

Master en Desarrollo de Software
Asignatura: “Fundamentos de Geometría y Geometría Computacional”

Tarea 1:
OpenGL - Modeling and rendering of simple 3D

Rev. 1

Francisco Guerrero Aranda

Tabla de Revisiones

Tarea 1: OpenGL - Modeling and rendering of simple 3D

Redactado por:	Revisado por:	Verificado por:	Aprobado por:
Francisco Guerrero Aranda	Nombre y Apellidos (NAA)	Nombre y Apellidos (NAA)	Nombre y Apellidos (NAA)
12/02/2022	DD/MM/AAAA	DD/MM/AAAA	DD/MM/AAAA

MASTER EN DESARROLLO DE SOFTWARE
Fundamentos de Geometría y Geometría Computacional

Tabla de Ediciones

Edición	Fecha	Objeto de la edición
1	12 Febrero 2022	Edición inicial
2		

Índice

1.	Enunciado	2
1.1.	Goals	2
1.2.	Development	2
2.	Desarrollo de la práctica	3
3.	Resultados	11

1. Enunciado

Programación de una aplicación en C++, QtCreator, librerías de Qt, OpenGL que permita crear un objeto 3D, un cubo, y añadir tres modos de visualización.

1.1. Goals

With this practice we want the student to learn to:

- Create and use data structures that allow to represent simple 3D objects.
- Use OpenGL drawing primitives to draw objects.
- Distinguish between what it is to create a model and what it is to visualize it.

1.2. Development

For the development of this practice, the skeleton of an event-based graphic application is delivered, using Qt 5, with the graphic part made by OpenGL, and being C++ the programming language. The application not only contains the OpenGL initialization code and the capture of key events, but a class structure that allows to represent 3D objects, including axes and tetrahedron objects. It is also implemented a camera that moves on the surface of a sphere by pressing the cursor keys, and page forward and page back keys to simulate a zoom in and zoom out.

The student must study and understand the code that is delivered. Once this is done, you must add the functions that allow you to draw in fill mode and chess mode. In addition, following the hierarchy already defined, you must create the cube class and use an instance that allows you to visualize it when the key 2 is pressed.

Therefore, the following drawing modes will be available at the end:

- Points
- Lines
- Fill
- Chess

To be able to visualize in fill mode, you only have to use triangles as a drawing primitive, `GL_TRIANGLES`, and change the way it is displayed using the `glPolygonMode` instruction, so that the inside is drawn. For chess mode, it is enough to draw in solid mode but alternatively changing the fill color.

Define the following keys to activate the different modes and objects:

- Key p: to render in point mode
- Key l: to render in line mode
- Key f: to render in fill mode
- Key c: to render in chess mode
- Key 1: to activate the tetrahedron
- Key 2: to activate the cube

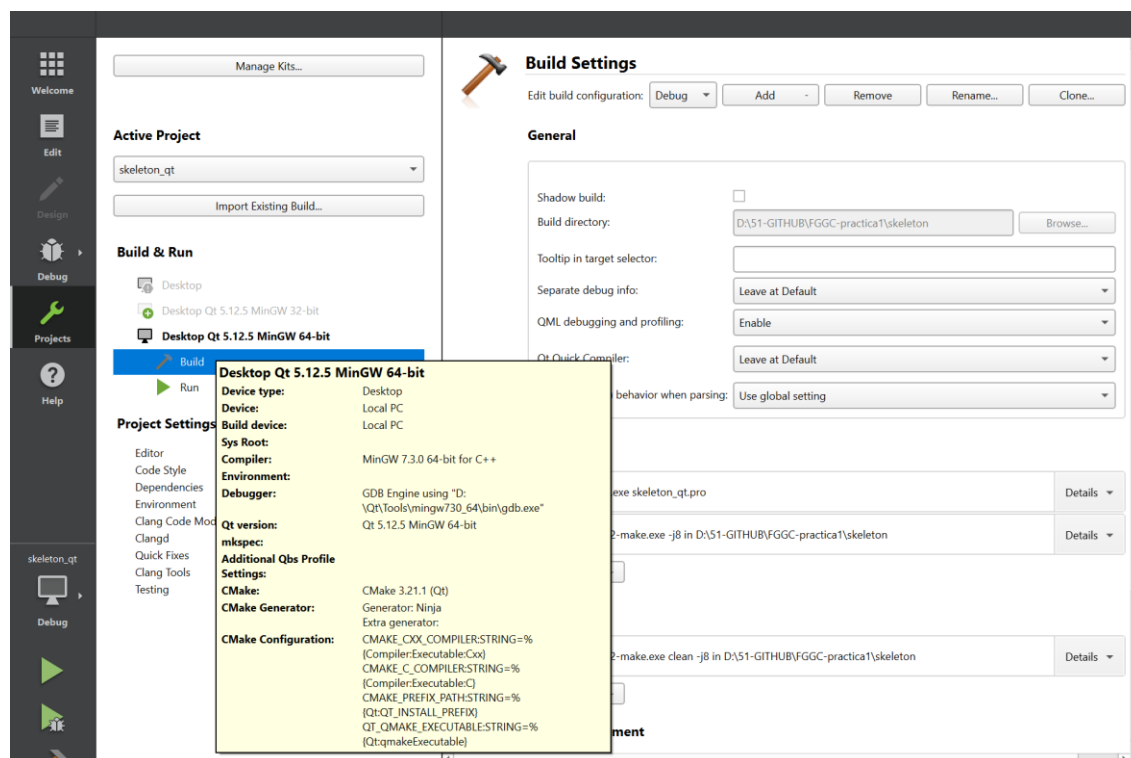
2. Desarrollo de la práctica

En primer lugar hemos repasado los conceptos teóricos de la documentación suministrada y la bibliografía recomendada además de volver a visionar la grabación de la clase que se impartió en su día.

Hemos instalado QT en el siguiente entorno:

- Sistema Operativo: Windows 11 Home.
 - o Versión: 21H2
 - o Sistema operativo de 64 bits, procesador basado en x64
- Hardware:
 - o Procesador: AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx 2.10 GHz
 - o RAM: 8 GB
 - o Modelo: Huawei MateBook 13
- Cmake para Windows x86_64. Version: 3.22.1
- Visual Studio 2022 (64-bit) Version 17.0.4

Algunas otras configuraciones de interés de Qt para el proyecto:



Se ha utilizado la versión skeleton.zip suministrada a través de Prado. Para esta práctica finalmente se ha hecho uso solamente de la versión 3D ya que lo que se solicita es exclusivamente referente a la versión 3D, aunque se ha consultado la 2D para estudiarla como referencia y aprender para la 3D.

Respecto al código desarrollado, está disponible en un repositorio github público para su consulta, la URL es: <https://github.com/franguerrero/FGGC-practica1>

En dicho repositorio se puede ver la evolución que hemos llevado a cabo, dentro de la rama de MAIN.

Inicialmente se cargó el esqueleto tal cual ha sido descargado de Prado, se subió al control de versiones y por cada una de las funcionalidades solicitadas se ha creado una rama independiente donde se ha probado la funcionalidad requerida y una vez conforme con el resultado se hacía un merge a la rama principal de MAIN.

Así, por ejemplo, en primer lugar se ha desarrollado la funcionalidad del relleno, la cual se encuentra en la rama de “fillmode”. En ella se puede ver que el cambio en ese Branch ha consistido fundamentalmente en la implementación de la función draw_fill de la clase object3d.cc.

Basicamente se ha añadido una funcionalidad similar a draw_line pero donde el glPolygonMode se usa GL_FILL. También hemos querido cambiar el color que viene por defecto en el esqueleto

El código ha consistido en lo siguiente:



```
@@ -39,7 +39,15 @@ void _object3D::draw_line()
39      39
40      40      void _object3D::draw_fill()
41      41      {
42      -
42      +      glColor3d (1, 1., .7);
43      +      glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
44      +      glBegin(GL_TRIANGLES);
45      +      for (unsigned int i=0; i<Triangles.size(); i++){
46      +          glVertex3fv((GLfloat *) &Vertices[Triangles[i]._0]);
47      +          glVertex3fv((GLfloat *) &Vertices[Triangles[i]._1]);
48      +          glVertex3fv((GLfloat *) &Vertices[Triangles[i]._2]);
49      +      }
50      +      glEnd();
43      51      }
44      52
```

Posteriormente se ha procedido a realizar el Branch de chessmode para implementar la función de coloreado en distintos colores. Basicamente lo que se ha hecho es pintar de azul cada 2 triangulos empezando por el par y pintar de rojo cada 2 triangulos empezando por el impar. Se ha implementado la función draw_chess de la clase objetc3d.cc

```
59
60 void _object3D::draw_chess()
61 {
62     glColor3fv ((GLfloat *)&BLUE);
63     glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
64     glBegin(GL_TRIANGLES);
65     for (unsigned int i=0; i<Triangles.size(); i+=2){
66         glVertex3fv((GLfloat *) &Vertices[Triangles[i]._0]);
67         glVertex3fv((GLfloat *) &Vertices[Triangles[i]._1]);
68         glVertex3fv((GLfloat *) &Vertices[Triangles[i]._2]);
69     }
70     glEnd();
71     glColor3fv ((GLfloat *)&RED);
72     glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
73     glBegin(GL_TRIANGLES);
74     for (unsigned int i=1; i<Triangles.size(); i+=2){
75         glVertex3fv((GLfloat *) &Vertices[Triangles[i]._0]);
76         glVertex3fv((GLfloat *) &Vertices[Triangles[i]._1]);
77         glVertex3fv((GLfloat *) &Vertices[Triangles[i]._2]);
78     }
79     glEnd();
80 }
81
```

Por último, se ha procedido a la realización del Branch de cubeclass para implementar el cubo y que se visualice pulsando el 2 y podamos ver todos los modos anteriores del tetraedro (puntos, líneas, relleno y ajedrez).

Para ello, hemos añadido una clase cubo que hereda deobject3D, el cubo lo hemos formado con 8 vértices y 12 triángulos, teniendo en cuenta que las normales apunten hacia fuera de las caras del cubo. Luego hemos modificado el glwidget para contemplar que pulsando el 2 se cambie al cubo y que se pueda dibujar en los distintos modos.

Los principales cambios en este Branch para esta funcionalidad han sido:

- Crear el fichero de cabecera cube.h:

MASTER EN DESARROLLO DE SOFTWARE
Fundamentos de Geometría y Geometría Computacional

```
✓ 27 ■■■■■ skeleton/cube.h
...  ...  @@ -0,0 +1,27 @@
1  +  /*! \file
2  +  * Copyright Francisco Guerrero
3  +  * email: franciscoga@correo.ugr.es
4  +  * 2022
5  +  * GPL 3
6  +  */
7  +
8  +
9  + #ifndef CUBE_H
10 + #define CUBE_H
11 +
12 + #include "object3d.h"
13 +
14 +
15 + /*****
16 +  *
17 +  *
18 +  *
19 +  *****/
20 +
21 + class _cube:public _object3D
22 + {
23 + public:
24 +   _cube(float Size=1.0);
25 + };
26 +
27 + #endif
```

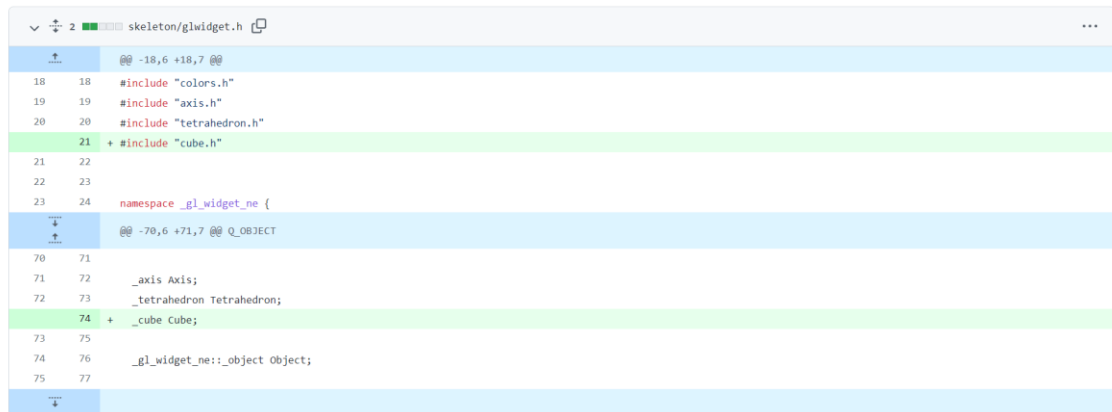
- Crear la clase cube.cc que implementa el cubo con sus 8 vértices y los 12 triángulos:

MASTER EN DESARROLLO DE SOFTWARE
Fundamentos de Geometría y Geometría Computacional

```
49 skeleton/cube.cc
...  ... @@ -0,0 +1,49 @@
1 + /*! \file
2 + * Copyright Francisco Guerrero
3 + * email: franciscoga@correo.ugr.es
4 + * 2022
5 + * GPL 3
6 + */
7 +
8 +
9 + #include "cube.h"
10 +
11 +
12 + /*****
13 + *
14 + * Clase cubo para la practica 1.
15 + * El cubo tiene 8 vertices que forman 6 caras por lo que vamos a tener 12 triangulos
16 + *
17 + *****/
18 +
19 + _cube::_cube(float Size)
20 + {
21 +     Vertices.resize(8);
22 +
23 +     Vertices[0]=_vertex3f(-Size/2,-Size/2,-Size/2);
24 +     Vertices[1]=_vertex3f(Size/2,-Size/2,-Size/2);
25 +     Vertices[2]=_vertex3f(Size/2,Size/2,-Size/2);
26 +     Vertices[3]=_vertex3f(-Size/2,Size/2,-Size/2);
27 +     Vertices[4]=_vertex3f(-Size/2,-Size/2,Size/2);
28 +     Vertices[5]=_vertex3f(Size/2,-Size/2,Size/2);
29 +     Vertices[6]=_vertex3f(Size/2,Size/2,Size/2);
30 +     Vertices[7]=_vertex3f(-Size/2,Size/2,Size/2);
31 +
32 +     Triangles.resize(12);
33 +
34 +     Triangles[0]=_vertex3ui(0,2,1);
35 +     Triangles[1]=_vertex3ui(0,3,2);
36 +     Triangles[2]=_vertex3ui(0,4,3);
37 +     Triangles[3]=_vertex3ui(4,7,3);
38 +
39 +     Triangles[4]=_vertex3ui(0,1,4);
40 +     Triangles[5]=_vertex3ui(4,1,5);
41 +     Triangles[6]=_vertex3ui(3,7,2);
42 +     Triangles[7]=_vertex3ui(7,6,2);
43 +
44 +     Triangles[8]=_vertex3ui(4,5,6);
45 +     Triangles[9]=_vertex3ui(7,4,6);
46 +     Triangles[10]=_vertex3ui(5,1,2);
47 +     Triangles[11]=_vertex3ui(2,6,5);
48 + }
49 +
```

- Modificar la cabecera de glwidget.h para incluir la clase del cubo:

MASTER EN DESARROLLO DE SOFTWARE
Fundamentos de Geometría y Geometría Computacional



```
2 skeleton/glwidget.h
18 18 #include "colors.h"
19 19 #include "axis.h"
20 20 #include "tetrahedron.h"
21 21 + #include "cube.h"
22 22
23 23 namespace _gl_widget_ne {
24 24
70 71
71 72     _axis Axis;
72 73     _tetrahedron Tetrahedron;
74 74 + _cube Cube;
73 75
74 76     _gl_widget_ne::object Object;
75 77
```

- Modificar la implementación de glwidget para tener en cuenta el caso del cubo para los distintos modos de pintado (puntos, líneas, relleno y ajedrez):

MASTER EN DESARROLLO DE SOFTWARE
Fundamentos de Geometría y Geometría Computacional

```

✓ 4 skeleton/glwidget.cc
↑... @@ -125,6 +125,7 @@ void _gl_widget::draw_objects()
125 125     glColor3fv((GLfloat *) &BLACK);
126 126     switch (Object){
127 127         case OBJECT_TETRAHEDRON:Tetrahedron.draw_point();break;
128 +     case OBJECT_CUBE:Cube.draw_point();break;
128 129         default:break;
129 130     }
130 131 }

↑... @@ -134,6 +135,7 @@ void _gl_widget::draw_objects()
134 135     glColor3fv((GLfloat *) &MAGENTA);
135 136     switch (Object){
136 137         case OBJECT_TETRAHEDRON:Tetrahedron.draw_line();break;
138 +     case OBJECT_CUBE:Cube.draw_line();break;
137 139         default:break;
138 140     }
139 141 }

↑... @@ -142,13 +144,15 @@ void _gl_widget::draw_objects()
142 144     glColor3fv((GLfloat *) &BLUE);
143 145     switch (Object){
144 146         case OBJECT_TETRAHEDRON:Tetrahedron.draw_fill();break;
147 +     case OBJECT_CUBE:Cube.draw_fill();break;
145 148         default:break;
146 149     }
147 150 }
148 151
149 152     if (Draw_chess){
150 153         switch (Object){
151 154             case OBJECT_TETRAHEDRON:Tetrahedron.draw_chess();break;
155 +         case OBJECT_CUBE:Cube.draw_chess();break;
152 156             default:break;
153 157         }
154 158     }

↓

```

- Modificar el fichero *.pro para incluir el header y la fuente de la nueva clase de cubo

MASTER EN DESARROLLO DE SOFTWARE
Fundamentos de Geometría y Geometría Computacional

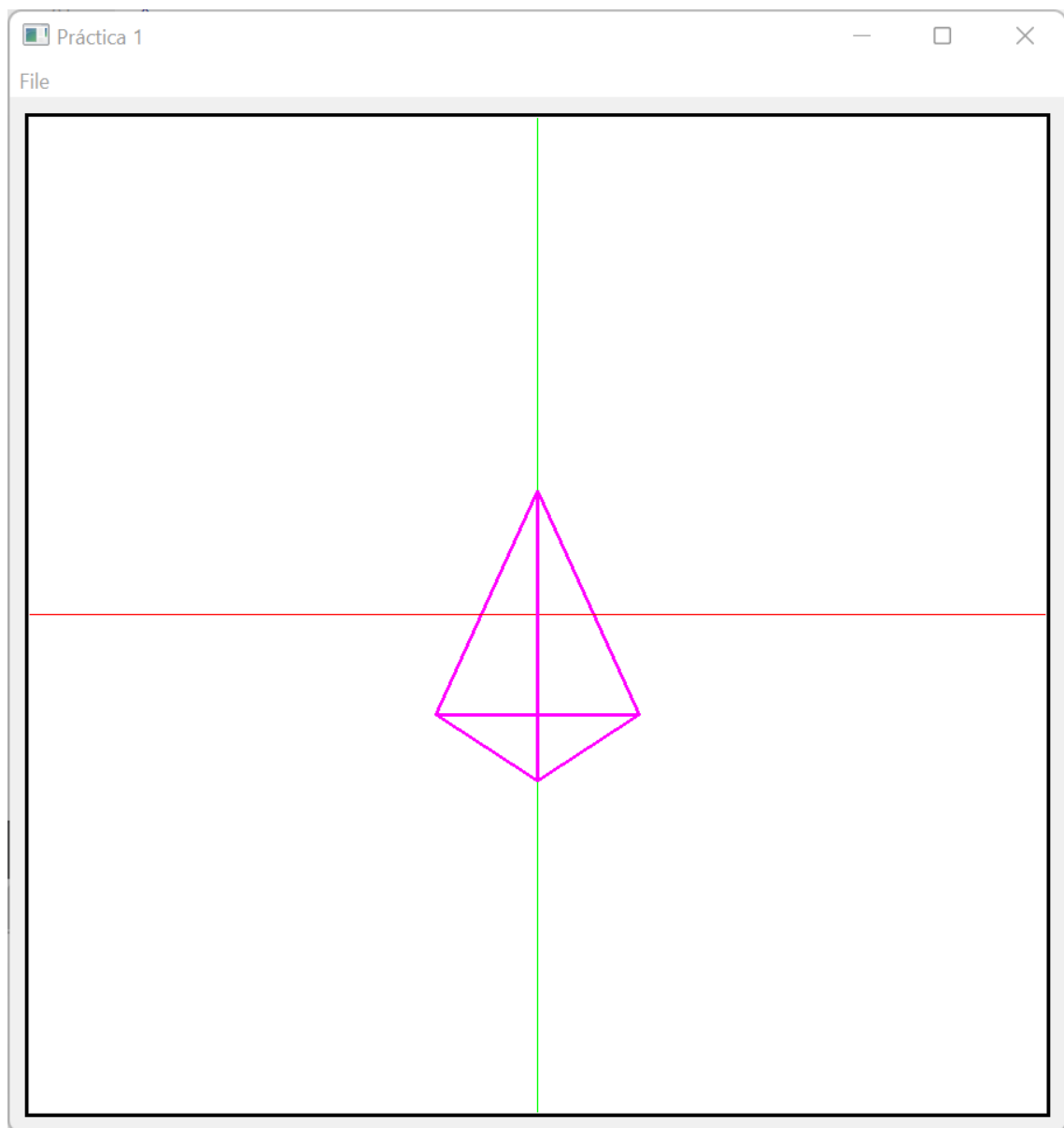
```
✓ ↕ 2 ■■■ skeleton/skeleton_qt.pro 📄  
↑ ..... @@ -4,6 +4,7 @@ DEFINES = WINDOWS  
4 4 HEADERS += \  
5 5 colors.h \  
6 6 basic_object3d.h \  
7 + cube.h \  
7 8 object3d.h \  
8 9 axis.h \  
9 10 tetrahedron.h \  
↕ ..... @@ -12,6 +13,7 @@ HEADERS += \  
12 13  
13 14 SOURCES += \  
14 15 basic_object3d.cc \  
16 + cube.cc \  
15 17 object3d.cc \  
16 18 axis.cc \  
17 19 tetrahedron.cc \  
↓ .....
```

Finalmente un vez comprobado los resultados, se ha hecho el merge a la rama MAIN se posteriormente se ha añadido un fichero de README y se ha eliminado el esqueleto de 2D que no se pide en la tarea 1 como parte de los resultados.

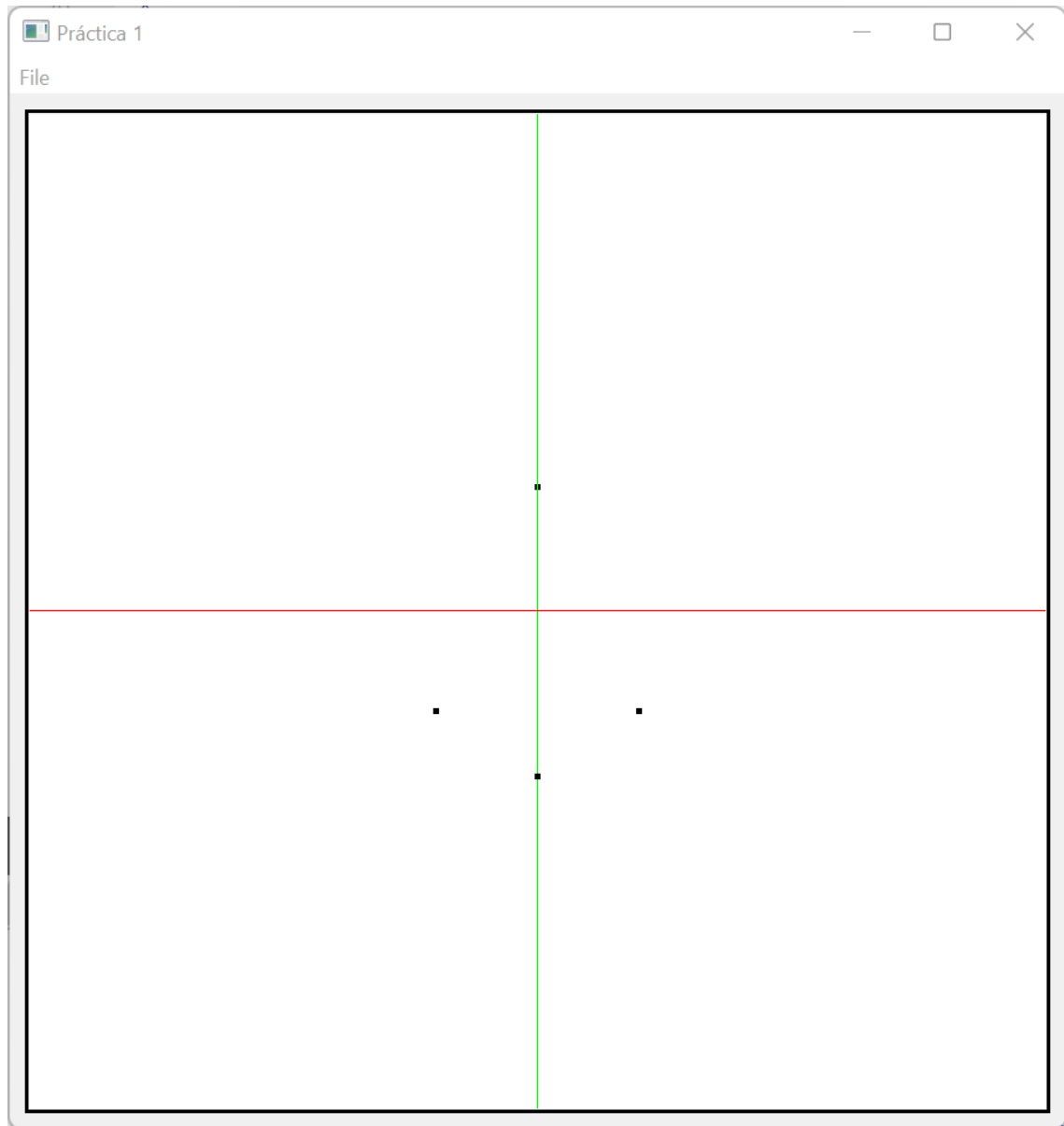
3. Resultados

Vamos a mostrar los distintos resultados que se han obtenido una vez implementado lo que se solicita y se ejecuta la aplicación:

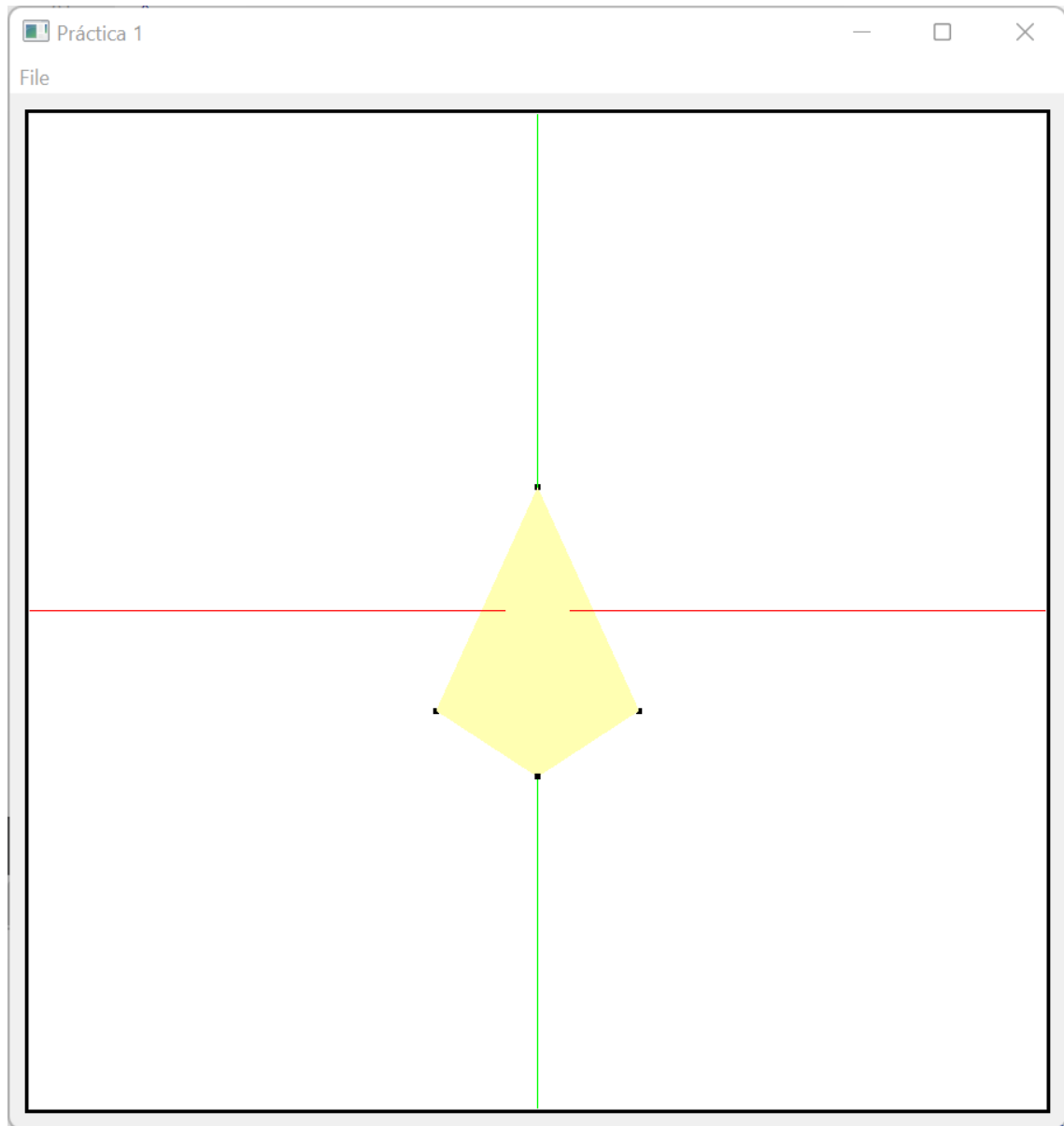
En primer lugar vemos el tetraedro con el dibujo de linea:



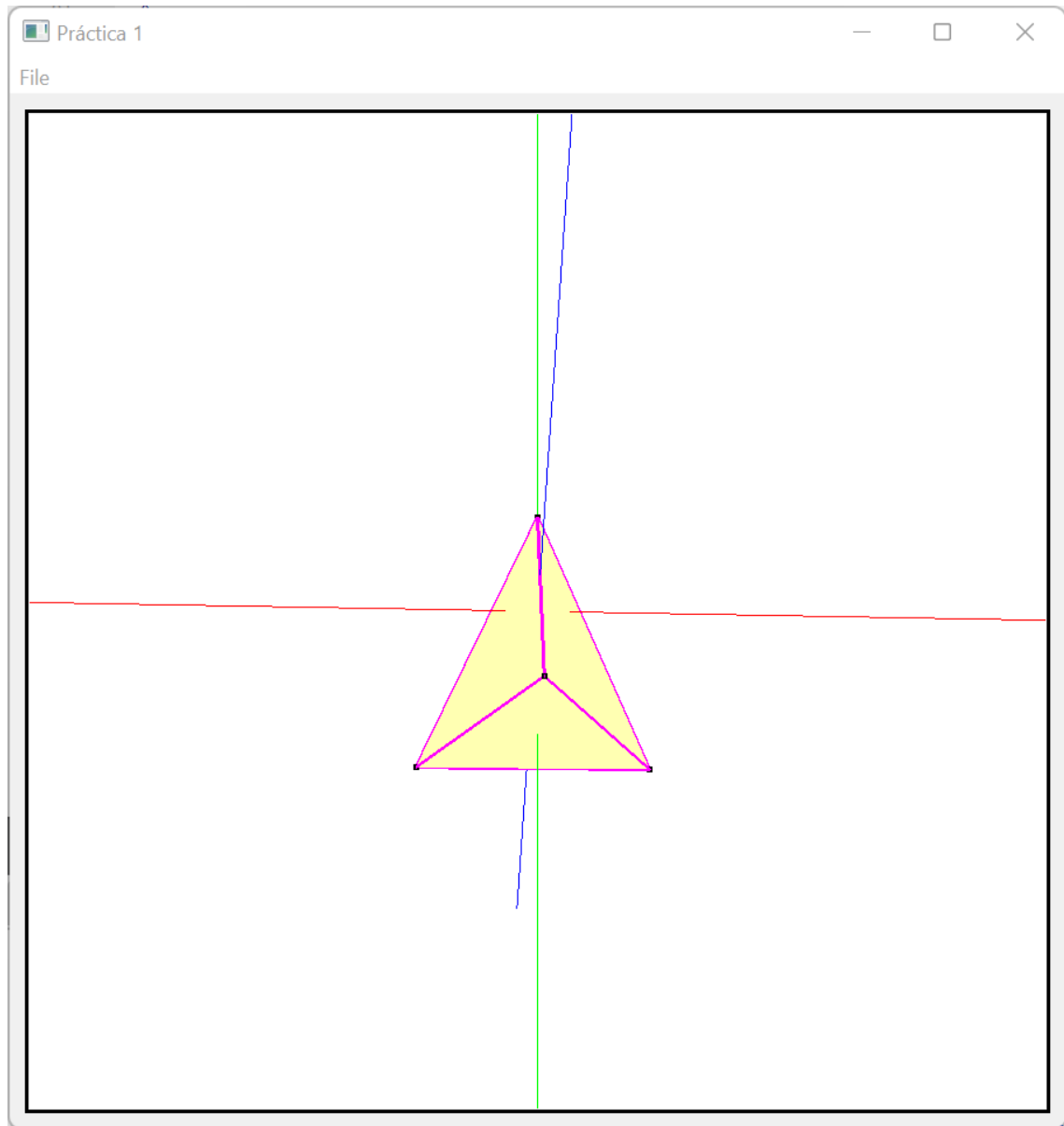
Si pulsamos “p” para que se visualicen los puntos y pulsamos “l” para que se dejen de visualizar las líneas veremos lo siguiente:



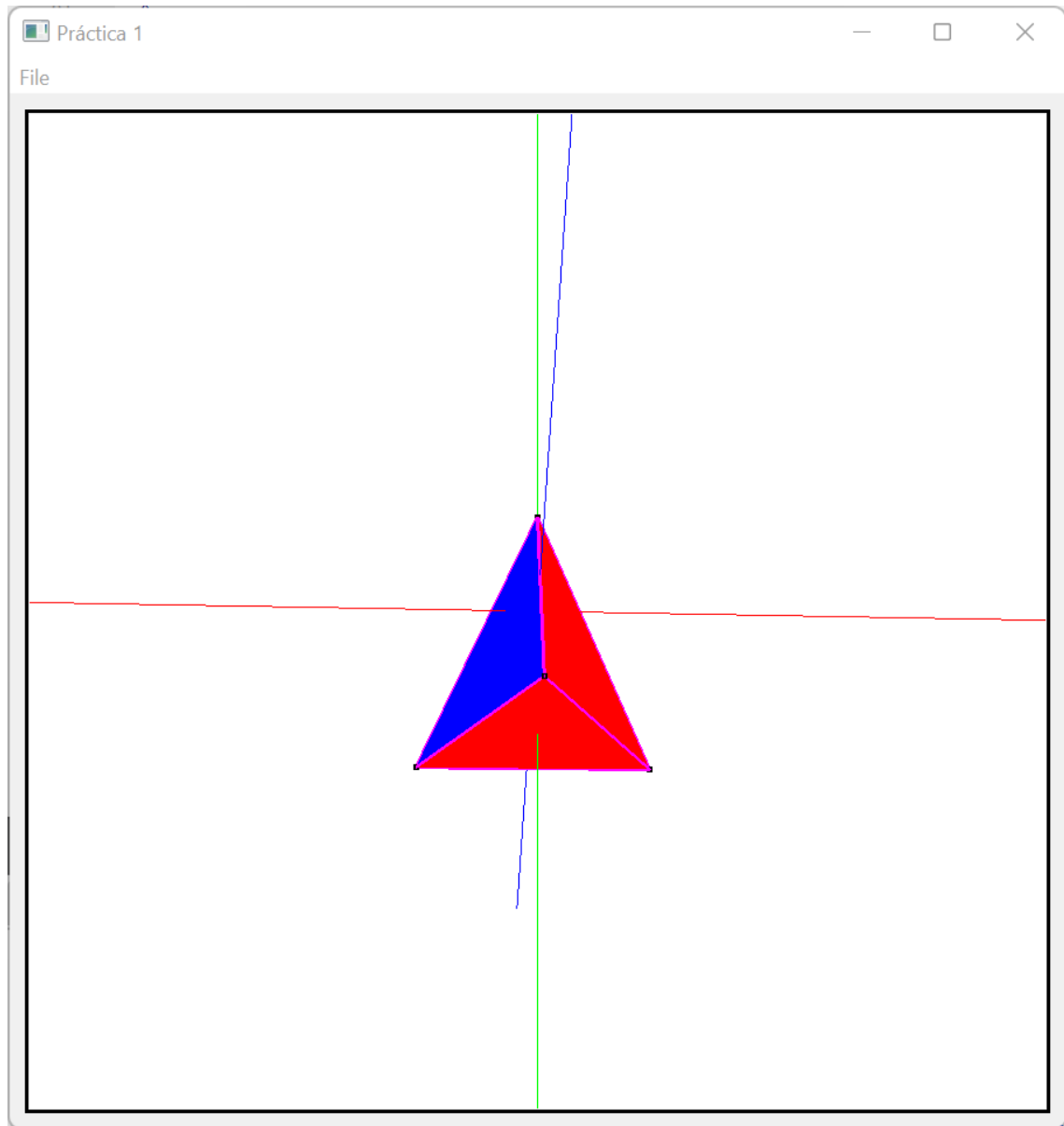
Si ahora pulsamos “f” veremos el modo relleno (y aun los puntos si no hemos pulsado “p” para quitarlos):



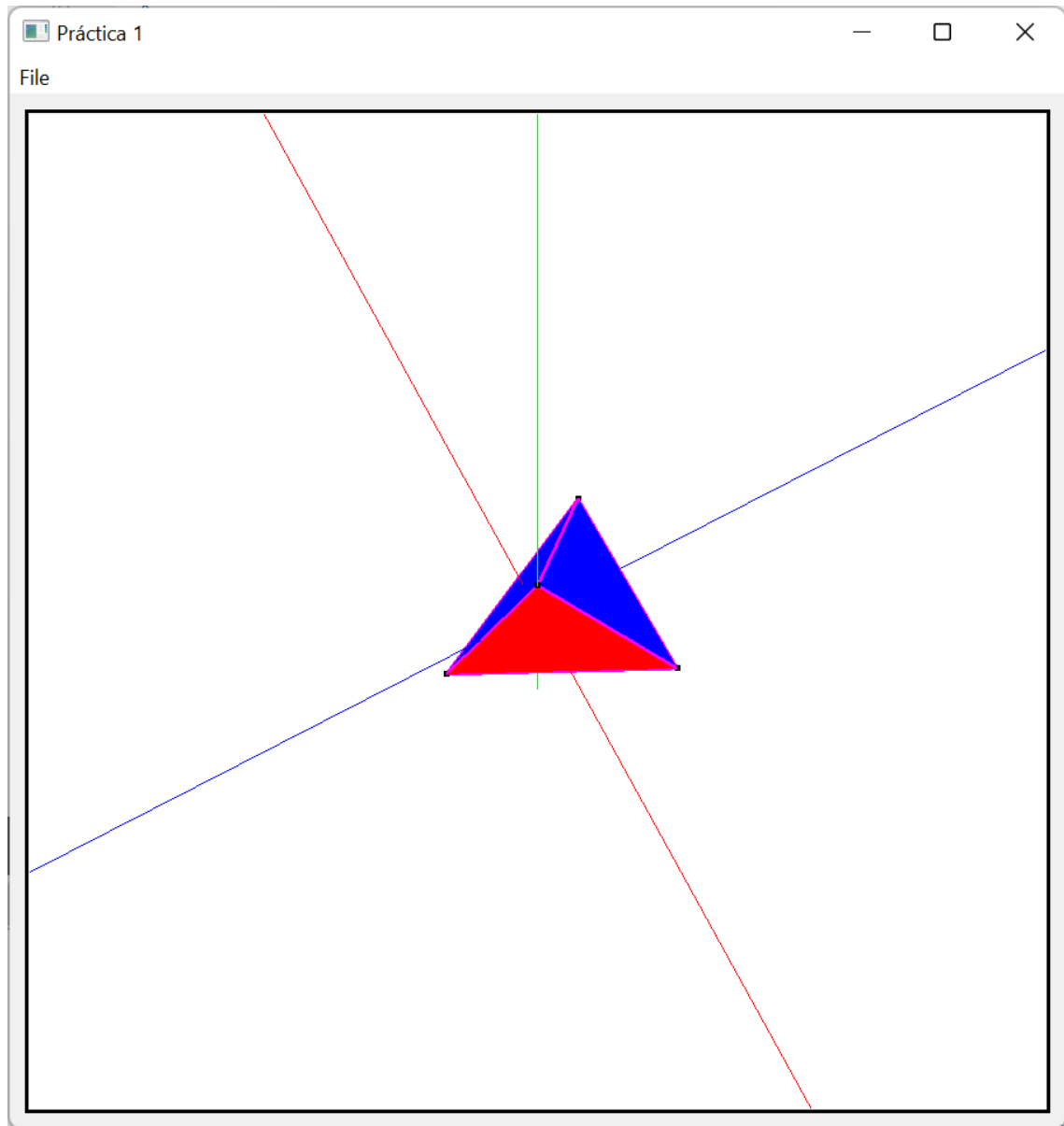
Vamos ahora a visualizar las líneas pulsando de nuevo “I” y a girar y rotar el tetraedro con las flechas del teclado hasta que visualizamos esto:



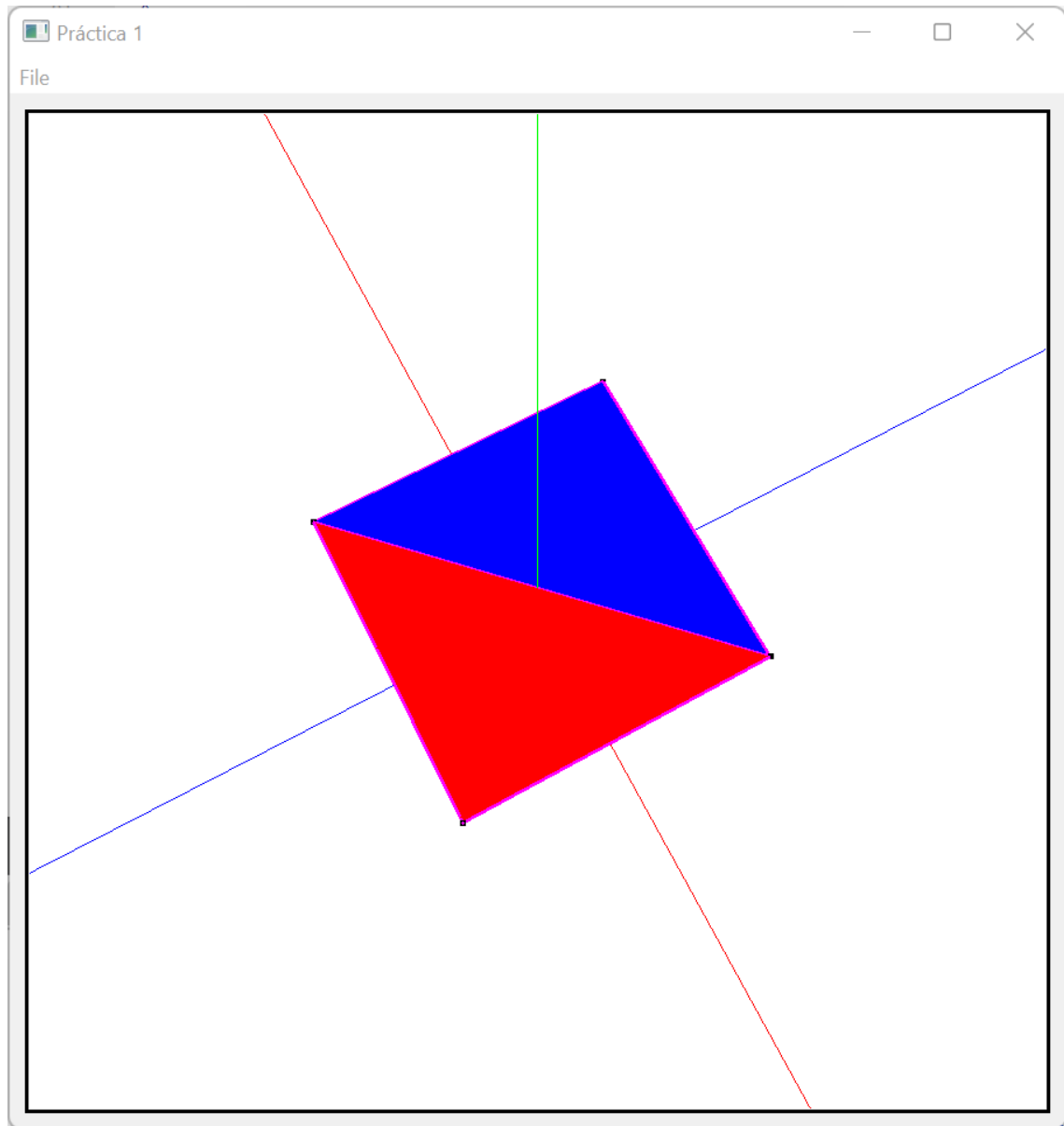
Para visualizar el modo ajedrez, pulsamos “c” (nos aseguramos que el modo “f” está quitado) y veremos algo como esto:



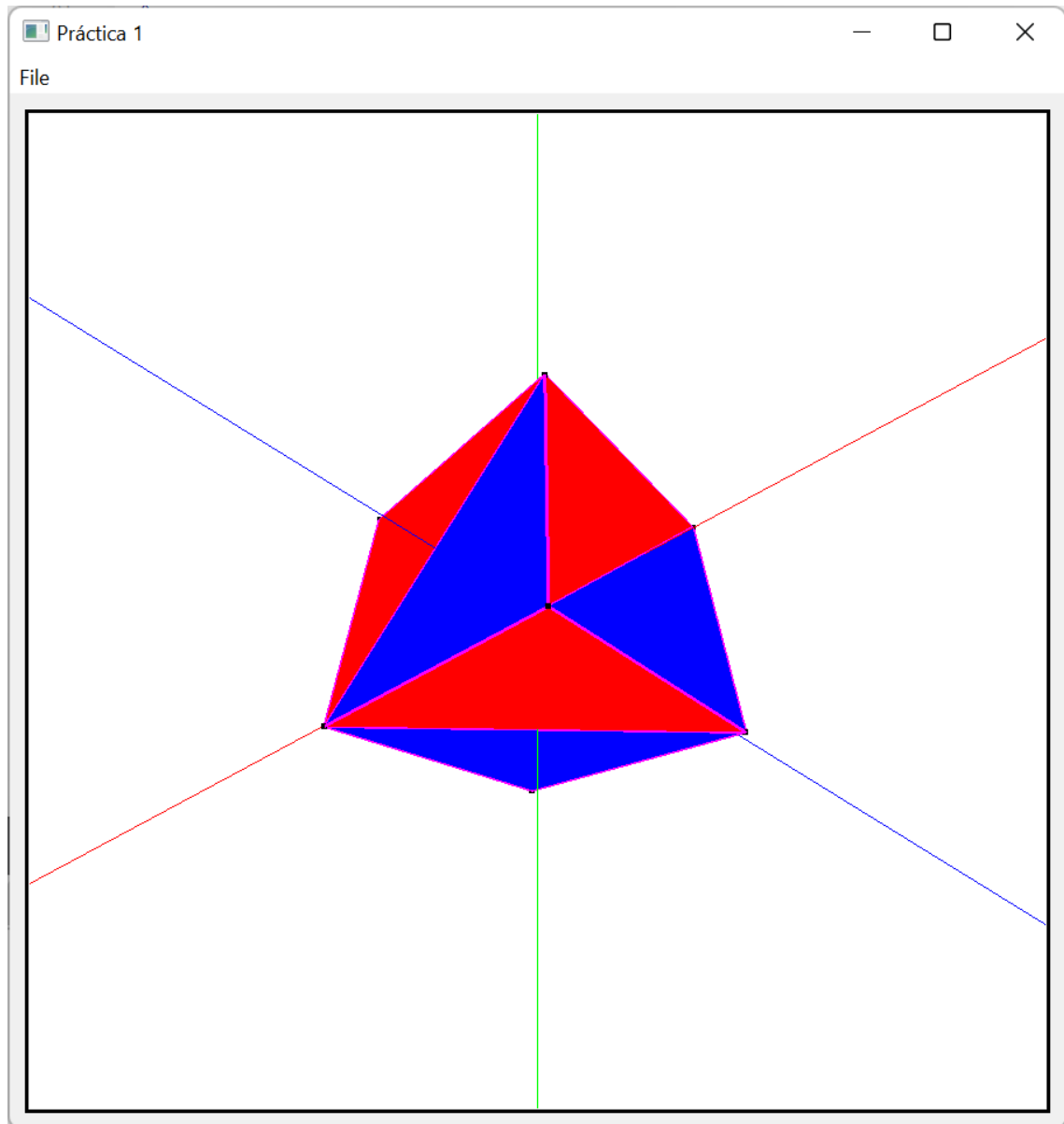
Si lo rotamos, vemos como 2 triángulos del tetraedro están pintados en rojo y 2 en azul:



Si ahora pulsamos el “2” pasaremos a ver el cubo en modo ajedrez:

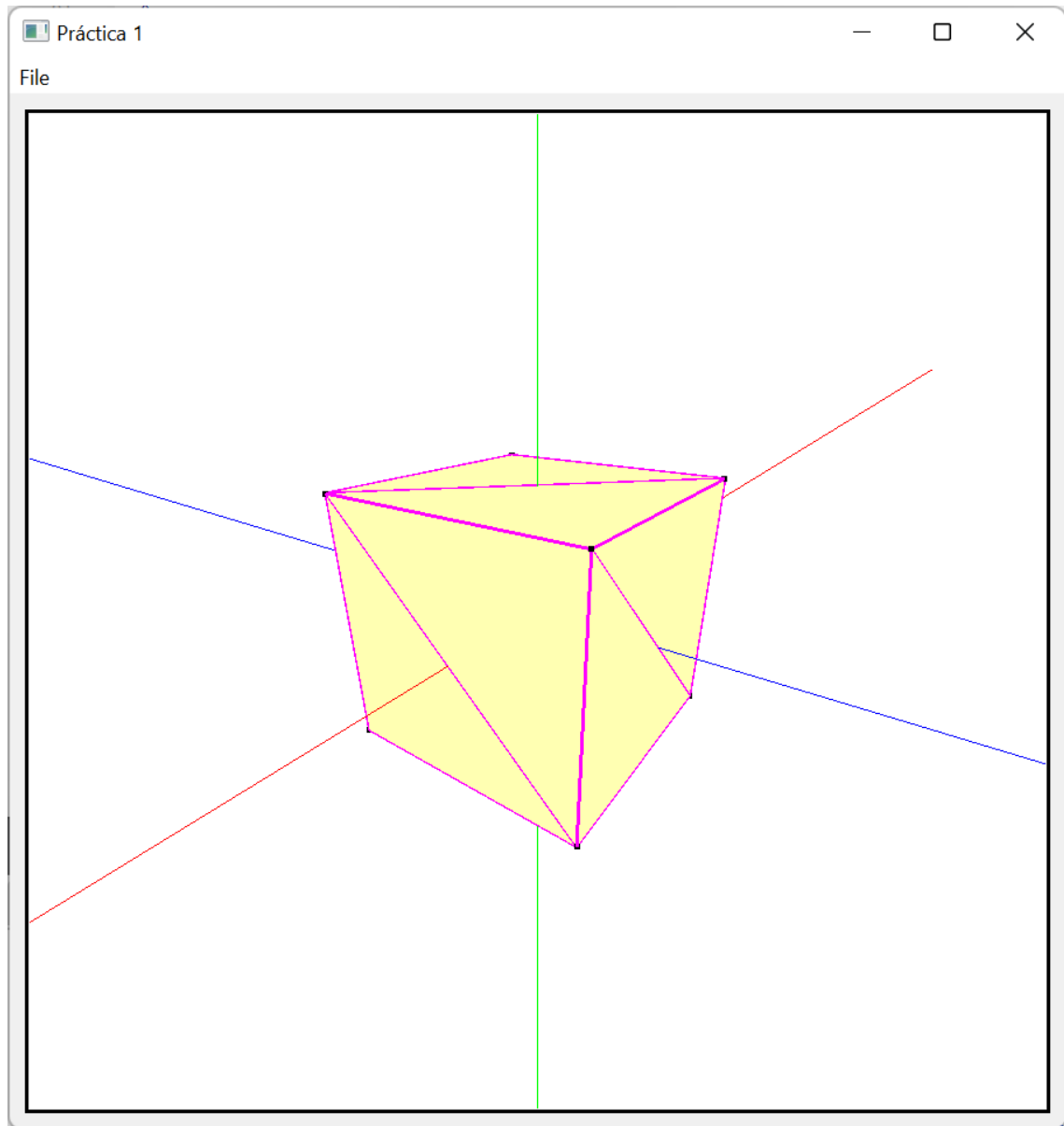


Lo rotamos para ver que efectivamente se trata de un cubo:

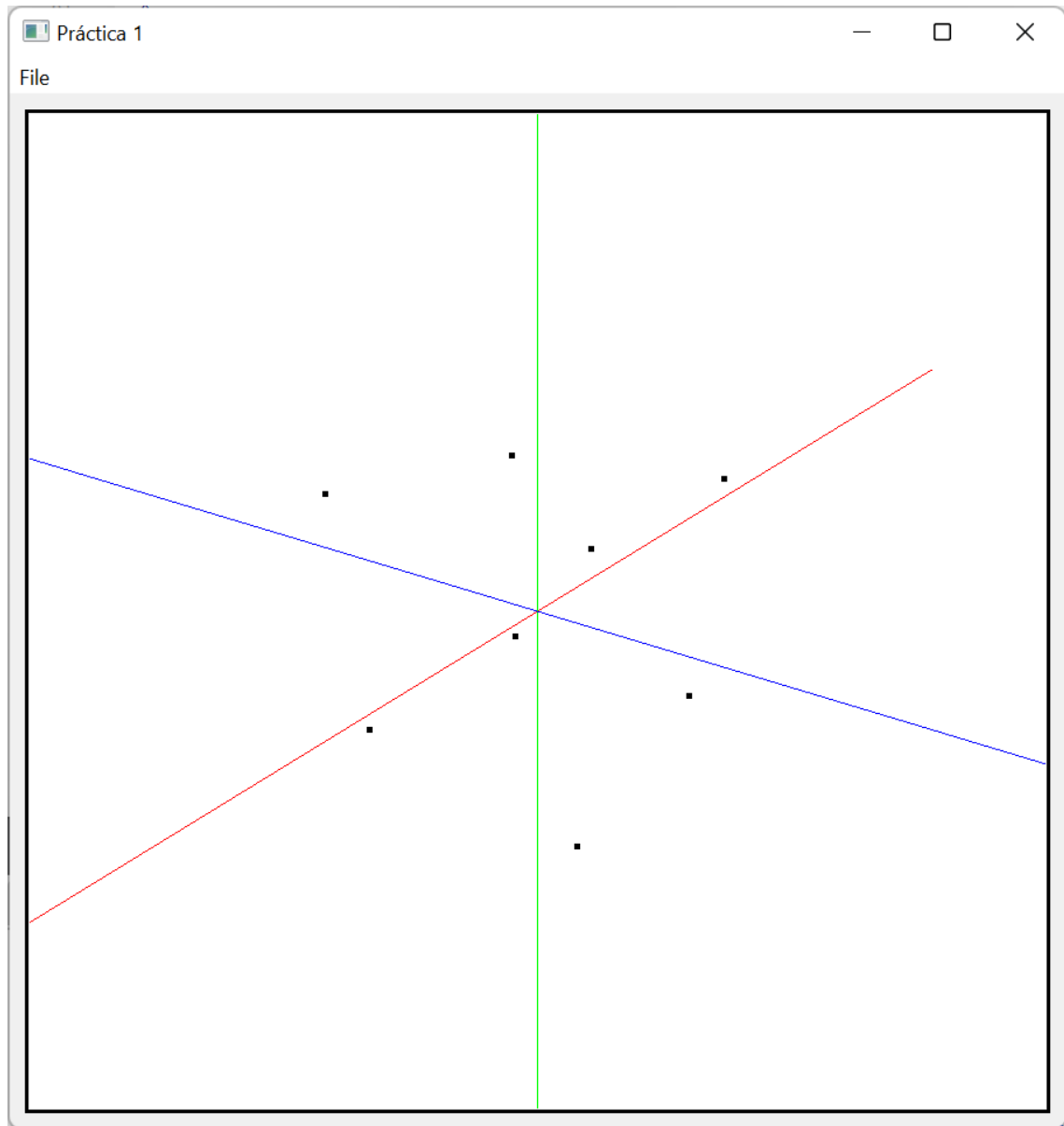


Y podemos jugar a ver las distintas representaciones con líneas, rellenos y puntos pulsando “c” para desactivar el ajedrez y “l”, “f” y/o “p”. Algunas representaciones son:

MASTER EN DESARROLLO DE SOFTWARE
Fundamentos de Geometría y Geometría Computacional



MASTER EN DESARROLLO DE SOFTWARE
Fundamentos de Geometría y Geometría Computacional



MASTER EN DESARROLLO DE SOFTWARE
Fundamentos de Geometría y Geometría Computacional

