Examen-practico-resuelto.pdf

PruebaAlien

Informática Gráfica

3º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

Axis.cc

```
#include "axis.h"

using namespace _colors_ne;


/*****************************************************************************//**
 *
 *
 *
 *****************************************************************************/

_axis::_axis(float Size)
{
  Vertices.resize(5);

  Vertices[0]=_vertex3f(-Size,0,0);
  Vertices[1]=_vertex3f(Size,0,0);

  Vertices[2]=_vertex3f(0,-Size,0);
  Vertices[3]=_vertex3f(0,Size,0);

  Vertices[4]=_vertex3f(0,0,-Size);
  Vertices[5]=_vertex3f(0,0,Size);
}


/*****************************************************************************//**
 *
 *
 *
 *****************************************************************************/

void _axis::draw_line()
{
  glLineWidth(1);
  glBegin(GL_LINES);
  // eje X, color rojo
  glColor3fv((GLfloat *) &RED);
  glVertex3fv((GLfloat *) &Vertices[0]);
  glVertex3fv((GLfloat *) &Vertices[1]);
  // eje Y, color verde
  glColor3fv((GLfloat *) &GREEN);
  glVertex3fv((GLfloat *) &Vertices[2]);
  glVertex3fv((GLfloat *) &Vertices[3]);
  // eje Z, color azul
  glColor3fv((GLfloat *) &BLUE);
  glVertex3fv((GLfloat *) &Vertices[4]);
  glVertex3fv((GLfloat *) &Vertices[5]);
  glEnd();
}
```

Basic_object3D.cc

```
#include "basic_object3d.h"


/*****************************************************************************//**
 *
 *
 *
 *****************************************************************************/

void _basic_object3D::draw_point()
{
  glBegin(GL_POINTS);
  for (unsigned int i=0;i<Vertices.size();i++){
    glVertex3fv((GLfloat *) &Vertices[i]);
  }
  glEnd();
}
```

Brazo1.cc

```
#include "brazo1.h"

_Brazo1::_Brazo1(){}
void _Brazo1::draw(int modo, int mover){
   glMatrixMode(GL_MODELVIEW);

   glPushMatrix();
   glTranslated(5+mover,0,0);
   glRotatef(90,0,0,1);
   glScalef(1,10,1);
   if(modo == 1)
      bra.draw_point();
   if(modo == 2)
      bra.draw_line();
   if(modo == 3)
      bra.draw_fill();
   if(modo == 4)
      bra.draw_chess();
   glPopMatrix();
}
```

Brazo1.h

```
#ifndef BRAZO1_H
#define BRAZO1_H
#include "cube.h"

class _Brazo1{
private:
```

KEEP CALM AND ESTUDIA UN POQUITO

WUOLAH + #QuédateEnCasa

```
   _cube bra;

   //tamaño del brazo
   float size=1;
public:
   _Brazo1();
   void draw(int modo, int mover);//dibuja
};

#endif
```

Brazo2.cc

```
#include "brazo2.h"

_Brazo2::_Brazo2(){}
void _Brazo2::draw(int modo, int mover){
   glMatrixMode(GL_MODELVIEW);

   glPushMatrix();
   glTranslatef(7.5,0,0);
   glRotatef(90,0,0,1);
   glScalef(1.5,15,1.5);
   if(modo == 1)
      bra.draw_point();
   if(modo == 2)
      bra.draw_line();
   if(modo == 3)
      bra.draw_fill();
   if(modo == 4)
      bra.draw_chess();

   glPopMatrix();
   glMatrixMode(GL_MODELVIEW);

   glPushMatrix();
   glTranslatef(7.5,0,0);
   bra1.draw(modo,mover);
   glPopMatrix();
}
```

Brazo2.h

```
#ifndef BRAZO2_H
#define BRAZO2_H
#include "object3dr.h"
#include "cube.h"
#include "brazo1.h"

class _Brazo2:public _object3Dr{
private:
```

```
   _cube bra;
   _Brazo1 bra1;

   //tamaño del brazo
   float size=1;
public:
   _Brazo2();
   void draw(int modo, int mover);//dibuja
};

#endif
```

Cilindro.cc

```
#include "cilindro.h"


/**************************************************************************************//**
 *
 *
 *
 **************************************************************************************/

_cilindro::_cilindro(float Size)
{
  Vertices.resize(4);
    //perfil (OJO TIENE QUE SEGUIR UN ORDEN ANTI ORARIO)
    //P0, P1, P2, P3
  Vertices[0]=_vertex3f(0,-Size/2,0);
  Vertices[1]=_vertex3f(Size/2,-Size/2,0);
  Vertices[2]=_vertex3f(Size/2,Size/2,0);
  Vertices[3]=_vertex3f(0,Size/2,0);

  //eje x
   /* Vertices[0]=_vertex3f(Size/2,0,0);
    Vertices[1]=_vertex3f(Size/2,Size/2,0);
    Vertices[2]=_vertex3f(-Size/2,Size/2,0);
    Vertices[3]=_vertex3f(-Size/2,0,0);*/
  //eje z
  /*Vertices[0]=_vertex3f(0,0,-Size/2);
  Vertices[1]=_vertex3f(0,Size/2,-Size/2);
  Vertices[2]=_vertex3f(0,Size/2,Size/2);
  Vertices[3]=_vertex3f(0,0,Size/2);*/
//P3, P2, P1, P0
//  Vertices[3]=_vertex3f(0,-Size/2,0);
//  Vertices[2]=_vertex3f(Size/2,-Size/2,0);
//  Vertices[1]=_vertex3f(Size/2,Size/2,0);
//  Vertices[0]=_vertex3f(0,Size/2,0);

  /* 3 (0,Y,0)
   * O-----O 2 (X,Y,0)
```

```
   * |       |
   * |       |
   * O-----O 1 (X,-Y,0)
   * 0 (0,-Y,0)
   */

}

_cilindro::~_cilindro(){
   Vertices.clear();
   Triangles.clear();
}
```

Cono.cc

```
#include "cono.h"
_cono::_cono(float Size)
{
  Vertices.resize(3);
    //perfil
  /*Vertices[0]=_vertex3f(0,-Size/2,0);
  Vertices[1]=_vertex3f(Size/2,-Size/2,0);
  Vertices[2]=_vertex3f(0,Size/2,0);*/
  /*Vertices[2]=_vertex3f(0,-Size/2,0);
  Vertices[1]=_vertex3f(Size/2,-Size/2,0);
  Vertices[0]=_vertex3f(0,Size/2,0);*/

  //eje x
  /*Vertices[0]=_vertex3f(Size/2,0,0);
  Vertices[1]=_vertex3f(Size/2,Size/2,0);
  Vertices[2]=_vertex3f(-Size/2,0,0);*/

  //eje z
  Vertices[0]=_vertex3f(0,0,-Size/2);
  Vertices[1]=_vertex3f(0,Size/2,-Size/2);
  Vertices[2]=_vertex3f(0,0,Size/2);
}
_cono::~_cono(){
   Vertices.clear();
   Triangles.clear();
}
```

Cube.cc

```
#include "cube.h"

_cube::_cube(float Size)
{
  Vertices.resize(8);
    //cara de atras
  Vertices[0]=_vertex3f(-Size/2,-Size/2,-Size/2);
  Vertices[1]=_vertex3f(Size/2,-Size/2,-Size/2);
```

```
  Vertices[2]=_vertex3f(Size/2,Size/2,-Size/2);
  Vertices[3]=_vertex3f(-Size/2,Size/2,-Size/2);

  //cara de alante
  Vertices[4]=_vertex3f(-Size/2,Size/2,Size/2);
  Vertices[5]=_vertex3f(-Size/2,-Size/2,Size/2);
  Vertices[6]=_vertex3f(Size/2,-Size/2,Size/2);
  Vertices[7]=_vertex3f(Size/2,Size/2,Size/2);

  //LOS 12 TRIANGULOS QUE FORMAN EL CUBO
  Triangles.resize(12);

  Triangles[0]=_vertex3ui(0,1,3);
  Triangles[1]=_vertex3ui(1,2,3);
  Triangles[2]=_vertex3ui(6,1,2);
  Triangles[3]=_vertex3ui(7,6,2);
  Triangles[4]=_vertex3ui(5,4,0);
  Triangles[5]=_vertex3ui(0,4,3);
  Triangles[6]=_vertex3ui(4,5,6);
  Triangles[7]=_vertex3ui(4,6,7);
  Triangles[8]=_vertex3ui(2,3,4);
  Triangles[9]=_vertex3ui(2,4,7);
  Triangles[10]=_vertex3ui(0,5,6);
  Triangles[11]=_vertex3ui(6,1,0);

}

_cube::~_cube(){
   Vertices.clear();
   Triangles.clear();
}
```

Esfera.cc

```
#include "esfera.h"
#include <cmath>

_esfera::_esfera(float Size)
{
  int tam = 30;//30
  Vertices.clear();
```

```
  Vertices.resize(tam+1);

  //creamos el angulo
  double angulo = M_PI/tam;

  //creamos el prefil un semicirculo
  for(int i=0; i<=tam; ++i){
     //Vertices.push_back(_vertex3f((Size/2)*sin(angulo*i),(Size/2)*cos(angulo*i) ,0));
     Vertices[i]=_vertex3f((Size/2)*sin(angulo*i),(Size/2)*cos(angulo*i) ,0);
  }
}

_esfera::~_esfera(){
   Vertices.clear();
   Triangles.clear();
}
```

Glwidget.cc

```
#include "glwidget.h"
#include "window.h"
#include "string"

using namespace std;
using namespace _gl_widget_ne;
using namespace _colors_ne;

_gl_widget::_gl_widget(_window *Window1):Window(Window1)
{
  setMinimumSize(300, 300);
  setFocusPolicy(Qt::StrongFocus);
  Cono.vueltas_optima(divisiones,'z');
  Cilindro.vueltas_optima(divisiones,'y');
  Esfera.vueltas_optima(divisiones,'y');
  ply_mio = new _ply("../skeleton/ply_models/big_porsche.ply");
}


/*****************************************************************************//**
 * Evento tecla pulsada
 *****************************************************************************/
//funcion que hace que eventos con el teclado (1,2,p,l,f,c)
void _gl_widget::keyPressEvent(QKeyEvent *Keyevent)
{
  switch(Keyevent->key()){
  case Qt::Key_1:Object=OBJECT_TETRAHEDRON;break;
  case Qt::Key_2:Object=OBJECT_CUBE;break;
  case Qt::Key_3:
     Object=OBJECT_CONO;
  break;
  case Qt::Key_4:
```

```
    Object=OBJECT_CILINDRO;
  break;
  case Qt::Key_5:
    Object=OBJECT_ESFERA;
  break;
  case Qt::Key_6:
    Object=OBJECT_PLY;
  break;
  case Qt::Key_7:
    Object=OBJECT_EX;
  break;
  case Qt::Key_8:
    Object=OBJECT_EX2;
    break;
  case Qt::Key_9:
    Object=OBJECT_EX3;
  break;
  case Qt::Key_P:Draw_point=!Draw_point;break;
  case Qt::Key_L:Draw_line=!Draw_line;break;
  case Qt::Key_Q:++mover;break;
  case Qt::Key_W:--mover;break;
  case Qt::Key_A:++mover;break;
  case Qt::Key_S:--mover;break;
  case Qt::Key_Z:++mover2;break;
  case Qt::Key_X:--mover2;break;
  case Qt::Key_F:
    Draw_fill=!Draw_fill;
    if(Draw_chess){
      Draw_chess=!Draw_chess;
    }
    break;

  case Qt::Key_C:
    Draw_chess=!Draw_chess;
    if(Draw_fill){
      Draw_fill=!Draw_fill;
     }
    break;

  case Qt::Key_Left:Observer_angle_y-=ANGLE_STEP;break;
  case Qt::Key_Right:Observer_angle_y+=ANGLE_STEP;break;
  case Qt::Key_Up:Observer_angle_x-=ANGLE_STEP;break;
  case Qt::Key_Down:Observer_angle_x+=ANGLE_STEP;break;
  case Qt::Key_PageUp:Observer_distance*=1.2;break;
  case Qt::Key_PageDown:Observer_distance/=1.2;break;
  }

  update();//función de qt que cada vez que hagas una acción llama a paintGL()
}


/*****************************************************************************//**
```

```
 * Limpiar ventana
 ********************************************************************************/

void _gl_widget::clear_window()
{
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
}

/*********************************************************************************//**
 * Funcion para definir la transformación de proyeccion
 ********************************************************************************/

void _gl_widget::change_projection()
{
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();

 // formato(x_minimo,x_maximo, y_minimo, y_maximo,Front_plane, plano_traser)
 // Front_plane>0  Back_plane>PlanoDelantero)

glFrustum(X_MIN,X_MAX,Y_MIN,Y_MAX,FRONT_PLANE_PERSPECTIVE,BACK_PLANE_PERSPECTIVE);
}

/*********************************************************************************//**
 * Funcion para definir la transformación de vista (posicionar la camara)
 ********************************************************************************/

void _gl_widget::change_observer()
{
 // posicion del observador
 glMatrixMode(GL_MODELVIEW);
 glLoadIdentity();
 glTranslatef(0,0,-Observer_distance);
 glRotatef(Observer_angle_x,1,0,0);
 glRotatef(Observer_angle_y,0,1,0);
}


/*********************************************************************************//**
 * Funcion que dibuja los objetos
 ********************************************************************************/
void _gl_widget::draw_objects()
{
 Axis.draw_line();
//dibuja los puntos
 if (Draw_point){
   glPointSize(5);
   glColor3fv((GLfloat *) &BLACK);

   switch (Object){
   case OBJECT_TETRAHEDRON:Tetrahedron.draw_point();break;
   case OBJECT_CUBE: Cube.draw_point();break;
```

```
      case OBJECT_EX: bra1.draw(1, 0);break;
      case OBJECT_EX2:
      bra2.draw(1,mover);
      break;
   case OBJECT_EX3:
   piv.draw(1,mover,mover2);
      break;
   case OBJECT_CONO:
      //dibuja los puntos
      Cono.draw_point();
      break;
   case OBJECT_CILINDRO:
      //dibuja los puntos
      Cilindro.draw_point();
      break;
   case OBJECT_ESFERA:
      Esfera.draw_point();
      break;
   case OBJECT_PLY:ply_mio->draw_point();break;
   default:break;
   }
  }

  //pinta las lineas
  if (Draw_line){
   glLineWidth(3);
   glColor3fv((GLfloat *) &MAGENTA);//para que lo pinte en el color que se indica
   switch (Object){
   case OBJECT_TETRAHEDRON:Tetrahedron.draw_line();break;//pinta tetraedro
   case OBJECT_CUBE:Cube.draw_line();break; // pinta cubo
   case OBJECT_CONO:Cono.draw_line();break; // pinta cono
   case OBJECT_CILINDRO:Cilindro.draw_line();break; // pinta cilindro
   case OBJECT_ESFERA:Esfera.draw_line(); break; //pinta la esfera
   case OBJECT_PLY:ply_mio->draw_line(); break;
      case OBJECT_EX:

      bra1.draw(2,0);break;
   case OBJECT_EX2:
   bra2.draw(2,mover);
   break;
   case OBJECT_EX3:
   piv.draw(2,mover,mover2);break;
   default:break;
   }
  }
//pinta relleno
  if (Draw_fill){
   glColor3fv((GLfloat *) &BLUE);
   switch (Object){
   case OBJECT_TETRAHEDRON:Tetrahedron.draw_fill();break;
   case OBJECT_CUBE:Cube.draw_fill();break;
   case OBJECT_CONO:Cono.draw_fill();break;
```

```
   case OBJECT_CILINDRO:Cilindro.draw_fill();break;
   case OBJECT_ESFERA:Esfera.draw_fill(); break;
   case OBJECT_PLY:ply_mio->draw_fill(); break;
      case OBJECT_EX: bra1.draw(3, 0);break;
   case OBJECT_EX2:
   bra2.draw(3,mover);
   break;
   case OBJECT_EX3:
   piv.draw(3,mover,mover2);break;
   default:break;
   }
 }
//pinta un triangulo de un color y el otro de otro color
 if (Draw_chess){
   switch (Object){
   case OBJECT_TETRAHEDRON:Tetrahedron.draw_chess();break;
   case OBJECT_CUBE:Cube.draw_chess();break;
   case OBJECT_CONO:Cono.draw_chess();break;
   case OBJECT_CILINDRO:Cilindro.draw_chess();break;
   case OBJECT_ESFERA:Esfera.draw_chess();break;
   case OBJECT_PLY:ply_mio->draw_chess();break;
      case OBJECT_EX: bra1.draw(4, 0);break;
   case OBJECT_EX2:
   bra2.draw(4,mover);
   break;
   case OBJECT_EX3:
   piv.draw(4,mover,mover2);
   break;
   default:break;
   }
 }
}

/*************************************************************************//**
 * Evento de dibujado
 ****************************************************************************/
//función que redibuja
void _gl_widget::paintGL()
{
 clear_window();//borra la ventana
 change_projection();//have los cambios de projección
 change_observer();//hace cambios de observador
 draw_objects();//pinta el nuevo dibujo
}

/*************************************************************************//**
 * Evento de cambio de tamaño de la ventana
 ****************************************************************************/

void _gl_widget::resizeGL(int Width1, int Height1)
{
 glViewport(0,0,Width1,Height1);
```

```
}

/*****************************************************************************//**
 * Inicialización de OpenGL
 *****************************************************************************/

void _gl_widget::initializeGL()
{
  const GLubyte* strm;

  strm = glGetString(GL_VENDOR);
  std::cerr << "Vendor: " << strm << "\n";
  strm = glGetString(GL_RENDERER);
  std::cerr << "Renderer: " << strm << "\n";
  strm = glGetString(GL_VERSION);
  std::cerr << "OpenGL Version: " << strm << "\n";

  if (strm[0] == '1'){
    std::cerr << "Only OpenGL 1.X supported!\n";
    exit(-1);
  }

  strm = glGetString(GL_SHADING_LANGUAGE_VERSION);
  std::cerr << "GLSL Version: " << strm << "\n";

  int Max_texture_size=0;
  glGetIntegerv(GL_MAX_TEXTURE_SIZE, &Max_texture_size);
  std::cerr << "Max texture size: " << Max_texture_size << "\n";

  glClearColor(1.0,1.0,1.0,1.0);
  glEnable(GL_DEPTH_TEST);;

  Observer_angle_x=0;
  Observer_angle_y=0;
  Observer_distance=DEFAULT_DISTANCE;

  Draw_point=true;
  Draw_line=false;
  Draw_fill=false;
  Draw_chess=false;

  Object = _gl_widget_ne::OBJECT_TETRAHEDRON;
}
```

Glwidget.h

```
#ifndef GLWIDGET_H
#define GLWIDGET_H

#include <GL/gl.h>
#include <QOpenGLWidget>
#include <QKeyEvent>
#include <iostream>
```

```
#include "vertex.h"
#include "colors.h"
#include "axis.h"
#include "tetrahedron.h"
#include "cube.h"
#include "cono.h"
#include "cilindro.h"
#include "esfera.h"
#include "ply.h"
#include <string.h>
#include "iostream"
#include "brazo1.h"
#include "brazo2.h"
#include "pivote.h"

namespace _gl_widget_ne {

  const float X_MIN=-.1;
  const float X_MAX=.1;
  const float Y_MIN=-.1;
  const float Y_MAX=.1;
  const float FRONT_PLANE_PERSPECTIVE=(X_MAX-X_MIN)/2;
  const float BACK_PLANE_PERSPECTIVE=1000;
  const float DEFAULT_DISTANCE=2;
  const float ANGLE_STEP=1;

  typedef enum {MODE_DRAW_POINT,MODE_DRAW_LINE,MODE_DRAW_FILL,MODE_DRAW_CHESS} _mode_draw;
  typedef enum {OBJECT_TETRAHEDRON,OBJECT_CUBE,OBJECT_CONO,OBJECT_CILINDRO,OBJECT_ESFERA,OBJECT_PLY,
OBJECT_EX, OBJECT_EX2,OBJECT_EX3} _object;}

class _window;

class _gl_widget : public QOpenGLWidget
{
Q_OBJECT //cada vez que programamos orientada a eventos
public:
  _gl_widget(_window *Window1);

  void clear_window();
  void change_projection();
  void change_observer();

  void draw_axis();
  void draw_objects();


protected:
  void resizeGL(int Width1, int Height1) Q_DECL_OVERRIDE;
  void paintGL() Q_DECL_OVERRIDE;
  void initializeGL() Q_DECL_OVERRIDE;
  void keyPressEvent(QKeyEvent *Keyevent) Q_DECL_OVERRIDE;
```

```
private:
 _window *Window;

 _axis Axis;
 _tetrahedron Tetrahedron;
 _cube Cube;
 _Brazo1 bra1;
 _Brazo2 bra2;
 _Pivote piv;
 _cono Cono;
 _cilindro Cilindro;
 _esfera Esfera;
 _ply *ply_mio;

 int mover = 0;
 int mover2 = 0;

 int divisiones = 30;//30

 _gl_widget_ne::_object Object;

 bool Draw_point;
 bool Draw_line;
 bool Draw_fill;
 bool Draw_chess;

 float Observer_angle_x;
 float Observer_angle_y;
 float Observer_distance;
};

#endif
```

Object3d.cc

```
#include "object3d.h"

using namespace _colors_ne;

void _object3D::draw_line()
{
   //le indico por las 2 caras y que pinte lineas
   glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
   glBegin(GL_TRIANGLES);

   for (unsigned int i=0;i<Triangles.size();i++){
     glVertex3fv((GLfloat *) &Vertices[Triangles[i]._0]);
     glVertex3fv((GLfloat *) &Vertices[Triangles[i]._1]);
     glVertex3fv((GLfloat *) &Vertices[Triangles[i]._2]);
   }
```

```
   glEnd();
}

void _object3D::draw_fill()
{
   glPolygonMode(GL_FRONT_AND_BACK,GL_FILL);
   //glPolygonMode(GL_FRONT,GL_FILL);
   glBegin(GL_TRIANGLES);
   for (unsigned int i=0;i<Triangles.size();i++){
     glVertex3fv((GLfloat *) &Vertices[Triangles[i]._0]);
     glVertex3fv((GLfloat *) &Vertices[Triangles[i]._1]);
     glVertex3fv((GLfloat *) &Vertices[Triangles[i]._2]);
   }
   glEnd();
}

void _object3D::draw_chess()
{
   //para que pinte la cara de alante y atras
   glPolygonMode(GL_FRONT_AND_BACK,GL_FILL);
   //selecciono los triangulos
   glBegin(GL_TRIANGLES);
   //recorro los triangulos 4 tetraedro 12 cubo
   for (unsigned int i=0;i<Triangles.size();++i){
      //si es impar los pinta negros
      glColor3fv((GLfloat *) &YEllOW);
      //y si es par lo pinta en rojo
      if(i%2==0){
         glColor3fv((GLfloat *) &RED);
      }

     //hace el relleno del triangulo de los tres puntos
     glVertex3fv((GLfloat *) &Vertices[Triangles[i]._0]);
     glVertex3fv((GLfloat *) &Vertices[Triangles[i]._1]);
     glVertex3fv((GLfloat *) &Vertices[Triangles[i]._2]);
   }
   glEnd();
}
```

Object3d.h

```
#ifndef OBJECT3D_H
#define OBJECT3D_H

#include "basic_object3d.h"

class _object3D:public _basic_object3D
{
  public:
  vector<_vertex3ui> Triangles;
```

```
  void draw_line();
  void draw_fill();
  void draw_chess();

};

#endif // OBJECT3D_H
```

Object3dr.cc

```
#include "object3dr.h"
#include <cmath>

//crea la estructura de alambres
void _object3Dr::create_triangles(int n_caras, int tam){
   int j=0,cara=0, mod=(n_caras+1)*tam;
   //int indice=0, pos_t=0;
   int tope = mod*2;

   Triangles.resize(tope);

   for(int i=0;i<mod;++i){

      if(cara<n_caras){
         ++cara;
      }else{
         cara=0;
         ++j;
      }
      if(cara%2){
         //creamos la tapa inferior y sus caras (superior)
         Triangles[i*2]=_vertex3ui(((i*tam)+j)%mod,(((i+1)*tam)+j)%mod,((i*tam)+j+1)%mod);
         //creamos la tapa superior y sus caras (inferior)
         Triangles[(2*i)+1]=_vertex3ui((((i+1)*tam)+j)%mod,(((i+1)*tam)+j+1)%mod,((i*tam)+j+1)%mod);
      }else{
         //creamos la tapa inferior y sus caras (superior)
         Triangles[(2*i)+1]=_vertex3ui(((i*tam)+j)%mod,(((i+1)*tam)+j)%mod,((i*tam)+j+1)%mod);
         //creamos la tapa superior y sus caras (inferior)
         Triangles[i*2]=_vertex3ui((((i+1)*tam)+j)%mod,(((i+1)*tam)+j+1)%mod,((i*tam)+j+1)%mod);
      }

    }
   /*for(int i=0;i<n_caras;++i){
      for(int j=0;j<tam-1;++j){

         //crea la estructura de alambres
         indice = (i*tam)+j;

         //creamos la tapa superior y sus caras (inferior)
         //ejemplo: i = 0, j = 1, tam = 4
         //1º = (1 * 4) + 0     = 4
```

```
        //2º = (1 * 4) + 0 + 4 = 8
        //3º = (1 * 4) + 0 + 1 = 5

        Triangles[pos_t]=_vertex3ui(indice,indice+tam,indice+1);
        ++pos_t;
        cout << "POS: " << pos_t << endl;
        Triangles[pos_t]=_vertex3ui(indice+1,indice+tam,indice+tam+1);
        ++pos_t;
      }
    }*/
    /*for(int i=0;i<n_caras;++i){
      for(int j=0;j<tam-1;++j){

        //crea la estructura de alambres
        indice = (i*tam)+j;

        //creamos la tapa inferior y sus caras (superior)
        //1º = (1 * 4) + 0 + 4 = 8
        //2º = (1 * 4) + 0 + 5 = 9
        //3º = (1 * 4) + 0 + 1 = 5
        Triangles[pos_t]=_vertex3ui(indice+1,indice+tam,indice+tam+1);
        ++pos_t;
      }
    }*/
}
//crea la estructura de alambres
void _object3Dr::create_triangles_optima(int n_caras, int tam){

    if(n_caras>0){
      /* GUIA
       * mod --> es el tamaño total de puntos
       * cara --> es la posición de cada cara generada (2 Triangulos)
       * j --> variable auxiliar matricial
       * tope --> es el numero total de triangulos a generar
       * first --> para que no repita triangulos de la primera tapa
       * last --> para que no repita triangulos de la ultima tapa
       * pos --> posicion en el vector de triangulos
       * first --> crea la tapa inferior
       * last --> crea la tapa superior
       */

      int j=0,cara=0, caras_total=(n_caras)*(tam-1), tope = (n_caras*(tam-3))*2 +(n_caras*2);
      int mod =Vertices.size(), pos=0;
      bool first=true, last=false;
      _vertex3ui triangulo_aux;//triangulo auxiliar para alternar los triangulos

      /*cout << "TOPE: " << tope << endl;
      cout << "PUNTOS TOTAL: " << mod << endl;
      cout << "CARAS TOTAL: " << caras_total << endl;
      cout << "DIVISIONES: " << n_caras << endl;
      cout << "PUNTOS BASE: " << tam << endl;*/
```

```
Triangles.resize(tope);//reservamos el tamaño
// k = (numero puntos base) - 2
int p3=1, p2=tam, k=tam-2, pl1=k, pl2=(k*2)+1, p1=1;

for(int i=0;i<caras_total;++i){//caras_total

   if(cara<n_caras){//recorre las caras
      ++cara;
   }else{
      //empieza a partir de la 2º fila 0, 1, 2, ...
      //porque la primera solo tiene 1 triangulo
      if(!first)
         ++j;
      //estos los he deducido por inducción
      //empieza siempre por 2 + j
      p3=2+j;
      //empieza siempre por el numero de puntos base + j
      p2=tam+j;
      //empieza siempre por 1 + j
      p1=1+j;
      //ha recorrido las caras de arriba
      first=false;
      //resetea el contador de caras a 1
      //caras va de 1 hasta n_caras(divisiones)
      cara=1;

   }
   //cout << "CARA: " << cara << endl;
   //si i coincide con la primera posición de las caras de arriba
   if(i==(caras_total-n_caras)){
      last=true;
   }
   //cout << "j: " << j << "---> " << pos << endl;

   //crea las caras de abajo
   if(first){

      //cout << "TRIANGULO 1: " << 0 << ", " << p2 << ", " << p3 << endl;
      if(pos%2==0)
         Triangles[pos+1]=_vertex3ui(0,p2,p3);
      else
         Triangles[pos-1]=_vertex3ui(0,p2,p3);

      //INCREMENTOS

      //si es el segundo triangulo le incremento
      //p3 += (numero puntos base)-1
      if(i+1==1)
         p3+=k+1;
      //en caso contrario le incremento
      //p3 += (numero puntos base)-2
      else
```

```
            p3+=k;

        //para todos los triangulos
        //p2 += (numero puntos base)-2
        p2+=k;
        //salvo si p2 es mayor o igual al total de puntos
        if(p2>=mod)
            ++p2;
        //y le hago modulo al total de puntos
        p2=p2%mod;

    //crea las caras de arriba
    }else if(last){

        //cout << "TRIANGULO 2: " << pl1 << ", " << pl2 << ", " << tam-1 << endl;
        Triangles[pos]=_vertex3ui(pl1,pl2,tam-1);

        //INCREMENTOS

        //si es la segunda cara le incremento
        //pl1 += (numero puntos base)-1
        if(cara+1==2){
            pl1+=k+1;
        //en caso contrario
        //pl1 += (numero puntos base)-2
        }else{
            pl1+=k;
        }

        //si es la ultima cara se incrementa
        //pl2 += (numero puntos base)-1
        if(cara+1==n_caras){
            pl2+=k+1;
        //en caso contrario
        //pl2 += (numero puntos base)-2
        }else{++j;
            pl2+=k;
        }
        //y le hago modulo al total de puntos
        pl2=pl2%mod;
    }else{

        /*cout << "TRIANGULO medio1: " << p1 << ", " << p2 << ", " << p3 << endl;
        cout << "TRIANGULO medio2: " << p2 << ", " << (p2+1)%mod << ", " << p3 << endl;*/
        if(cara%2){
            Triangles[pos]=_vertex3ui(p1,p2,p3);
            ++pos;
            Triangles[pos]=_vertex3ui(p2,(p2+1)%mod,p3);
        }else{
            Triangles[pos]=_vertex3ui(p2,(p2+1)%mod,p3);
            ++pos;
            Triangles[pos]=_vertex3ui(p1,p2,p3);
```

```
            }

            //INCREMENTOS

            //si es la segunda cara de la columna incrementa
            //p3 y p1 += (numero puntos base) - 1
            if(cara+1==2){
               p3+=k+1;
               p1+=k+1;
            //en caso contrario
            //p1 y p3 += (numero puntos base) - 2
            }else{
               p3+=k;
               p1+=k;
            }
            //si es la ultima cara de la columna
            //p2 += (numero puntos base) - 1
            if(cara+1==n_caras){
               p2+=k+1;
            //en caso contrario
            //p2 += (numero puntos base) - 2
            }else{
               p2+=k;
            }

            //y les hago modulo al numero de puntos total p1 y p2
            p2 = p2%mod;
            p1 = p1%mod;
         }
         ++pos;
      }
   }
}


//crea la estructura de puntos
void _object3Dr::vueltas(int n_caras){
   _vertex3f aux;
   //calculo el angulo inicial (angulo de partida)
   double ang_inicial = (2*M_PI)/n_caras;
   //calculo el nuevo angulo (angulo de referencia a sumar)
   double ang_mod = (2*M_PI)/n_caras;

   int tam = Vertices.size(), cont_c=0;

   for(int i=0; i<n_caras*tam; ++i){
      ++cont_c;
      Vertices.push_back(_vertex3f(Vertices[i%tam].x*cos(ang_inicial),//x_nueva = R * cos(angulo)
                  Vertices[i%tam].y,// y_nueva = y
                  Vertices[i%tam].x*sin(ang_inicial)));//z_nueva = R * sin(angulo));
      Vertices[tam+i]=_vertex3f(Vertices[i%tam].x*cos(ang_inicial),//x_nueva = R * cos(angulo)
                        Vertices[i%tam].y,// y_nueva = y
```

```
                        Vertices[i%tam].x*sin(ang_inicial));//z_nueva = R * sin(angulo)

      //cout << "PUNTOS: " << i << ", mod: " << i%tam << endl;

      if(cont_c==tam){
        //cout << "Cuantas veces entra? " << i << endl;
        //paso al siguiente angulo (siguiente división)
        ang_inicial+=ang_mod;
        cont_c=0;
      }

    }

    /*for(int i=0; i<n_caras; ++i){
      for(int j=0; j<tam; ++j){
        //R = x
        //creamos los vertices
        Vertices.push_back(_vertex3f(Vertices[j].x*cos(ang_inicial),//x_nueva = R * cos(angulo)
                                     Vertices[j].y,// y_nueva = y
                                     Vertices[j].x*sin(ang_inicial)));//z_nueva = R * sin(angulo)
      }
      //paso al siguiente angulo (siguiente división)
      ang_inicial+=ang_mod;
    }*/
    create_triangles(n_caras, tam);
}

//crea la estructura de puntos de forma optima
void _object3Dr::vueltas_optima(int n_caras, char eje){
    _vertex3f aux;
    //calculo el angulo inicial (angulo de partida)
    double ang_inicial = (2*M_PI)/n_caras;
    //calculo el nuevo angulo (angulo de referencia a sumar)
    double ang_mod = (2*M_PI)/n_caras;

    //guardo el numero de puntos base
    int tam = Vertices.size(), cont_p=1;

    //cout << "TAMAÑO BUCLE: " << (n_caras-1)*(tam-2) << endl;
    float x,y,z;
    //recorro (n_caras-1) * (tam-2)
    //porque para que no genere puntos repetidos tengo que eliminar los puntos
    //repetidos centrales y los de la ultima cara que coinciden con los de la base
    for(int i=0; i<(n_caras-1)*(tam-2); ++i){//(n_caras-1)*(tam-2)

      //cout << "PUNTOS: " << i << ", cont_p: " << cont_p << endl;

      //genero el punto
      switch (eje) {
        case 'x':
          //y_nueva = R * cos(angulo)
          y = (Vertices[cont_p].y*cos(ang_inicial)) - (Vertices[cont_p].z*cos(ang_inicial));
```

```
            //z_nueva = R * sin(angulo)
            z = (Vertices[cont_p].y*sin(ang_inicial)) + (Vertices[cont_p].z*cos(ang_inicial));
            // x_nueva = x
            Vertices.push_back(_vertex3f(Vertices[cont_p].x,y,z));
         break;
         case 'y':
            //x_nueva = R * cos(angulo)
            x = (Vertices[cont_p].x*cos(ang_inicial)) + (Vertices[cont_p].z*sin(ang_inicial));
            //z_nueva = R * sin(angulo)
            z = (Vertices[cont_p].x*sin(ang_inicial)) + ((Vertices[cont_p].z*cos(ang_inicial)));
            // y_nueva = y
            Vertices.push_back(_vertex3f(x,Vertices[cont_p].y,z));
         break;
         case 'z':
            //x_nueva = R * cos(angulo)
            x = (Vertices[cont_p].x*cos(ang_inicial))+(Vertices[cont_p].y*sin(ang_inicial));
            //y_nueva = R * sin(angulo)
            y = (Vertices[cont_p].x*sin(ang_inicial))+(Vertices[cont_p].y*cos(ang_inicial));
            //z_nueva = z
            Vertices.push_back(_vertex3f(x,y,Vertices[cont_p].z));
         break;
      }


      //incremento la posición de los puntos base
      ++cont_p;
      //si el punto es uno de los centrales (0 o n-1) lo pongo a 1
      if(cont_p%(tam-1)==0){
         cont_p=1;
      }
      //si el siguiente punto es 1 avanzo en la rotación
      if(cont_p==1){
         //cout << "Cambia angulo" << endl;
         //paso al siguiente angulo (siguiente división)
         ang_inicial+=ang_mod;
      }

   }
   //genera los triangulos
   create_triangles_optima(n_caras, tam);
}
```

Object3dr.h

```
#ifndef O3DR_H
#define O3DR_H

#include <object3d.h>

class _object3Dr : public _object3D{
  private:
   bool tapa_superior=false, tapa_inferior=false;
   _vertex3f primero, ultimo;
  public:
   void create_triangles(int n_caras, int tam);
   void vueltas(int n_caras);

   //optima
   void create_triangles_optima(int n_caras, int tam);
   void vueltas_optima(int n_caras, char eje);
};

#endif // O3DR_H
```

Pivote.cc

```
#include "pivote.h"

_Pivote::_Pivote(){p.vueltas_optima(30,'y');}
void _Pivote::draw(int modo, int mover, int rotar){
   glMatrixMode(GL_MODELVIEW);

   glPushMatrix();
   glRotatef(90,1,0,0);
   glScalef(1.5,1.5,1.5);
   if(modo == 1)
      p.draw_point();
   if(modo == 2)
      p.draw_line();
   if(modo == 3)
      p.draw_fill();
   if(modo == 4)
      p.draw_chess();
   glPopMatrix();

   glMatrixMode(GL_MODELVIEW);

   glPushMatrix();
   glRotatef(rotar,0,0,1);
   glTranslatef(0.75,0,0);

   br.draw(modo,mover);
   glPopMatrix();
```

```
}
```

Pivote.h

```
#ifndef PIVOTE_H
#define PIVOTE_H
#include "cilindro.h"
#include "brazo2.h"

class _Pivote{
private:
   _cilindro p;
   _Brazo2 br;

   //tamaño del brazo
   float size=1;
public:
   _Pivote();
   void draw(int modo, int mover, int rotar);//dibuja
};

#endif
```

Tetrahedron.cc

```
#include "tetrahedron.h"

_tetrahedron::_tetrahedron(float Size)
{
 Vertices.resize(4);

 Vertices[0]=_vertex3f(-Size/2,-Size/2,-Size/2);
 Vertices[1]=_vertex3f(0,-Size/2,Size/2);
 Vertices[2]=_vertex3f(Size/2,-Size/2,-Size/2);
 Vertices[3]=_vertex3f(0,Size/2,0);

 Triangles.resize(4);


 Triangles[0]=_vertex3ui(1,2,3);
 Triangles[1]=_vertex3ui(0,1,3);
 Triangles[2]=_vertex3ui(2,0,3);
 Triangles[3]=_vertex3ui(0,2,1);
}

_tetrahedron::~_tetrahedron(){
   Vertices.clear();
   Triangles.clear();
}
```

La practica del examen la podeis descargar aquí

https://mega.nz/file/VH5CUILD#HMa4gSQnOwATuK5gg7Bd5GVetlhQUF3rG1JsDsauTGA