case 't':
1279        glui_GrupoModoDelMouse->set_int_val(0);MODO_de_mover=glui_GrupoModoDelMouse->get_int_val();
1280        break;
1281    case 'R':
1282    case 'r':
1283
1284        //      glLoadIdentity();
1285        //      glOrtho(gLeft, gRight, gBottom, gTop, gFar, -gFar);
1286        //      glGetFloatv( GL_MODELVIEW_MATRIX, (GLfloat*)MatrizRotacionGlobal );
1287        //      TrasponeMatriz();
1288
1289        glui_GrupoModoDelMouse->set_int_val(1);MODO_de_mover=glui_GrupoModoDelMouse->get_int_val();
1290        break;
1291    case 'S':
1292    case 's':
1293        glui_GrupoModoDelMouse->set_int_val(2);MODO_de_mover=glui_GrupoModoDelMouse->get_int_val();
1294        break;
1295    case 'E':
1296    case 'e':
1297        glui_GrupoModoDelMouse->set_int_val(3);MODO_de_mover=glui_GrupoModoDelMouse->get_int_val();
1298        break;
1299    case '+':
1300        if (NumON==1 ) {
1301            NumEscala /=1.2;
1302        } else {
1303            char s[100];
1304            npasadas*=2;
1305            if (npasadas>maxpasadas)    npasadas=maxpasadas;
1306            sprintf(s,"(+)=npasadas++=%d",npasadas);
1307            AddMensaje(s);
1308        }
1309        break;
1310    case '-':
1311        if (NumON==1 ) {
1312            NumEscala *=1.2;
1313        } else {
1314            char s[100];
1315            npasadas/=2;
1316            if (npasadas<=0)                npasadas=1;
1317            sprintf(s,"(-)=npasadas--=%d",npasadas);
1318            AddMensaje(s);
1319        }
1320        break;
1321    case 'M':
1322    case 'm':
1323        gluiMueve->set_int_val(!MueveCentro);MueveCentro=gluiMueve->get_int_val();
1324        if (MueveCentro)    AddMensaje((char *)"M=Mueve Centro");
1325        else                AddMensaje("..end: Mueve centro");
1326        break;
1327    case 'P':
1328    case 'p':
1329        gluiClipping->set_int_val(!ClippingON);ClippingON=gluiClipping->get_int_val();
1330        /*
1331        ClippingON=1-ClippingON;
1332         */
1333        if (ClippingON)    AddMensaje("P= Cli(P)ping on..");
1334        else                AddMensaje("P= Cli(P)ping off..");
1335        break;
1336    case 'C':
1337    case 'c':
1338        cout<<"case 'C': ColorON="<<ColorON<<"-->";
1339        ColorON++;
1340        if (ColorON==1 & F.size()!=gtotal->nV3D) ColorON++;
1341        if (ColorON==2 & F2Nodos.size()!=gtotal->nV3D) ColorON++;
1342
1343
1344        if (ColorON>3)
1345            ColorON=0;
1346        if (ColorON)    AddMensaje("C= Color on..");
1347        else                AddMensaje("C= Color off..");
```

```
1348        cout<<ColorON<<endl;
1349
1350        break;
1351
1352    case 'F':
1353    case 'f':
1354        fullscreen = !fullscreen;
1355        if (fullscreen) {
1356            old_x = glutGet(GLUT_WINDOW_X);
1357            old_y = glutGet(GLUT_WINDOW_Y);
1358            old_width = glutGet(GLUT_WINDOW_WIDTH);
1359            old_height = glutGet(GLUT_WINDOW_HEIGHT);
1360            glutFullScreen();
1361        } else {
1362            glutReshapeWindow(old_width, old_height);
1363            glutPositionWindow(old_x, old_y);
1364        }
1365        break;
1366
1367    case 'G':
1368    case 'g':
1369        ModoGame++;if (ModoGame>3) ModoGame=0;
1370        printf("ModoGame=%d\n",ModoGame);
1371        switch (ModoGame)
1372        {
1373        case 1:
1374        case 2:
1375        case 3:
1376            if (ModoGame==1) glutGameModeString("1280x1024:32@60");
1377            if (ModoGame==2) glutGameModeString("800x600:16@60");
1378            if (ModoGame==3) glutGameModeString("640x480:16@60");
1379            glutEnterGameMode();
1380
1381            main_window = glutGetWindow();
1382            //      initWindow();
1383            glutDisplayFunc(DrawGraphics);
1384            glutReshapeFunc(ResizeGraphics);
1385            glutMouseFunc(CB_mouse);
1386            glutMotionFunc(CB_motion);
1387            glutKeyboardFunc(CB_keyboard);
1388            glutSpecialFunc( CB_keyboardSpecial );
1389
1390            glutIdleFunc( idleevent );
1391            glutVisibilityFunc(visible);
1392            inicializacionGL();
1393            DrawGraphics();
1394
1395            break;
1396        case 0:
1397
1398            glutLeaveGameMode();
1399            main_window = glutGetWindow();
1400            glutDisplayFunc(DrawGraphics);
1401            glutReshapeFunc(ResizeGraphics);
1402            glutMouseFunc(CB_mouse);
1403            glutMotionFunc(CB_motion);
1404            glutKeyboardFunc(CB_keyboard);
1405            glutSpecialFunc( CB_keyboardSpecial );
1406
1407            glutIdleFunc( idleevent );
1408            glutVisibilityFunc(visible);
1409            inicializacionGL();
1410            DrawGraphics();
1411            break;
1412        }
1413    }
1414    cout<<"CB_keyboard():END"<<endl;
1415 }
1416
1417
```