



Universidad Autónoma de San Luis Potosí



Facultad de Ingeniería

Área de Ciencias de la Computación

Graficación por Computadora A

Dr. José Ignacio Núñez Varela

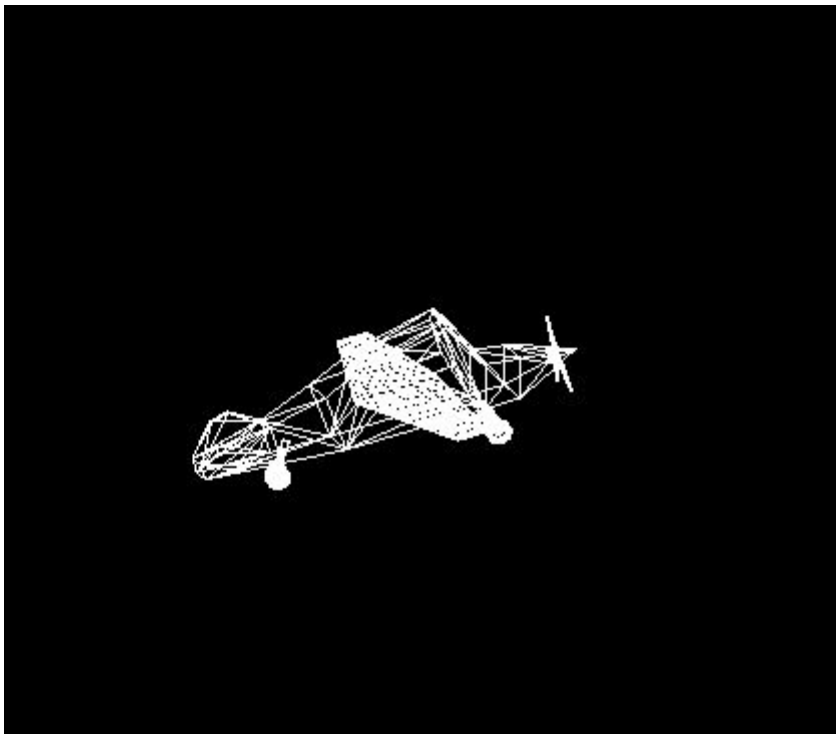
Reporte: Simulación de Avión

Méndez Gutiérrez Nestor Javier



Introducción:

Este proyecto tiene como objetivo simular el despegue, vuelo y aterrizaje de un avión, todo esto a través del área de la graficación por computadora, realmente es un proyecto bastante simple ya que no se toman en cuenta todos los factores físicos que conlleva el despegue de un avión en la vida real, más sin embargo es el inicio de mi experiencia con las gráficas por computadora y es importante ya que parte del objetivo del proyecto es aprender cómo es que a partir de primitivas geométricas se pueden graficar imágenes que pueden llegar a representar una aproximación de objetos de la vida real, por esta razón es que las gráficas por computadoras tienen una relevancia tan grande en nuestra vida actual, desde el entretenimiento como películas y videojuegos hasta el área de ciencias de la salud donde se utiliza en para visualizar imágenes tridimensionales tomadas del cuerpo de algún paciente y las cuales fueron generadas por escáneres, otro ejemplo que vale la pena mencionar es la aplicación de esta área de la computación es en la astrofísica donde literalmente se utilizan para simular el origen del universo y con esto analizar la evolución que este ha tenido durante miles de millones de años, menciono estos ejemplos solo para recalcar la importancia del conocimiento y estudio del CG y para mi este es el principal objetivo de aprender cómo funciona esta área de la computación.



Desarrollo

OpenGL (Open Graphics Library) es un API de desarrollo de proyectos de para la definición de imágenes en 3D y 2D, antes de que existiera OpenGL dentro de la industria de la graficación por computadora las cosas eran bastante más complicadas, ya que para poder desarrollar proyectos de esta índole los desarrolladores tenían que conocer bastante bien el hardware de gráficos con el que estaban trabajando porque el desarrollo de este tipo de proyectos se realizaba desde cero, con la llegada de OpenGL las cosas se simplificaron, ya que este es una interfaz que consta de más de 250 funciones que pueden usarse para dibujar escenas tridimensionales complejas a partir de primitivas geométricas simples, tales como puntos, líneas y triángulos.

OpenGL fue desarrollada originalmente por Silicon Graphics Inc. (**SGI**) en 1992 y es usada ampliamente en CAD, realidad virtual, representación científica, visualización de información y simulaciones de vuelo.

En este proyecto OpenGL nos ayuda para la representación gráfica, de la información contenida en nuestras estructuras de datos, mucha de esta información es obtenida de leer uno o varios archivos de tipo .obj el cual contiene información de definición geométrica.



también en este proyecto se implementan algoritmos de visualización de superficie visible, otro de los algoritmos que se implementan es el cálculo de la iluminación, estos algoritmos se explican con mayor detalle en las siguientes páginas.

Estructura del proyecto

Los pasos para el desarrollo de este proyecto fueron:

Lector de archivos OBJ

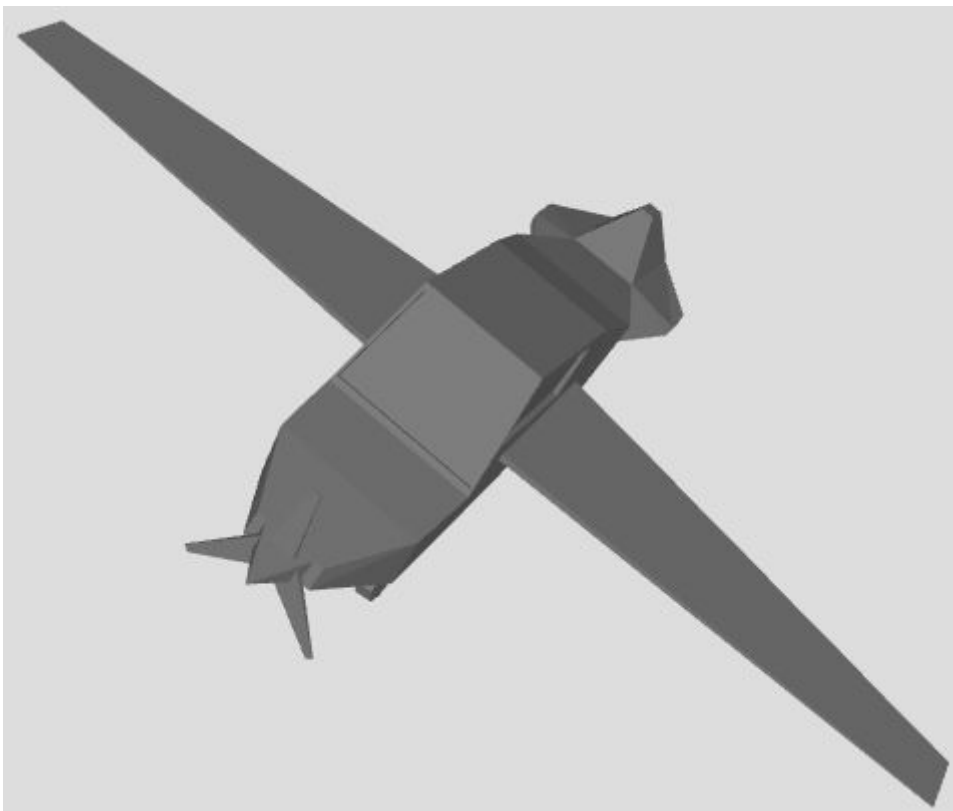
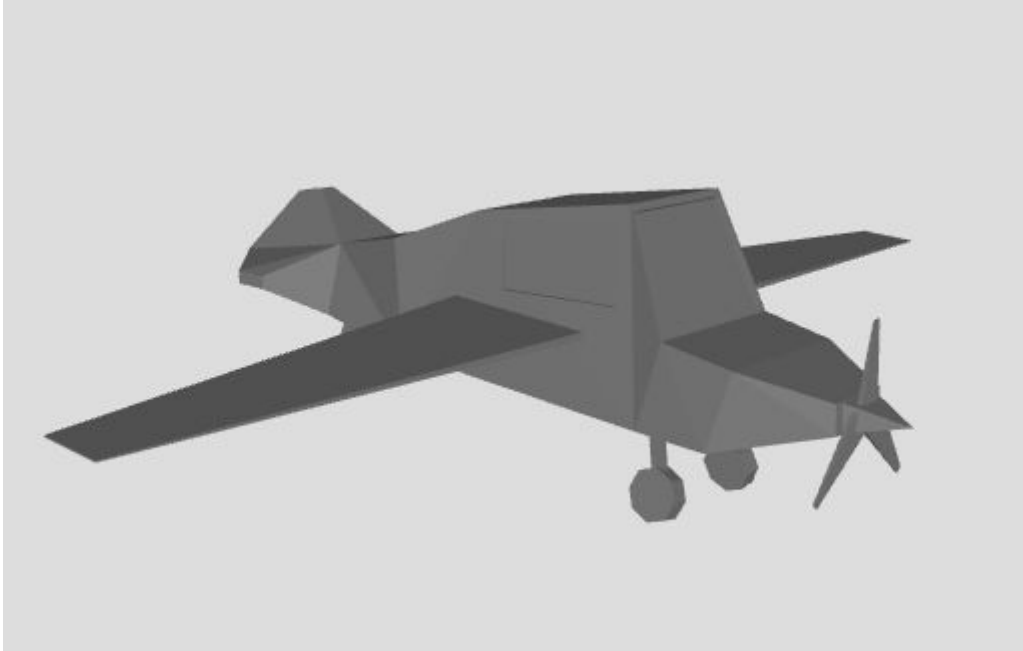
El desarrollo de un lector de archivos OBJ, esto es indispensable para el proyecto ya que los modelos usados en el proyecto se generan a través de programas de dibujo por computadora, en el caso de este proyecto el software usado fue Blender, una vez diseñado el modelo a usar, blender nos ayuda a guardar este modelo en formato OBJ, que como ya se mencionó es un archivo que contiene información geométrica, siendo más específicos, contiene información como coordenadas tridimensionales (Vértices) y una lista de caras en la cual se especifica cuáles son los vértices que componen cada una de las caras del modelo.

Estructura de un archivo .OBJ

# Blender v2.79 (sub 0) OBJ File	El carácter numeral '#' indica que esa
línea	
# www.blender.org	en el archivo es un comentario.
o Airplane	La 'o' indica el nombre del objeto.
v -0.303699 0.090539 0.004706	La letra 'v' indica que lo que contiene esa
v -0.303699 0.090539 -0.002138	línea es un vértice y lo siguiente que existe
v -0.338548 0.026392 -0.002138	en esa misma línea son números que
v -0.338548 0.026392 0.004706	representan las coordenadas (x, y, z) de
v -0.292282 0.084337 0.004706	ese vértice.
v -0.303699 0.090539 0.004706	
f 1 2 3	La letra 'f' indica que lo que existe en esa
f 4 1 3	línea es la información que compone a
una	
f 5 6 7	cara, de manera más explícita lo que
f 8 5 7	realmente contiene es una lista de los
f 9 10 11	vértices que componen a esa cara en
f 12 9 11	específico.

Modelos usados

Para este proyecto solo fue usado un modelo simple de avión el cual se muestra a continuación.

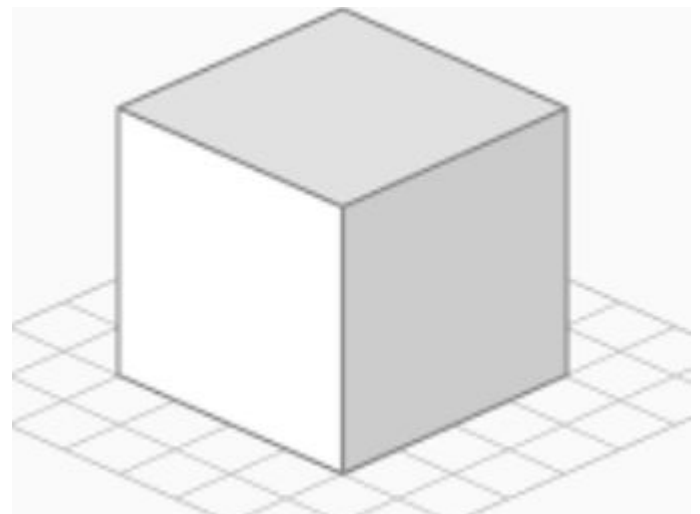
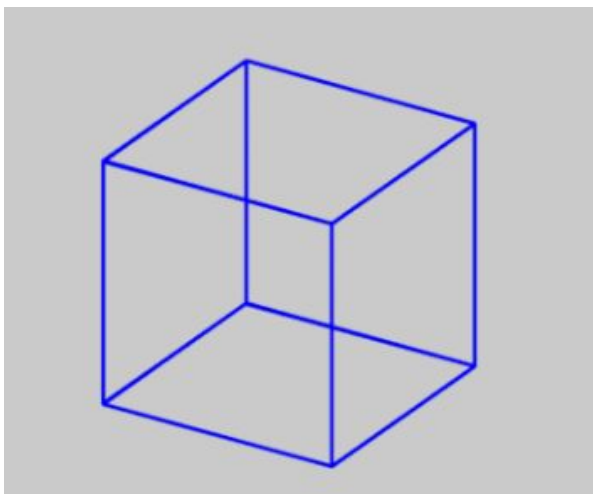


Movimientos

El movimiento que describe el avión es generado a través de el algoritmo de Curvas de Bézier, dicho algoritmo puede generar una curva a partir de cuatro puntos, el punto inicial y final de la curva y dos puntos más los cuales son puntos de control este par de puntos son los que ayudan a controlar la ruta que ha de tomar la curva. Las curvas de Bézier han sido ampliamente usadas en los gráficos generados por computadora para modelado de curvas suaves.

Detección de superficie visible

Este algoritmo se usa para detectar qué superficies del modelo se tienen que dibujar, esto es de bastante utilidad ya que con este algoritmo disminuye la información a procesar y por ende también disminuye el tiempo de ejecución, esto tiene sentido por que suponiendo que el modelo que vamos a graficar tiene miles de caras y de estas miles de caras solo van a ser visibles unas cuantas, ¿para que habríamos de dibujar una superficie que no va a ser visible para la persona que se encuentra detrás del monitor de la computadora?, en la siguiente imagen puede apreciar en un modelo simple cual es la diferencia entre dibujar todas las caras del modelo y solo dibujar las que son estrictamente necesarias.



Iluminación

Este procedimiento se lleva a cabo para lograr más realismo, dentro de lo que corresponde a este algoritmo es el cálculo de diferentes tipos de iluminación, las cuales en su conjunto generan un mayor realismo en el modelo que se está graficando, este procedimiento se realiza para cada cara que contiene el modelo. Este algoritmo se apoya en el conocimiento de los vectores normales de las superficies que se han de iluminar, la posición de la fuente de luz y las variables que a continuación se describen.

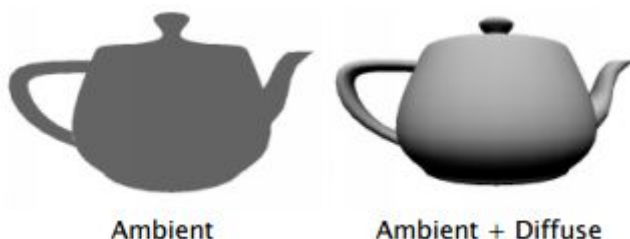
- Ambiente:
Propiedades
 - K_a : Coeficiente de luz ambiente
 - I_a : Intensidad de luz ambiente
- Difusa:
Propiedades
 - K_d : Coeficiente de luz difusa
 - I_l : Intensidad de luz difusa

La ecuación final queda de la siguiente forma:

$$I = K_a * I_a + K_d * I_l * N \cdot L$$

Donde $N \cdot L$ es el producto punto entre el vector normal de la superficie de la cual se está calculando la iluminación y la fuente de luz.

En la siguiente imagen podemos apreciar cómo es que el cálculo de la luz afecta la estética del modelo.





En este par de imágenes se puede apreciar la diferencia entre calcular o no la iluminación para el modelo de avión usado en el proyecto.

Descripción de métodos y clases usadas en el proyecto.

Para este proyecto se han usado las siguientes bibliotecas, entre las que se incluye <GL/glut.h>, la cual nos permite acceder a las funciones de OpenGL, también están incluidas las clases que se usan para el funcionamiento del proyecto.

```
#include <cstring>
#include <cstdio>
#include <vector>
#include <iostream>
#include <math.h>
#include <GL/glut.h>
#include "Archivo.h"
#include "Obj.h"
#include "Cara.h"
#include "Vertice.h"
```

Lo siguiente son las funciones que se encuentran dentro de la función principal (main), dentro de estas se incluyen varias funciones de OpenGL y otras que son propias del proyecto.

- **glutInit(&argc, argv):**
Inicializa la Biblioteca GLUT.
- **glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);**
Especifica la manera de mostrar la ventana.
- **glutInitWindowSize (850, 850);**
Especifica el tamaño de la ventana.
- **glutInitWindowPosition (400, 400);**
Especifica la posición inicial de la ventana.
- **glutCreateWindow ("Avión");**
Especifica el nombre que se le ha de asignar a la ventana.
- **init();**
En esta función inicializamos variables que han de ser necesarios para el funcionamiento del programa.

- **glutDisplayFunc(display);**
Define cual es la función que se va a llamar de manera recurrente para el dibujo en la ventana, es decir la función especificada va a realizar la actualización de la pantalla.
- **glutKeyboardFunc(keyboard);**
Esta función permite detectar cual tecla es presionada y de esta forma mover la perspectiva desde la cual se está visualizando el modelo.
- **glutMouseFunc(mouse);**
Permite detectar cuando se presiona alguno de los botones del mouse, para este proyecto esto se usa para que cuando se da click izquierdo se genere el movimiento de nuestro modelo
- **glutMainLoop();**
Esta función genera un ciclo infinito en dentro del cual nuestro programa siempre se estará ejecutando y cambiará su comportamiento en función de los eventos generados por “glutKeyboardFunc(keyboard)” y glutMouseFunc(mouse).

```
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize (850, 850);
    glutInitWindowPosition (400, 400);
    glutCreateWindow ("Avión");
    init();

    glutDisplayFunc(display);
    glutKeyboardFunc(keyboard);
    glutMouseFunc(mouse);

    glutMainLoop();
    return 0;
}
```

Esta función no se encuentra en el main

void display(void):

Esta función manda a llamar a las funciones que recalculan los vectores normales y la iluminación, para posteriormente redibujar lo que se encuentra en nuestra pantalla y de esta manera generar el efecto de animación.

Descripción del lector de OBJ

Archivo cargaArchivo()

Esta función se encarga de leer el archivo con extensión .OBJ, y lo que hace es que devuelve un objeto de tipo “Archivo” el cual contiene dos listas, una de Obj y otra de la clase Vertice, a su vez cada Obj contiene una lista de identificadores de los vértices que componen cada una de las caras y una Variable de la clase Normal, la cual indica los componentes de la normal de esa cara.

Archivo
vector<Obj> Objs; vector<Vertice> Vertcs;

Obj
vector<Cara> Caras;

Vertice
float x, y, z;

Cara
vector <int> l1IDVertices; Normal normal;

Normal
float A, B, C;

void ImprimeClaseArchivo(Archivo nArch)

Esta función imprime en consola la información que fue obtenida del archivo OBJ, esta función es principalmente para comprobar que la información era la correcta.

void calculaPuntos() y void InicializaBezier()

Este par de funciones son usadas para generar la curva que sigue el modelo InicializaBezier() se encarga de inicializar el valor de los puntos que definen la curva. y calculaPuntos() hace el cálculo de los puntos que sigue el modelo durante la ejecución del programa.

void CalculaNormalVisibilidad();

Esta función realiza el cálculo de la normal e inmediatamente verifica si la superficie ha de ser visible o no.

float iluminacion(int j);

Realiza el cálculo de la iluminación para cada una de las caras que son visibles.

void ActualizaPerspectiva();

Esta se usa cuando se realiza el movimiento de la cámara en el programa ya que el punto de vista está cambiando se tienen que restablecer los valores de las funciones (glMatrixMode, glLoadIdentity, glFrustum, gluLookAt, glutPostRedisplay).

Problemas y Experimentos

En un principio un problema que con el que estuve lidiando bastante durante el desarrollo de este proyecto es que realmente tengo muy poca experiencia trabajando con C++ entonces en cada ocasión en la que me encontré con un problema tenía que investigar casi siempre en foros en internet cómo es que podría resolverlo, por ejemplo, para el desarrollo del lector de OBJ primero tuve que investigar que tipo de estructuras dinámicas debía utilizar, en un principio pensé en utilizar la clase List mas sin embargo cuando llegue a la parte en la que debía acceder a la información dentro de la lista no pude hacerlo como yo creía que podría, entonces después de leer la documentación de la clase List y darme cuenta de que no era posible lo que yo tenía pensado investigue nuevamente y me encontré con la clase vector la cual si me permite realizar lo que en un principio había pensado, realmente la mayor parte de lo problemas que tuve fueron problemas generados por el desconocimiento del lenguaje aunque también me enfrenté al problema de comprender cómo es que funciona OpenGL y las funciones de las que dispone ya que si no se tiene idea de cómo funcionan estas es bastante complicado comenzar a programar.

Comentarios Finales

Este proyecto es sólo un ejemplo de la infinidad de cosas que se pueden desarrollar dentro del área de las gráficas por ordenador, obviamente es bastante simple pero este integra los conceptos básicos de este campo, los conceptos a los que me refiero son el movimiento de los objetos, la simulación de algo tan importante como lo es una fuente de luz y el cálculo de las superficies visibles, estoy bastante consciente que existen una infinidad de mejoras que pueden realizarse dentro de este proyecto, cuestiones como la interacción entre objetos o la simulación de una realidad física, todos estos detalles pueden ser simulados mediante las matemáticas, las cuales son prácticamente el lenguaje de la naturaleza y harían de este proyecto una simulación más fiel de lo que es nuestra realidad.

Referencias

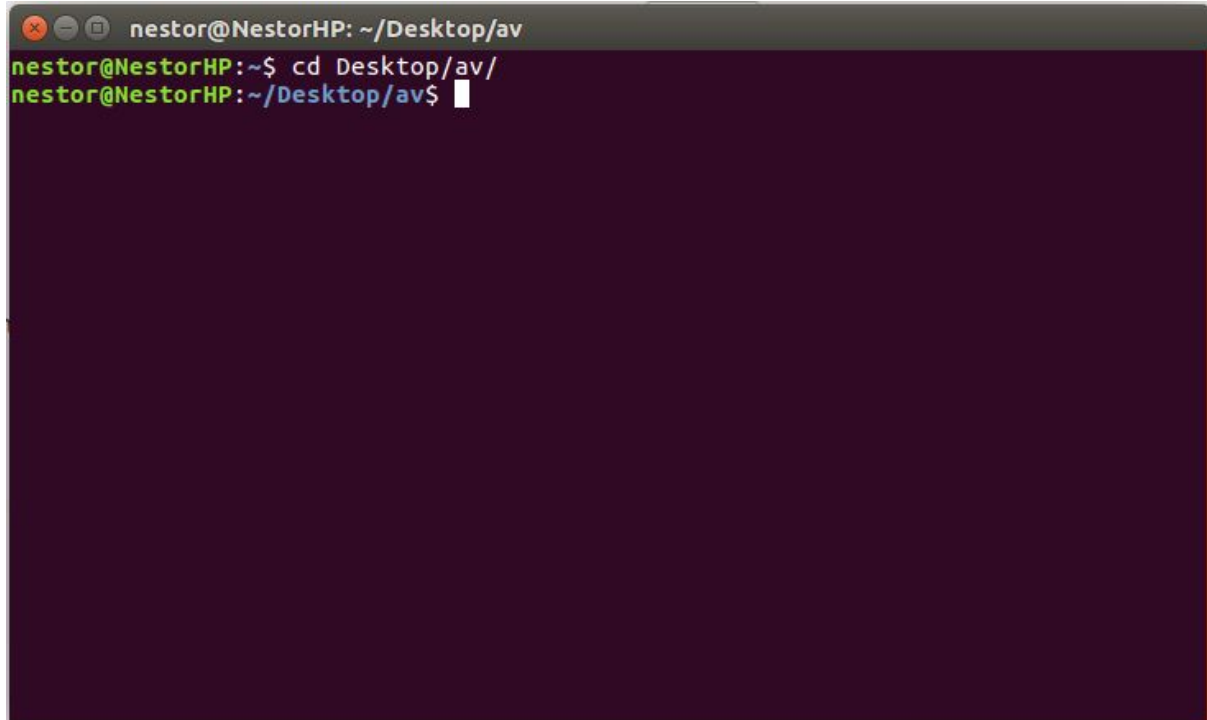
<https://www.khronos.org/registry/OpenGL-Refpages/>

OpenGL Examples, <http://cs.lmu.edu/~ray/notes/openglexamples>

Hearn D., y Baker P. Gráficas por Computadora (con OpenGL) 3a. Edición, Prentice Hall, 2006

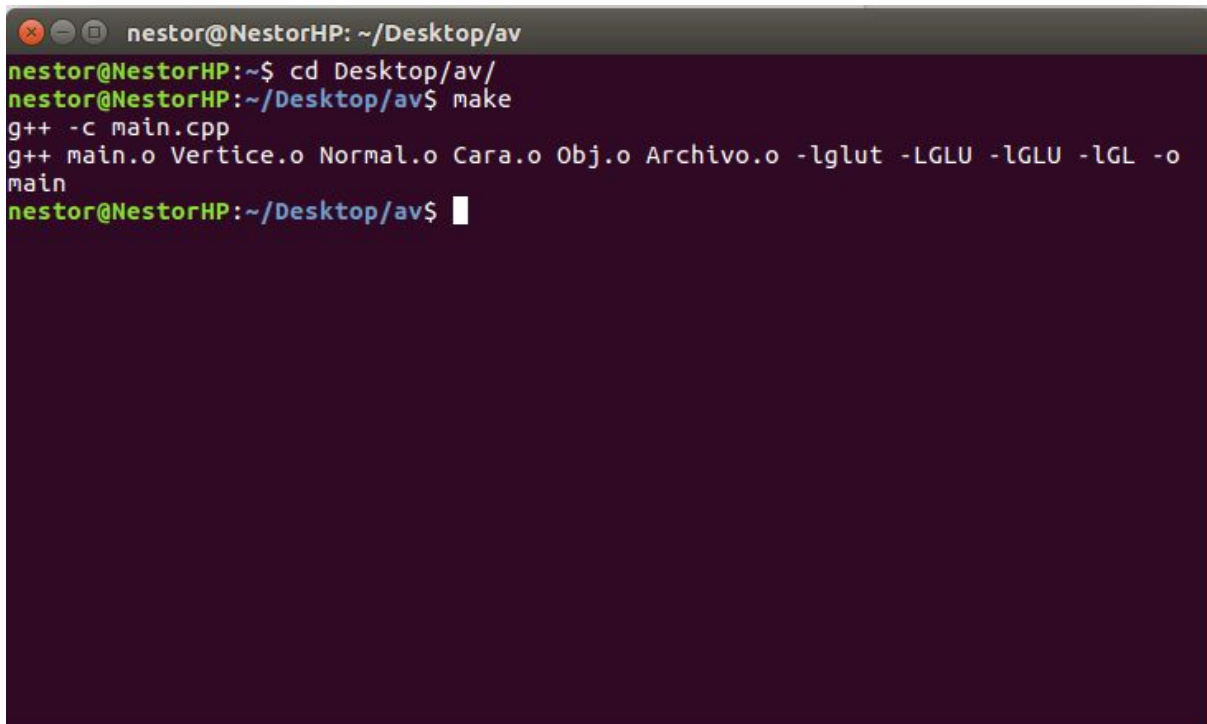
Manual de usuario

1. El primer paso para ejecutar el programa es abrir la terminal de Ubuntu y ubicarse en la carpeta en la que se encuentran los archivos del proyecto.

A terminal window with a dark purple background. The title bar shows 'nestor@NestorHP: ~/Desktop/av'. The prompt is 'nestor@NestorHP:~\$'. The user enters 'cd Desktop/av/' and the prompt changes to 'nestor@NestorHP:~/Desktop/av\$'.

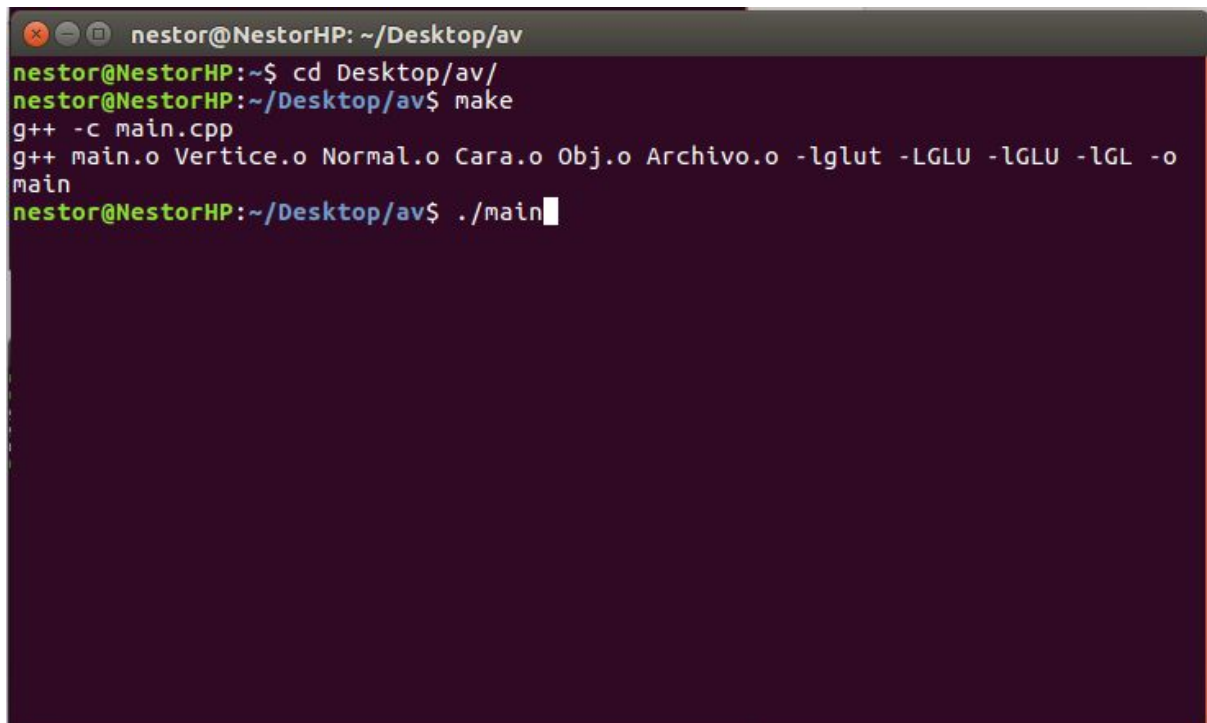
```
nestor@NestorHP: ~/Desktop/av
nestor@NestorHP:~$ cd Desktop/av/
nestor@NestorHP:~/Desktop/av$
```

2. introducir el comando “make”, esto para compilar el proyecto.

A terminal window with a dark purple background. The title bar shows 'nestor@NestorHP: ~/Desktop/av'. The prompt is 'nestor@NestorHP:~/Desktop/av\$'. The user enters 'make'. The terminal shows the compilation process: 'g++ -c main.cpp', 'g++ main.o Vertice.o Normal.o Cara.o Obj.o Archivo.o -lglut -LGLU -lGLU -lGL -o main', and the prompt returns to 'nestor@NestorHP:~/Desktop/av\$'.

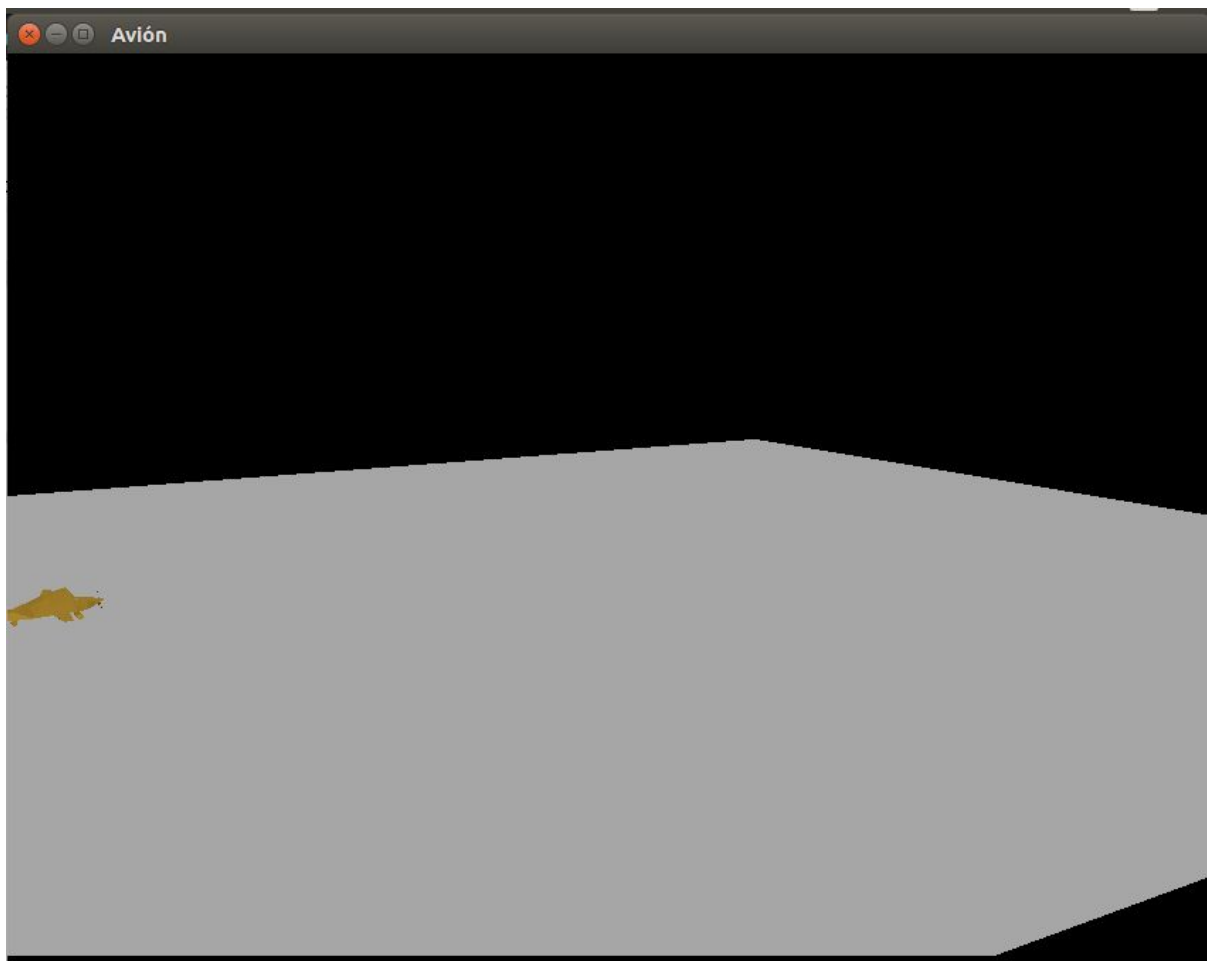
```
nestor@NestorHP: ~/Desktop/av
nestor@NestorHP:~/Desktop/av$ make
g++ -c main.cpp
g++ main.o Vertice.o Normal.o Cara.o Obj.o Archivo.o -lglut -LGLU -lGLU -lGL -o
main
nestor@NestorHP:~/Desktop/av$
```

3. Ejecutar desde consola el programa, que en nuestro caso el nombre del ejecutable es “main”, por tanto hay que escribir “./main”.

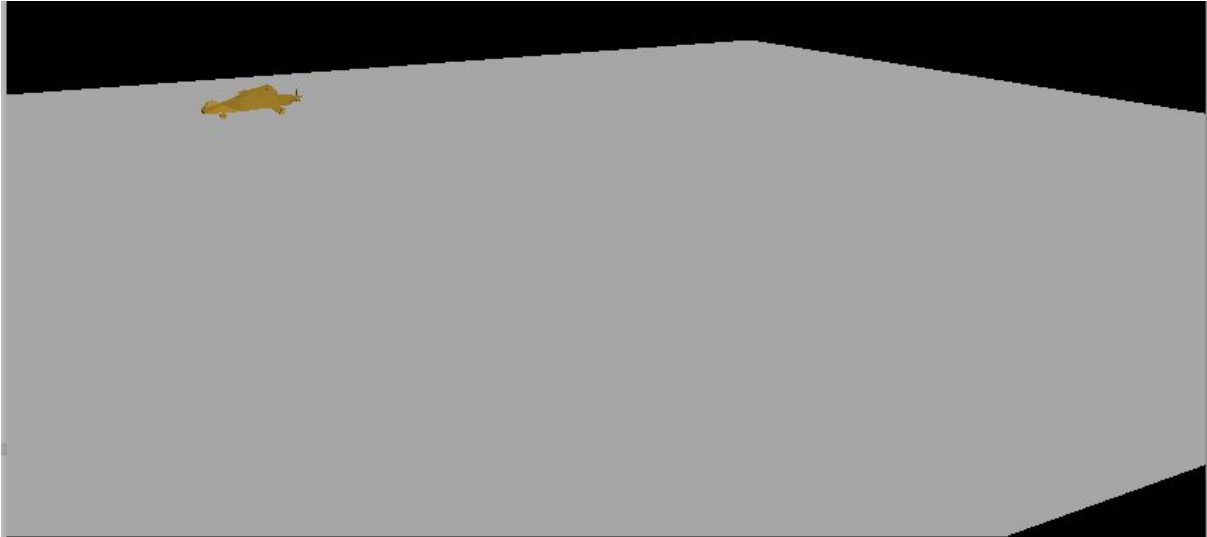


```
nestor@NestorHP: ~/Desktop/av
nestor@NestorHP:~$ cd Desktop/av/
nestor@NestorHP:~/Desktop/av$ make
g++ -c main.cpp
g++ main.o Vertice.o Normal.o Cara.o Obj.o Archivo.o -lglut -LGLU -lGLU -lGL -o
main
nestor@NestorHP:~/Desktop/av$ ./main
```

4. Una vez ejecutado el programa aparece la siguiente pantalla.



5. Para que la animación comience solo queda dar click izquierdo sobre la pantalla.



6. Ahora para mover la perspectiva desde la cual se está observando se hace lo siguiente.

- para navegar en el eje de las X se usan las teclas “a” para moverse hacia -X y “d” para moverse hacia +X.
- para navegar en el eje de las Y se usan las teclas “s” para moverse hacia -Y y “w” para moverse hacia +Y.
- para navegar en el eje de las Z se usan las teclas “+” para moverse hacia -Z y “-” para moverse hacia +Z.

en la siguiente imagen se muestra la animación vista desde otro punto de vista.

