examen-practico-resuelto2.pdf

PruebaAlien

Informática Gráfica

3º Grado en Ingeniería Informática

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

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();
}
```

Cilindro.h

```
#ifndef CILINDRO_H
#define CILINDRO_H

#include "object3dr.h"

class _cilindro:public _object3Dr
{
public:
  _cilindro(float Size=1.0);
  ~_cilindro();
};

#endif
```

Esqueleto_qt.pro

```
HEADERS += \
  cilindro.h \
  colors.h \
  basic_object3d.h \
  cono.h \
  cube.h \
  esfera.h \
  object3d.h \
  axis.h \
  object3dr.h \
  ply.h \
  soporte.h \
  tetrahedron.h \
  glwidget.h \
  file_ply_stl.h \
  varilla.h \
  window.h

SOURCES += \
  basic_object3d.cc \
  cilindro.cc \
  cube.cc \
  esfera.cc \
  object3d.cc \
  axis.cc \
  object3dr.cc \
  ply.cc \
  soporte.cc \
  tetrahedron.cc \
  glwidget.cc \
  varilla.cc \
  window.cc \
  file_ply_stl.cc \
```

KEEP CALM AND ESTUDIA UN POQUITO

WUOLAH + #QuédateEnCasa

```
 cono.cc \
 main.cc


LIBS += -L/usr/X11R6/lib64 -lGL


CONFIG += c++11
QT += widgets
```

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_P:Draw_point=!Draw_point;break;
  case Qt::Key_L:Draw_line=!Draw_line;break;
  case Qt::Key_A:++mover;break;
  case Qt::Key_S:--mover;break;
  case Qt::Key_Z:++mover1;break;
  case Qt::Key_X:--mover1;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: barilla.draw(1, mover);break;
     case OBJECT_EX2:
     sop.draw(1,mover1);
     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:

      barilla.draw(2, mover);break;
    case OBJECT_EX2:
    sop.draw(2,mover1);
    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: barilla.draw(3, mover);break;
    case OBJECT_EX2:
    sop.draw(3,mover1);
    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: barilla.draw(4, mover);break;
   case OBJECT_EX2:
   sop.draw(4,mover1);
   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 "varilla.h"
#include "soporte.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;
}

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;
 Varilla barilla;
 _cono Cono;
 _cilindro Cilindro;
 _esfera Esfera;
 _ply *ply_mio;
 Soporte sop;
```

```
  int mover = 0;
  int mover1 = 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
```

Soporte.cc

```
#include "soporte.h"

Soporte::Soporte(){sop.vueltas_optima(30,'y');}
void Soporte::draw(int modo, int mover){
   //soporte
   glMatrixMode(GL_MODELVIEW);
   glPushMatrix();
   glTranslatef(0,7.5,0);
   glScalef(1,15,1);
   if(modo == 1)
      sop.draw_point();
   if(modo == 2)
      sop.draw_line();
   if(modo == 3)
      sop.draw_fill();
   if(modo == 4)
      sop.draw_chess();
   glPopMatrix();

   //varillas
   for(int i=0; i<g; ++i){
      glMatrixMode(GL_MODELVIEW);
      glPushMatrix();
      glRotatef((360/g)*i,0,1,0);
      glTranslatef(0.5,15,0);
      var.draw(modo,mover);
      glPopMatrix();
   }

}
```

Soporte.h

```
#ifndef SOPORTE_H
#define SOPORTE_H
#include "cilindro.h"
#include "varilla.h"
class Soporte{
private:
   _cilindro sop;
   Varilla var;

   //tamaño de la barilla
   float size=1;
   int g=((7%5)+4);
public:
   Soporte();
   void draw(int modo, int mover);//dibuja
};

#endif // SOPORTE_H
```

Varilla.cc

```
#include "varilla.h"

Varilla::Varilla(){bar.vueltas_optima(30,'y');}
void Varilla::draw(int modo, int mover){
   glMatrixMode(GL_MODELVIEW);

   glPushMatrix();
   //traslada el radar con respecto al tamaño del cilindro

   glRotated(10+mover,0,0,1);
   glTranslatef(0,-5,0);
   glScalef(1,10,1);
   if(modo == 1)
      bar.draw_point();
   if(modo == 2)
      bar.draw_line();
   if(modo == 3)
      bar.draw_fill();
   if(modo == 4)
      bar.draw_chess();
   glPopMatrix();
}
```

Varilla.h

```
#ifndef BARILLA_H
#define BARILLA_H
#include "cilindro.h"

class Varilla{
private:
   _cilindro bar;

   //tamaño de la barilla
   float size=1;
public:
   Varilla();
   void draw(int modo, int mover);//dibuja
};

#endif
```

Podeis descargar la practica aqui

https://mega.nz/file/JGIzSYzA#aFMa-3kZTv8f6I7vzEqNDj9nrsWEAtu3LVbmBn4XlLE