UNIVERSIDAD DE GUADALAJARA

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS DIVISIÓN DE ELECTRÓNICA Y COMPUTACIÓN

DEPARTAMENTO DE CIENCIAS COMPUTACIONALES

INGENIERÍA EN COMPUTACIÓN

SIMULACION POR COMPUTADORA

CLAVE: I7042

Proyecto Final

Nombre: Elizalde- Loera Felipe de Jesus

Codigo: 211715281

Contenido

Introducción

La proyeccion perspectiva nos deja ver objetas en diferentes tamaños, entre mas lejos mas pequeño y entre mas cercas mas grande. La proyeccion te permite ver un objeto en 2D como si lo estuvieras viendo apartir de una camara. La proyeccion sera el metodo que estaremos utilizando en el proyecto final. Estaremos proyectando un proyectil y rotando el proyectil en unas de las ejes ya sea X,Y o Z.

Proyección

La proyeccion como ya se habia mencionado es la perspectiva de la camara. La posicion de la camara la orientacion controlan la transformación de la proyección. Por ejemplo si proyectamos un cubo con dos piramides a los lados se ve de la siguiente manera cuando la camara esta en la posicion 0,0,1000 y orientacion 0,0,0.

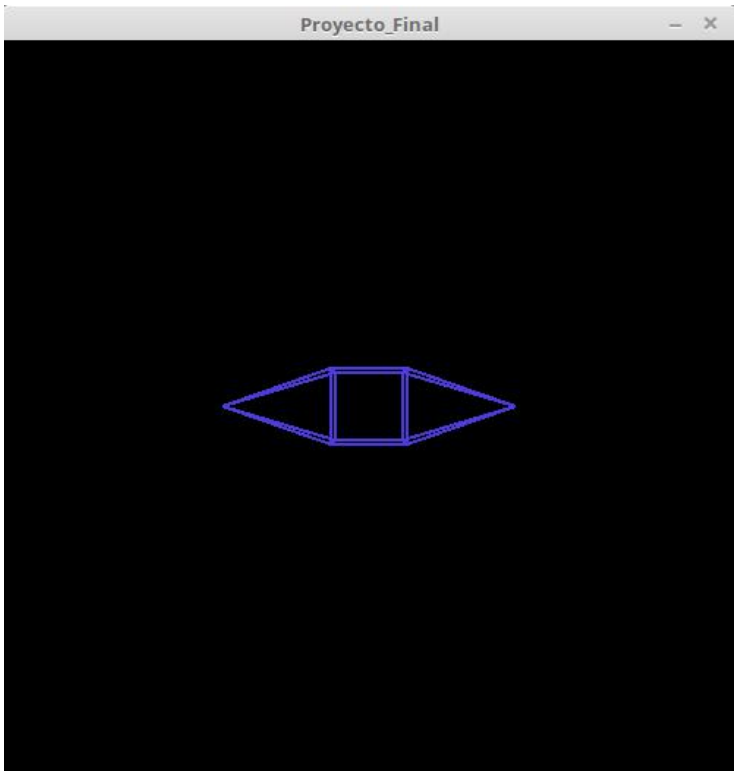


Figura 1: Posicion de Camara(0,0,1000)

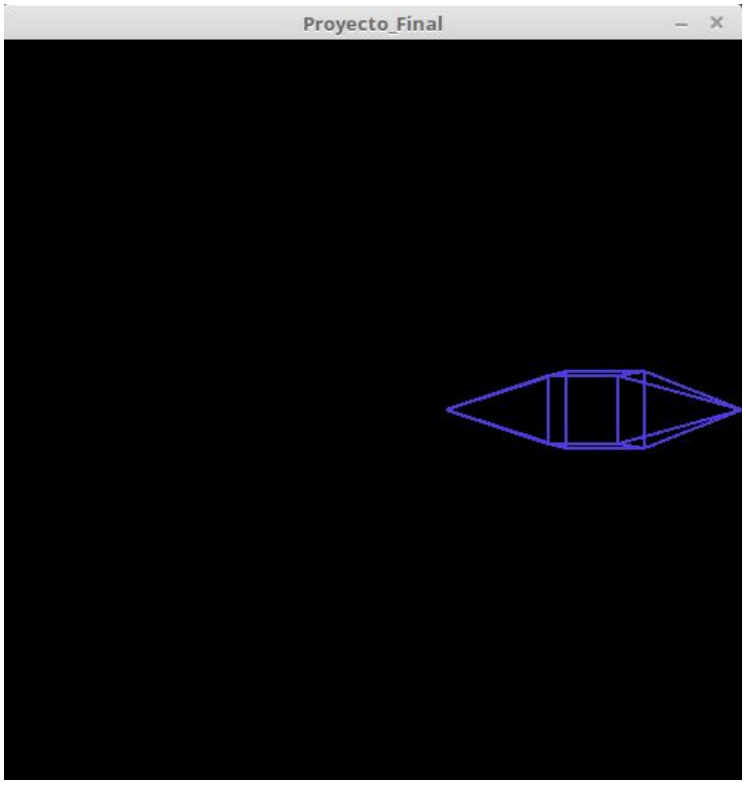


Figura 2: Posicion de Camara (300,0,1000)

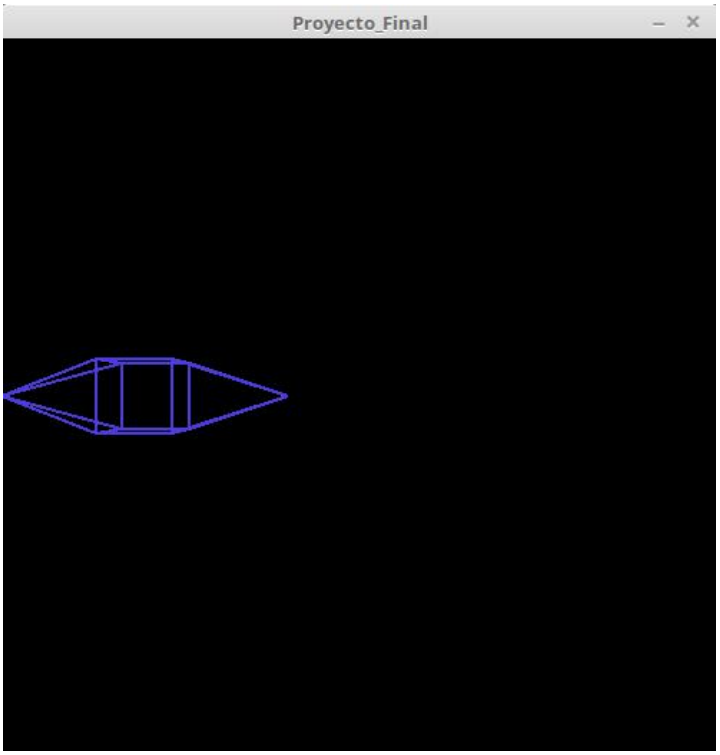


Figura 3: Posicion de Camara (-300,0,1000)

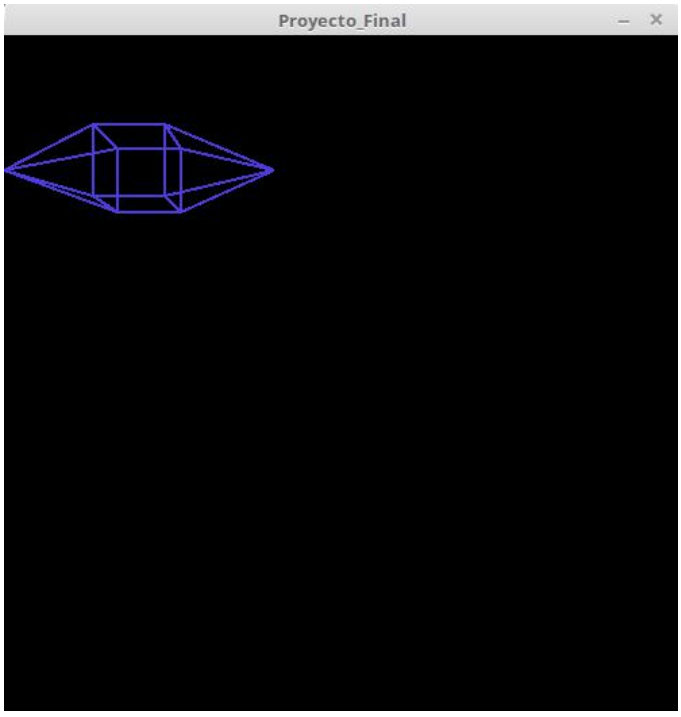


Figura 4: Posicion de Camara(-300,300,100)

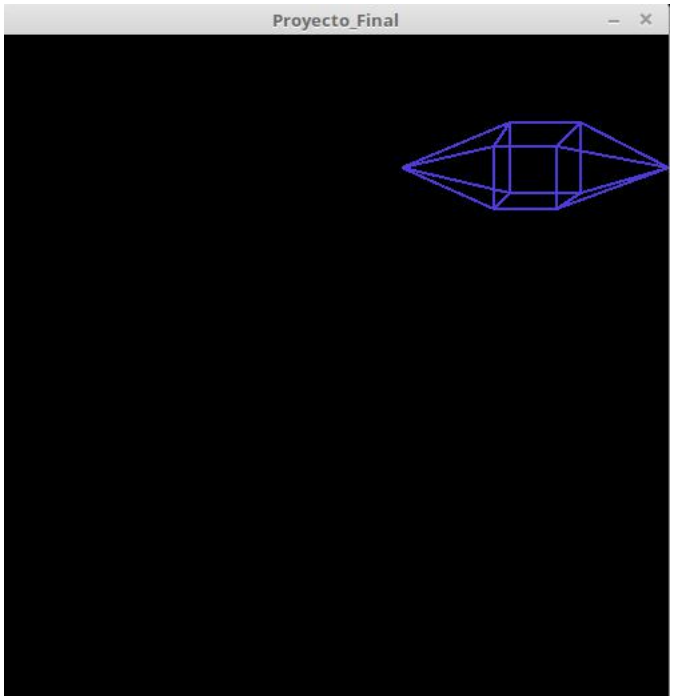


Figura 5: Posición de Camara (300,300,100)

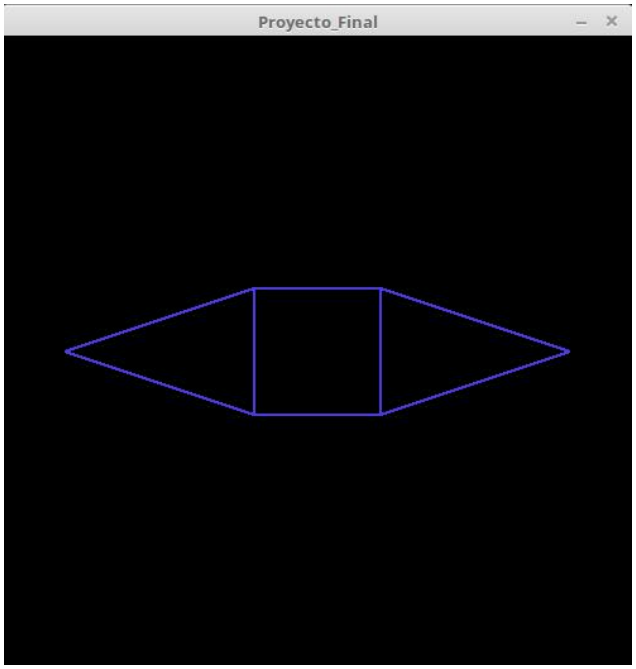


Figura 6: Objecto sin proyección

Como vieron la proyección nos dibuja una figura en 3d con la perspectiva de la camara. Ya dependera de la posición y orientacion de la camara obtener diferentes vistas del objeta. Tambien se podra ver la figura cuando no se hace proyección. Se ve que es una figura de dos dimensiones pero en realidad es de 3D en memoria. Lo que sucede aqui es que el entorno grafica que estamos utilizando no es capaz de manejar las 3 dimension, no mas dos. Para solucionar este problema agregamos el eje de Z a todos los puntos y creamos un figura tridimenisonal. Aplicamos la proyeccion a todos los puntos y obtenemos nuevos puntos que simulan el efecto de 3d o la perspectiva de la camara.

Calcular la Proyección Perspectiva

Para calcular el proyeccion perspectiva se ocupa las siguientes variables:

Points: Es un punto o conjunto de puntos. Estos son los puntos que seran proyectados.
Camara: Posicion de donde se encuentra la camara.
Theta: Es la orientacion de la camara.
Vista: Es la vista relativa al display.

Cada una de estas variables tiene los ejes de **x,y,z.**

Para calcular la proyección se utilizara la siguiente formula:

$$\begin{bmatrix} \mathbf{d}_x \\ \mathbf{d}_y \\ \mathbf{d}_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_x) & \sin(\theta_x) \\ 0 & -\sin(\theta_x) & \cos(\theta_x) \end{bmatrix} \begin{bmatrix} \cos(\theta_y) & 0 & -\sin(\theta_y) \\ 0 & 1 & 0 \\ \sin(\theta_y) & 0 & \cos(\theta_y) \end{bmatrix} \begin{bmatrix} \cos(\theta_z) & \sin(\theta_z) & 0 \\ -\sin(\theta_z) & \cos(\theta_z) & 0 \\ 0 & 0 & 1 \end{bmatrix} \left(\begin{bmatrix} \mathbf{a}_x \\ \mathbf{a}_y \\ \mathbf{a}_z \end{bmatrix} - \begin{bmatrix} \mathbf{c}_x \\ \mathbf{c}_y \\ \mathbf{c}_z \end{bmatrix} \right)$$

El **a** es el punto actual, **c** es la camara. El **θ** es thetha. Entonces utilizando esta formula te debera de generar un nuevo punto que los llamaremos **d.**

Despues utilizamos la siguiente formula, en donde **e** es la vista del display.

$$\begin{aligned} \mathbf{b}_x &= \frac{\mathbf{e}_z}{\mathbf{d}_z}\mathbf{d}_x - \mathbf{e}_x \\ \mathbf{b}_y &= \frac{\mathbf{e}_z}{\mathbf{d}_z}\mathbf{d}_y - \mathbf{e}_y \end{aligned}.$$

Bx y **By** es el punto original proyectado en 2d. Basicamente convertivmos un punto 3d a 2d. Esto lo tendremos que hacer para cada punto para obtener una figura 3d proyectada a 2d.

Calcular la Rotacion de 3 dimensiones

Para calcular la rotaciones de 3 dimensiones se utilizan la siguientes matrices.

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Tendras que multiplicar cada punto de la figura con unas de estas matrices, ya dependera de que rotación se desea. Esto se puede ver es las siguientes matrices.

1	0	0
0	Cos**θ**	-sin**θ**
0	Sin**θ**	cos**θ**

x
Y
Z

Se puede ver la rotacion en x en la siguiente figura:

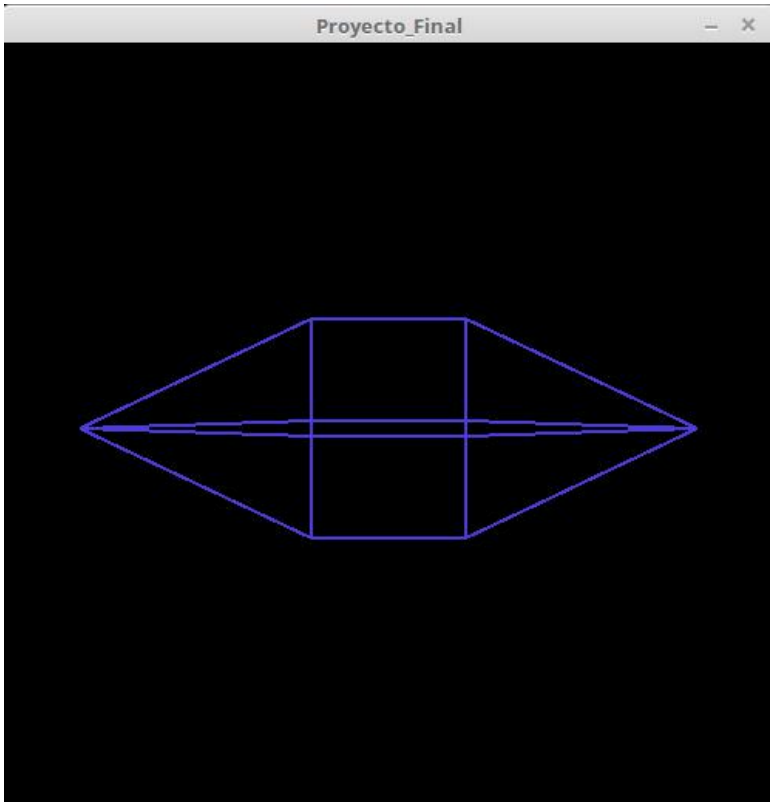


Figura 7: Rotacion en el eje X.

Desarrollo de la aplicación

Para solucionar este problema se utilizo el lenguaje de programación c++ con la biblioteca de Allegro 5 para manejar los graficos 2d. Al momento de iniciar el programa ya tenemos predeterminado la figura que se va a crear en una funcion llamada createFigure(). Ya lo unico de debemos que hacer es utilizar el metodo de proyeccion perspectiva para obtener los puntos de proyeccion en 2d para cada punto. Pero antes de hacer la proyeccion rotamos la figura y despues proyectamos para obtener una mejor visualizacion de como funcion la proyección.

Conclusión

Despues de haber realizado la actividad vi que depende mucho la posición de la vista de display y la posición de la camara para tener un buena visualización de la figura porque se puede ver momentos en donde la camara esta dentro de la figura y no mas puedes ver lineas de la figura. Por ultimo tenemos que recordar que la proyección perspectiva es la conversion de 3d a 2d como si el objeto es siendo visto por una camara.

Apéndice 1 (Codigo)

Header Files

```
#include <iostream>
#include <math.h>
#include <fstream>
#include "Points.h"
class Geo_Func
{
private:
    static std::vector<Points> points;
    static std::vector<Points> tempP;
    static std::vector<Points> camaraPoints;
    static float perspectiveM[][4];
    static float cx,cy,cz;
    static float thetaX,thetaY,thetaZ;
    static float ex,ey,ez;
    static int rx;
    static int tracker;
    static int counter;
    static bool dirX,dirY;
    static int middleX;
    static int middleY;
    static int middleZ;
    static int start;
    static int end;
    static float fuerza;
public:
    Geo_Func();
    static void addPoint(int&,int&,int&);
    static std::vector<Points> getPoints();
    static std::vector<Points> getCamaraPoints();
    static void projection();
    static void createFigure();
    static double degToRad(double);
    static void resetP();
    static void rotateX();
    static void rotateY();
    static void rotateZ();
    static void rotateCamaraY();
    static void loadCamaraPoints();
    static void moveObject();
    static void move_to_origin();
    static void moveBack();


};

```

```h
#ifndef INIT_ALLEGRO_H
#define INIT_ALLEGRO_H

#include <allegro5/allegro.h>

#include <iostream>

class Init_Allegro
{
private:
    static bool execute;
public:
    Init_Allegro(void);

    ALLEGRO_DISPLAY * display;
    ALLEGRO_COLOR blue = al_map_rgb(78,60,220);
    ALLEGRO_COLOR red = al_map_rgb(200,50,70);
    ALLEGRO_EVENT_QUEUE *eventqueue = NULL;
    ALLEGRO_TIMER *timer = NULL;
    void inicialize(void);
    int init_display(void);

};

#endif // INIT_ALLEGRO_H
```

```h
#ifndef KEYS_H
#define KEYS_H

#include "allegro5/allegro.h"
#include "Geometary_Functions.h"
#include <iostream>

class Keys
{
private:

public:
    Keys();
    static int x,y,z;
    static bool checkInputs(ALLEGRO_EVENT&);
};

#endif // KEYS_H
```

```h
#ifndef POINTS_H
#define POINTS_H


class Points
{
private:
    int x;
    int y;
    int z;
public:
    Points(int,int,int);
    int getX();
    int getY();
    int getZ();
    void setX(int);
    void setY(int);
    void setZ(int);
};

#endif // POINTS_H
```

Cpp Files

```cpp
#include "Geometary_Functions.h"

float Geo_Func::cx = 0;
float Geo_Func::cy = 0;
float Geo_Func::cz =  1000;
float Geo_Func::thetaX = 0;
float Geo_Func::thetaY =    0;
float Geo_Func::thetaZ = 00;
float Geo_Func::ex = 0;
float Geo_Func::ey = 0;
float Geo_Func::ez = 500;
int Geo_Func::rx = 1;
int Geo_Func::tracker = 0;
int Geo_Func::counter = 0;
int Geo_Func::middleX = 0;
int Geo_Func::middleY = 0;
int Geo_Func::middleZ = 0;
int Geo_Func::start = 516;
int Geo_Func::end = 1548;
float Geo_Func::fuerza = 1.3;

bool Geo_Func::dirX = true;
bool Geo_Func::dirY = false;
std::vector<Points> Geo_Func::points;
std::vector<Points> Geo_Func::tempP;
std::vector<Points> Geo_Func::camaraPoints;
Geo_Func::Geo_Func()
{

}
void Geo_Func::addPoint(int &x, int &y,int &z)
{
    points.push_back(Points(x-250,(y-250)*-1,z));
    std::cout << "X: " << x << "\n";
    std::cout << "Y: " << y << "\n";
    std::cout << "Z: " << z << "\n";
}

std::vector<Points> Geo_Func::getPoints()
{
    return points;
}
```

```
43  std::vector<Points>Geo_Func::getCamaraPoints()
44  {
45      return camaraPoints;
46  }
47  void Geo_Func::resetP()
48  {
49      points = tempP;
50  }
51  void Geo_Func::projection()
52  {
53      std::vector<Points> temp;
54      for(Points value:points)
55      {
56          double x = value.getX() - cx;
57          double y = value.getY() - cy;
58          start++;
59          double z = value.getZ() - cz;
60          double tx = degToRad(thetaX);
61          double ty = degToRad(thetaY);
62          double tz = degToRad(thetaZ);
63
64         double dx = cos( ty ) * ( sin( tz ) * y
65                   + cos( tz ) * x )
66                   - sin( ty ) * z;
67
68         double dy = sin( tx ) * ( cos( ty ) * z
69                   + sin( ty ) * ( sin( tz ) * y
70                   + cos( tz ) * x))
71                   + cos( tx ) * ( cos( tz ) * y
72                   - sin( tz ) * x);
73
74         double dz = cos( tx ) * ( cos( ty ) * z
75                   + sin( ty ) * ( sin( tz ) * y
76                   + cos( tz ) * x))
77                   + sin( tx ) * ( cos( tz ) * y
78                   - sin( tz ) * x);
79
80           int bx = (ez/dz*dx + ex);
81           int by = (ez/dz*dy + ey);
82          temp.push_back(Points(bx,by,0));
```

```
83
84          std::cout << "X: " << value.getX() << std::endl;
85          std::cout << "Y: " << value.getY() << std::endl;
86          std::cout << "X': " << bx << std::endl;
87          std::cout << "Y': " << by << std::endl;
88
89          if(tracker == camaraPoints.size()-1)
90          {
91              tracker = 0;
92          }
93        //  cx = camaraPoints[tracker].getX();
94  //          cz = camaraPoints[tracker].getY()+2000;
95          //std::cout << cx << std::endl;
96          if(counter == 3)
97          {
98              tracker++;
99
00          }
01          if(counter == 3)
02          {
03              counter = 0;
04          }
05          counter++;
06      }
07      Geo_Func::points.clear();
08      Geo_Func::points = temp;
09
10
11     //Camara Rotation
12
13  }
14  void Geo_Func::rotateX()
15  {
16      std::vector<Points> temp;
17      double gx = (degToRad(rx));
18      for(Points value:points)
19      {
20          temp.push_back(Points(
21                              value.getX(),
22                              round(value.getY()*cos(gx) - value.getZ()*sin(gx)),
23                              round(value.getY() * sin(gx) + value.getZ()*cos(gx))));
24      }
25
```

```
125
126     points = temp;
127     tempP = points;
128
129 }
130 void Geo_Func::rotateY()
131 {
132     std::vector<Points> temp;
133     double gx = (degToRad(rx));
134     for(Points value:points)
135     {
136         temp.push_back(Points(
137                         round(value.getX()*cos(gx) + value.getZ() * sin(gx)),
138                         value.getY(),
139                         round(-value.getX() * sin(gx) + value.getZ() * cos(gx))
140                            ));
141     }
142     points = temp;
143     //tempP = points;
144 }
145
146 void Geo_Func::rotateZ()
147 {
148     std::vector<Points> temp;
149     double gx = (degToRad(rx));
150     for(Points value:points)
151     {
152         temp.push_back(Points(
153                         round(value.getX() * cos(gx) - value.getY() * sin(gx)),
154                         round(value.getX() * sin(gx) + value.getY() * cos(gx)),
155                         value.getZ()
156                            ));
157     }
158     points = temp;
159     tempP = points;
160 }
161 //Elipse
```

```
void Geo_Func::createFigure()
{
    //Front vertices
    points.push_back(Points(-50,50,-50));
    points.push_back(Points(50,50,-50));

    points.push_back(Points(50,50,-50));
    points.push_back(Points(50,-50,-50));

    points.push_back(Points(50,-50,-50));
    points.push_back(Points(-50,-50,-50));

    points.push_back(Points(-50,-50,-50));
    points.push_back(Points(-50,50,-50));


    //Back Vertices

    points.push_back(Points(-50,50,50));
    points.push_back(Points(50,50,50));

    points.push_back(Points(50,50,50));
    points.push_back(Points(50,-50,50));

    points.push_back(Points(50,-50,50));
    points.push_back(Points(-50,-50,50));

    points.push_back(Points(-50,-50,50));
    points.push_back(Points(-50,50,50));


   // Conecting Back and front vertices

    points.push_back(Points(-50,50,-50));
    points.push_back(Points(-50,50,50));

    points.push_back(Points(50,50,-50));
    points.push_back(Points(50,50,50));

    points.push_back(Points(50,-50,-50));
    points.push_back(Points(50,-50,50));

    points.push_back(Points(-50,-50,-50));
```

```
205     points.push_back(Points(-50,-50,50));
206
207 //Pyramid
208     points.push_back(Points(50,50,-50));
209     points.push_back(Points(200,0,0));
210
211     points.push_back(Points(50,-50,-50));
212     points.push_back(Points(200,0,0));
213
214     points.push_back(Points(50,50,50));
215     points.push_back(Points(200,0,0));
216
217     points.push_back(Points(50,-50,50));
218     points.push_back(Points(200,0,0));
219
220
221 //Pyramid
222     points.push_back(Points(-50,50,-50));
223     points.push_back(Points(-200,0,0));
224
225     points.push_back(Points(-50,-50,-50));
226     points.push_back(Points(-200,0,0));
227
228     points.push_back(Points(-50,50,50));
229     points.push_back(Points(-200,0,0));
230
231     points.push_back(Points(-50,-50,50));
232     points.push_back(Points(-200,0,0));
233
234
235
236
237
238
239
240
241     tempP = points;
242 }
243
244 double Geo_Func::degToRad(double deg)
245 {
246     //std::cout << M_PI * deg / 180.0 << "\n";
247     return M_PI * deg / 180.0;
248 }
```

```
}
void Geo_Func::loadCamaraPoints()
{
    std::ifstream camaraFile;
    camaraFile.open("Elipse Points.txt");
    std::string strTemp = "";
    std::string temp = "";
    std::vector<std::string>str;
    while(camaraFile >> strTemp)
    {
        str.push_back(strTemp);
    }
    std::cout << str[0] << std::endl;
    for(int i = 0; i < str.size();i++)
    {
        int x;
        int y;
        for(int j = 0; j < str[i].size();j++)
        {
            std::string s = str[i].substr(j,1);
            if(s.compare(",") == 0)
            {
                x = std::stoi(temp);
                std::cout << "X: " << x << std::endl;
               // x -=250;
                temp = "";
            }
            else
                temp +=str[i][j];
        }
        y = std::stoi(temp);
        std::cout << "Y: " << y << std::endl;
       // y = (y -250)+-1;

        camaraPoints.push_back(Points(x,y,0));
         temp = "";
    }
}
```

```cpp
void Geo_Func::move_to_origin()
{
    if(!points.empty())
    {
        std::cout << "TO Origin" << std::endl;
        int maxX = points[0].getX();
        int minX = points[0].getX();
        int maxY = points[0].getY();
        int minY = points[0].getY();
        int maxZ = points[0].getZ();
        int minZ = points[0].getZ();
        for(Points values: points)
        {
            if(maxX < values.getX())
                maxX = values.getX();
            if(maxY < values.getY())
                maxY  = values.getY();
            if(maxZ < values.getZ())
                maxZ = values.getZ();
            if(minX > values.getX())
                minX = values.getX();
            if(minY > values.getY())
                minY = values.getY();
            if(minZ > values.getZ())
                minZ = values.getZ();
        }
        int middleX = round((maxX +  minX) / 2);
        int middleY = round((maxY +  minY) / 2);
        int middleZ = round((maxZ +  minZ) / 2);
        std::vector<Points> temp;
        std::cout << "Mx: " << middleX << std::endl;
        std::cout << "My: " << middleY << std::endl;
        std::cout << "Mz: " << middleZ << std::endl;
        for(Points value:points)
        {
            temp.push_back(Points(value.getX()-middleX,
                                  value.getY()-middleY,
                                  value.getZ() - middleZ));
        }
        points = temp;
    }
}
```

```
void Geo_Func::moveBack()
{
     if(!points.empty())
    {
         std::vector<Points> temp;
    for(Points value:points)
    {
        temp.push_back(Points(
                            value.getX()+middleX,
                            value.getY()+middleY,
                            value.getZ()+middleZ));
    }
    points = temp;
     }
     middleX = 0;
     middleY = 0;
     middleZ = 0;
     tempP = points;
}

```

```
#include "Init_allegro.h"

Init_Allegro::Init_Allegro(void)
{

}

bool Init_Allegro::execute = false;


void Init_Allegro::inicialize(void)
{
    if(!execute)
{
    std::cout << "Inicialized" << std::endl;
    al_install_keyboard();
    al_install_mouse();
    eventqueue = al_create_event_queue();
    timer = al_create_timer(1.0/30);
    al_register_event_source(eventqueue,al_get_keyboard_event_source());
    al_register_event_source(eventqueue,al_get_timer_event_source(timer));
    al_register_event_source(eventqueue,al_get_display_event_source(display));
    al_register_event_source(eventqueue,al_get_mouse_event_source());
    execute = true;
    al_start_timer(timer);
}
else
{
    std::cout << "Did no execute because allegro components should only be initialized once" << std::endl;
    std::cin.get();
}

}
int Init_Allegro::init_display()
{
    display = al_create_display(500,500);
    if(!display)
    {
        fprintf(stderr, "failed to create display!\n");
        return -1;
    }
     else return 1;
}

```

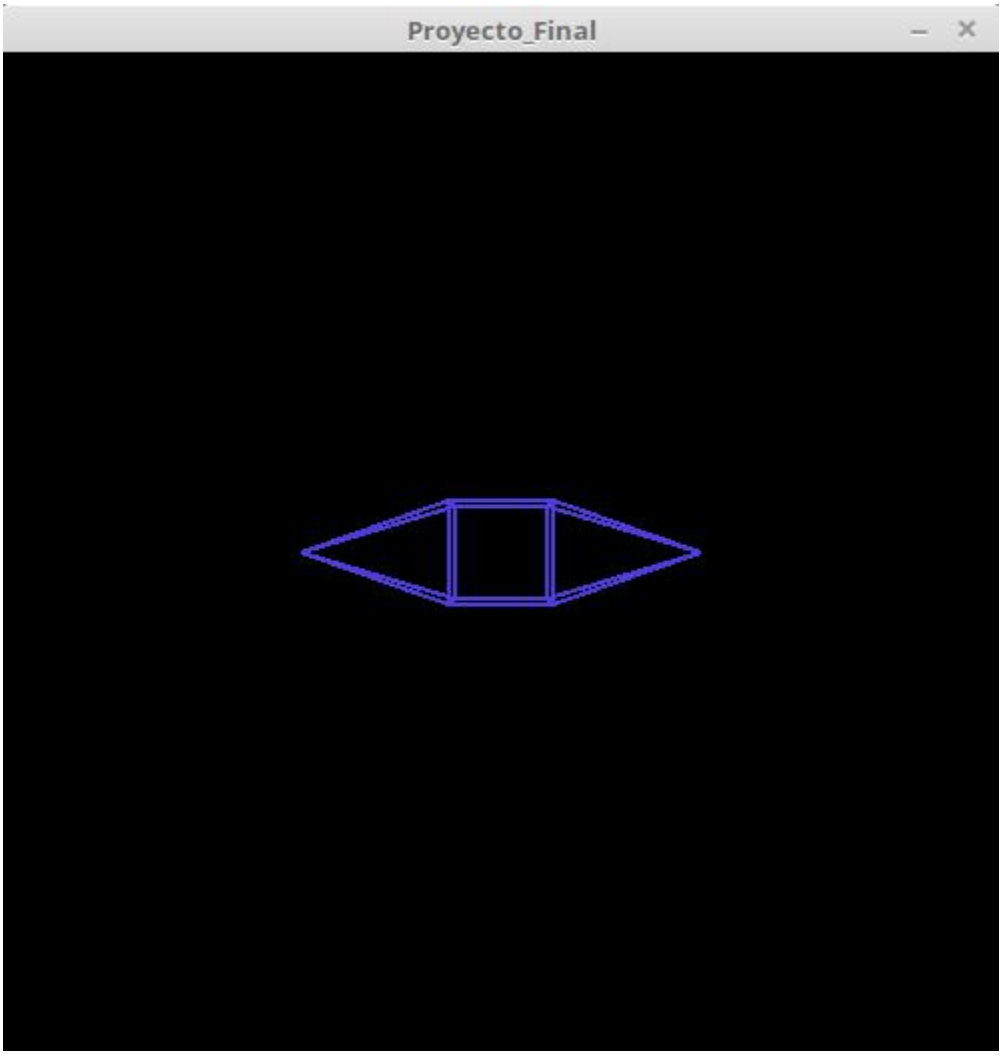
```cpp
#include "Keys.h"

int Keys::x;
int Keys::y;
int Keys::z;

Keys::Keys()
{

}

bool Keys::checkInputs(ALLEGRO_EVENT &event)
{
    if(event.type == ALLEGRO_EVENT_KEY_DOWN)
    {
        switch (event.keyboard.keycode) {
        case ALLEGRO_KEY_P:
           std::cout << "projecting" << std::endl;
           Geo_Func::rotateY();
           Geo_Func::projection();
           return true;
           break;
        case ALLEGRO_KEY_C:
            Geo_Func::createFigure();
            return true;
            break;
        default:
            break;
        }
    }
    else if(event.type == ALLEGRO_EVENT_MOUSE_BUTTON_DOWN)
    {
        if(event.mouse.button & 1)
        {
            Keys::x = event.mouse.x;
            Keys::y = event.mouse.y;
            Keys::z = 0;
            Geo_Func::addPoint(x,y,z);
            return true;
        }
    }
    return false;
}
```

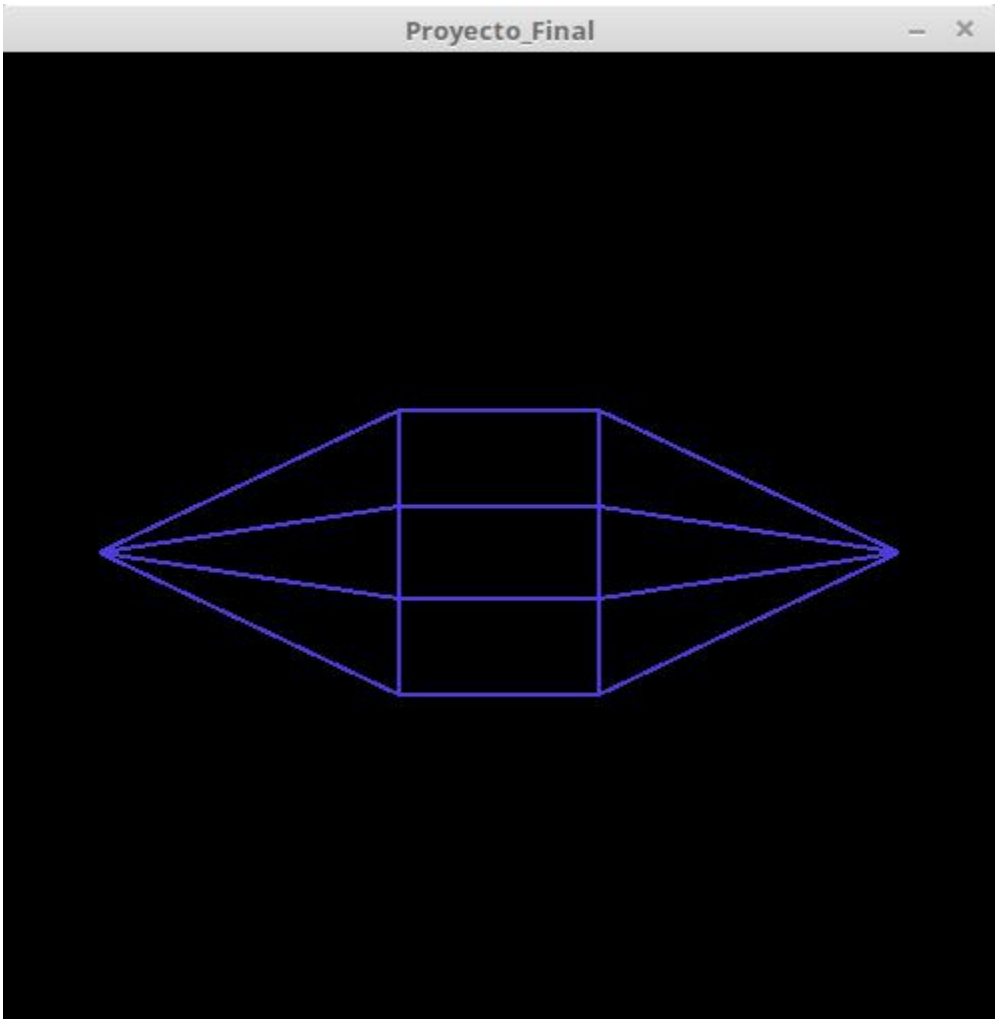
```cpp
#include "Points.h"

Points::Points(int x,int y,int z)
{
    this->x = x;
    this->y = y;
    this->z = z;
}

int Points::getX()
{return x;}

int Points::getY()
{return y;}
int Points::getZ()
{return z;}

void Points::setX(int x)
{
    this->x = x;
}

void Points::setY(int y)
{
    this->y = y;
}
void Points::setZ(int z)
{
    this->z = z;
}

```

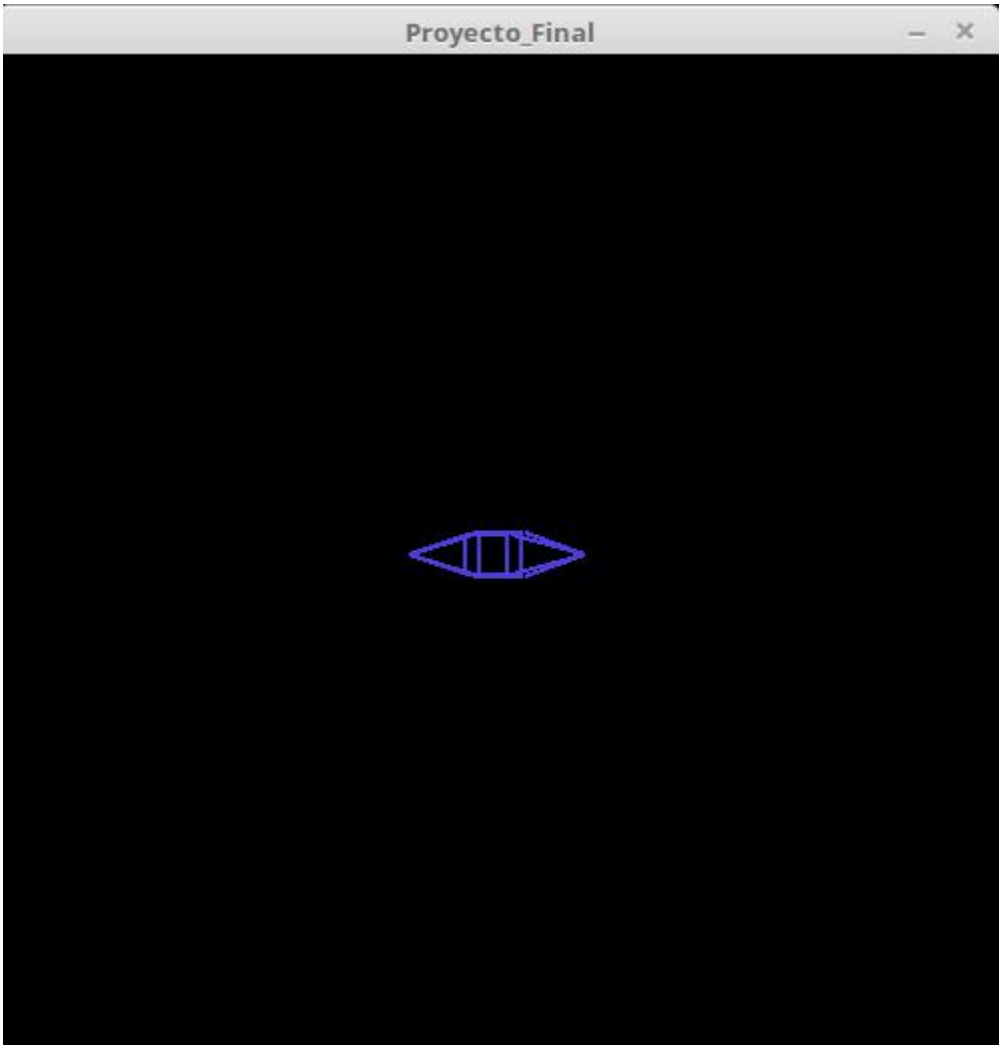
```cpp
#include <iostream>
#include <allegro5/allegro.h>
#include <allegro5/allegro_primitives.h>
#include <Init_allegro.h>
#include "Keys.h"

int main(void)
{
    if(!al_init())
    {
        return -1;
    }

    Init_Allegro inicia;
    Geo_Func::loadCamaraPoints();
    Geo_Func::move_to_origin();
    /*for(int i = 0; i < Geo_Func::getCamaraPoints().size();i++)
    {
        std::cout << Geo_Func::getCamaraPoints()[i].getX() << std::endl;
    }
    */

    if(!inicia.init_display())
    return -1;


    bool gameLoop = false;
    bool redraw = false;

    inicia.inicialize();

   while(!gameLoop)
   {
       ALLEGRO_EVENT event;
       al_wait_for_event(inicia.eventqueue,&event);
       if(event.type == ALLEGRO_EVENT_TIMER)
       {
            redraw = true;
       }
       else if(event.type == ALLEGRO_EVENT_DISPLAY_CLOSE)
       {
            gameLoop = true;
       }
```

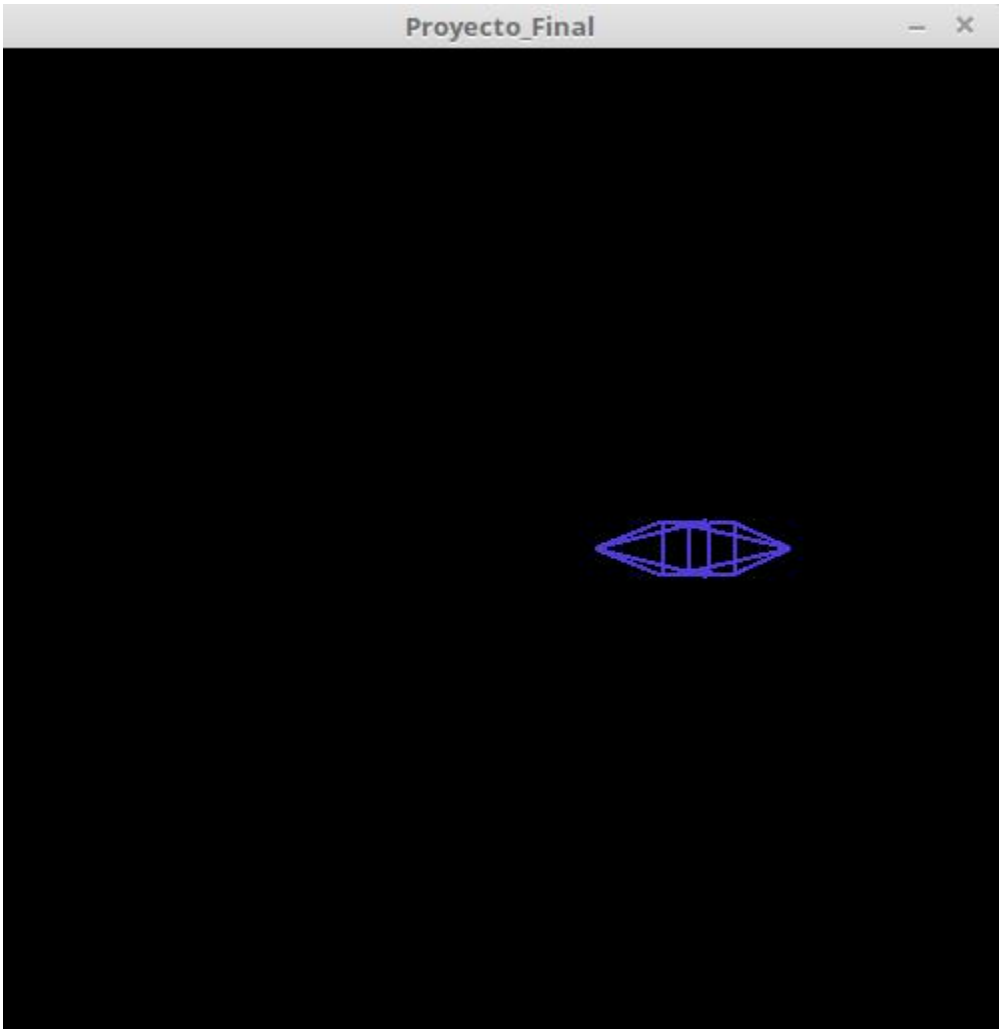
```
        }
        if(Keys::checkInputs(event))
            redraw = true;
        if(redraw == true and al_is_event_queue_empty(inicia.eventqueue))
        {

            redraw = false;
            al_clear_to_color(al_map_rgb(0,0,0));
            //al_draw_line(250,0,250,500,inicia.red,1.0);
            //al_draw_line(0,250,500,250,inicia.red,1.0);
           // Geo_Func::rotateZ();
            //Geo_Func::rotateX();
            Geo_Func::move_to_origin();
            Geo_Func::rotateX();
           // Geo_Func::rotateY();
            Geo_Func::moveBack();


            //Geo_Func::projection();
            for(int i = 1; i < Geo_Func::getPoints().size();i+=2)
            {
                 al_draw_line(
                             Geo_Func::getPoints()[i-1].getX()+250,
                             Geo_Func::getPoints()[i-1].getY()*-1+250,
                             Geo_Func::getPoints()[i].getX()+250,
                             Geo_Func::getPoints()[i].getY()*-1+250,
                             inicia.blue,2.0);
            }
            Geo_Func::resetP();
            al_flip_display();
        }
    }
    return 0;
}

```

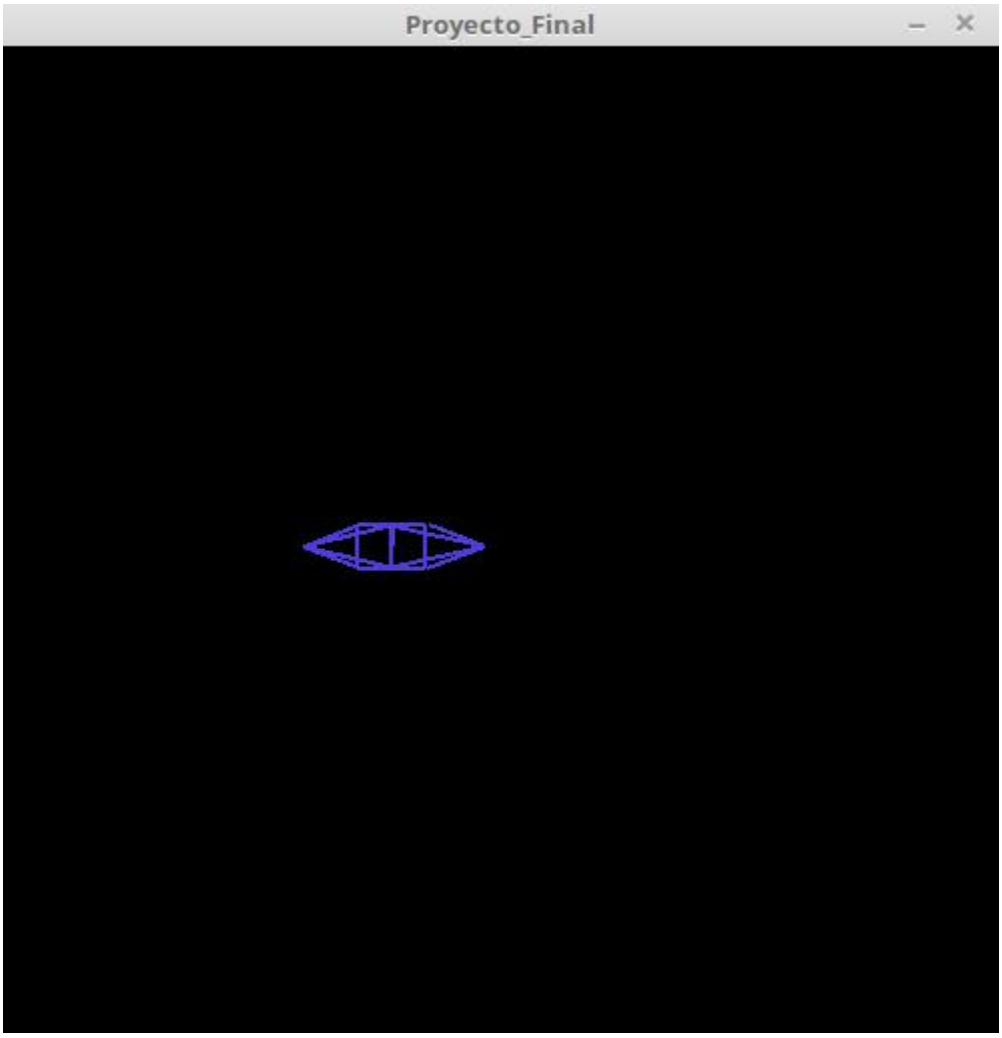
Apéndice 2 (Simulación)




Proyecto_Final


Proyecto_Final


Proyecto_Final


Proyecto_Final