Universidad Autónoma de San Luis Potosí

Reporte: Lanzamiento de un paquete con paracaídas.

Clave: 285476

Nombre del alumno: Mendoza Rodríguez Fernando

Materia: Graficación por computadora

Nombre del profesor: Nuñez Varela José Ignacio

Fecha: Martes 3 de Diciembre del 2019

Introducción

“Lanzamiento de un paquete con paracaídas” es un proyecto que simula las trayectorias de una caja la cual tiene incorporado un paracaídas y un avión que lanza la caja, el proyecto consiste en imitar el movimiento mencionado por medio de modelos hechos en computadora y algunos algoritmos, los cuales sirven para calcular curvas y lineas, dicho de otra manera se utilizan los algoritmos para calcular el movimiento de los modelos y llevar a cabo el funcionamiento.

El objetivo de este proyecto es servir como material de apoyo para el aprendizaje de física, con él se podría ejemplificar a los aprendices como es que se vería la caída de un cuerpo con paracaídas en algunas situaciones hipotéticas, también puede servir como material de aprendizaje en programación y simulación ya que es un ejemplo de cómo utilizar diferentes elementos de la biblioteca OpenGL y cómo resolver ciertas situaciones relacionadas a la graficación como la determinación de superficies visibles de un modelo (decidir qué caras del modelo se dibujan en pantalla), puede servir para que que alguien mas modifique el funcionamiento logrando diferentes resultados y adaptándolo a su gusto, cómo hacer que el avión realice piruetas en el espacio,por último puede servir como mera curiosidad para la gente que quiera ver algo que se puede hacer desarrollando software.

Desarrollo

OpenGL

OpenGL es una API (Application Programming Interface) la cual consiste en una serie de funciones que usan las instrucciones de cada tarjeta grafica para obtener los mismos resultados en cuanto a creacion de graficos 2D y 3D en computadora.

Un objetivo de OpenGL es brindar la posibilidad de hacer gráficos con calidad ocultando las primitivas de cada tarjeta gráfica y así facilitando el proceso.

OpenGL puede ser instalado en algunas distribuciones de linux ejecutando los siguientes comandos:

```
$ sudo apt-get update
$ sudo apt-get install libglu1-mesa-dev freeglut3-dev mesa-common-dev
```

Para utilizarlo hay que crear un programa en este caso se utilizará un programa hecho en C++.

Es importante mencionar que OpenGL proporciona un entorno de programa orientado a eventos, el cual es básicamente un ciclo infinito a la espera de que algo suceda para ejecutar alguna acción.

Ejemplo de programa OpenGL en C++:

```
#include <GL/glut.h>  //Debemos importar esta biblioteca
//Esta es la funcion que dibuja los gráficos
void displayMe(void){
   glClear(GL_COLOR_BUFFER_BIT);
   glBegin(GL_POLYGON);
      glVertex3f(0.5, 0.0, 0.5);
      glVertex3f(0.5, 0.0, 0.0);
      glVertex3f(0.0, 0.5, 0.0);
      glVertex3f(0.0, 0.0, 0.5);
   glEnd();
   glFlush();
}
//El main solo sirve para inicializar lo necesario, una vez terminado se vuelve orientado a eventos
int main(int argc, char** argv){
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE);
   glutInitWindowSize(400, 300);
   glutInitWindowPosition(100, 100);
   glutCreateWindow("Hello world!");
   glutDisplayFunc(displayMe);
   glutMainLoop();
   return 0;
}
```

Para compilar un programa en OpenGL hay que ejecutar el siguiente comando:

```
$ g++ nombreDelArchivo.cpp -o nombreDeseadoDelEjecutable -lglut -lGLU -lGL
```

Cabe mencionar que no si se ocupan más de un archivo .cpp se ponen uno seguido de otro en el comando:

```
$ g++ archivo1.cpp archivo2.cpp -o nombreDeseadoDelEjecutable -lglut -lGLU -lGL
```

Para ejecutarlo solo hay que ejecutar el comando:

```
$./nombreDelEjecutable
```

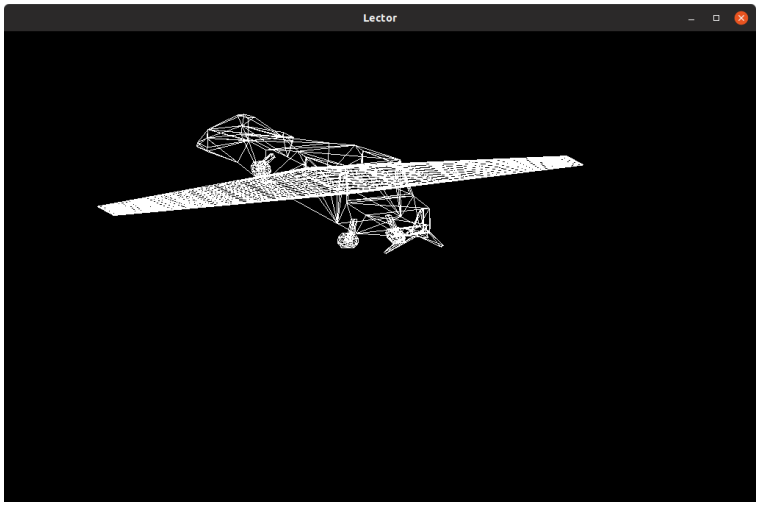
Proyecto

Lector Obj/ Archivos Obj

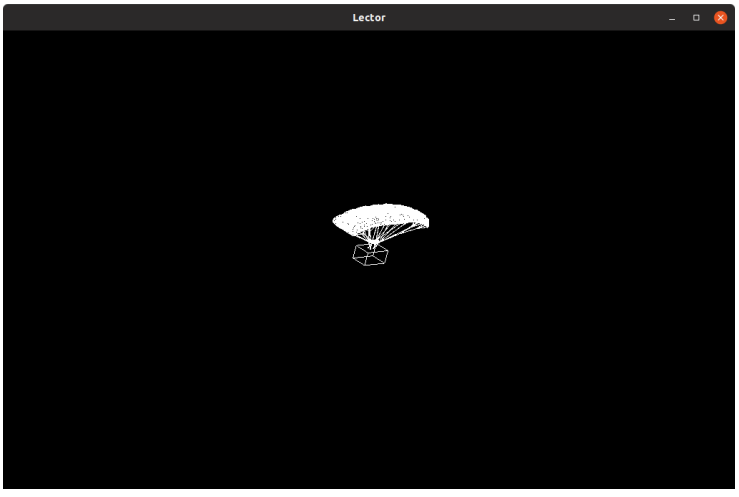
El Lector de archivos obj es un programa que nos permitirá leer archivos con extensión .obj, es importante mencionar que dicha extensión corresponde a un formato de archivos que es usado para guardar la información utilizada para generar de manera gráfica modelos a computadora, la función principal de este programa consiste en la posibilidad de desplegar en pantalla dichos modelos en el modo de visualización llamado wireframe, el cual consiste en dibujar sólo las aristas que constituyen a una figura (en otras palabras no se realiza el relleno de color en las caras) el programa soporta la lectura de archivos que contienen más de un modelo. El lector está hecho en el lenguaje de programación C++ haciendo uso del paradigma orientado a objetos.

Modelos

Para el proyecto se utilizarán dos modelos, uno de un avión (el cual bajé de internet)
y otro de una caja con un paracaídas.



Para el segundo modelo bajé el paracaídas de internet y lo fusioné con un cubo utilizando el programa blender.



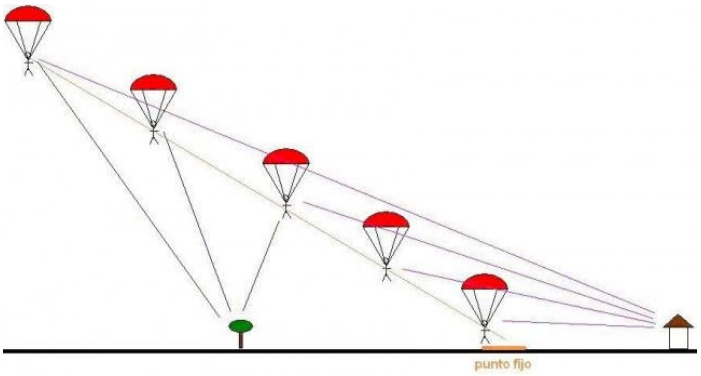
Movimientos

Para el avión se calculan los puntos de una línea recta con el algoritmo DDA ya que se supone que realiza un movimiento rectilíneo uniforme, dicho de otro modo avanza a velocidad constante.

Para controlar la velocidad de el avión se utiliza la frecuencia de refrescado de la pantalla, si se desea que el avión vuele más rápido hay que aumentar el número de cuadros por segundo de la animación, lo que es igual a actualizar la pantalla con una frecuencia equivalente a 1000 ms/Cuadros por segundo.

De esta manera el movimiento del avión será siempre el mismo pero cuando haya una frecuencia de refrescado menor el avión “avanzará” más rápido (en realidad el programa completo).

En cuanto al paquete la trayectoria es curva pero muy poco pronunciada, ya que el paracaídas alcanza una velocidad terminal, además de que realiza pequeñas rotaciones mientras se estabiliza, se utilizó el algoritmo de Bezier para calcular curvas.

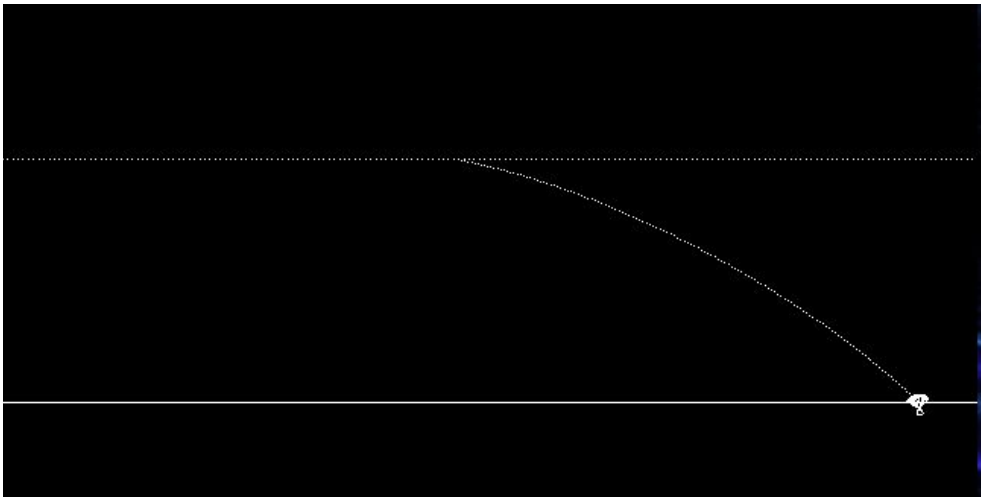


El algoritmo de Bezier consiste en cuatro puntos de control, dos que sirven como los extremos de la línea y dos que sirven para dar cierta curvatura a la línea, el algoritmo depende de un incremento y una variable de control que avanza dependiendo del incremento, dicha variable recorre valores de 0 a 1, indicando el porcentaje de la curva en el que se encuentra y por tanto el porcentaje de componente que debe tomar de los puntos de control para calcular cada punto de la línea.

En las primeras etapas del desarrollo de este programa se utilizó la página Desmos (https://www.desmos.com/calculator/cahqdxeshd), en la cual se puede utilizar una calculadora de curvas Bezier, para definir una curva la cual se veía bien con el lanzamiento del paracaídas sin embargo con la implementación de otras características esto fue cambiando.

Al momento de realizar pruebas con una frecuencia de refrescado de 15 se busco una trayectoria para el algoritmo de Bezier que concordara con el movimiento del avión, haciendo experimentos se encontraron los siguientes puntos de control.

P1(200, 50, 0)
P2(230, 44, 0)
P3(260, 24, 0)
P2(290, 0, 0)



Debido a que la trayectoria del paracaídas depende de la velocidad de el avión tenemos que hacer que los puntos de control de la curva Bezier varíen en el eje x dependiendo de la frecuencia de refrescado, el punto de lanzamiento siempre será el mismo en x.

P1(200, 50, 0)

Pero los demás tienen las siguientes fórmulas:

P2(200 + (900 / (frecuencia_Refrescado * 2)), 44, 0)
P3(200 + (1800 / (frecuencia_Refrescado * 2)), 24, 0)
P2(200 + (2700 / (frecuencia_Refrescado * 2)), 0, 0)

Con esto logramos que la frecuencia de refrescado afecte de forma proporcional el componente x en los puntos, lo cual extiende la curva.

La velocidad de avance del paracaídas también se ve afectada por la velocidad del avión, ya que la velocidad cambia la forma y por tanto el número de puntos necesarios para la curva, entre más extensa sea la curva más puntos necesitaremos, esto quiere decir que tenemos que cambiar el incremento de el algoritmo Bezier con la siguiente fórmula:

incrementoBezier = 0.007*(frecuenciaRefrescado/20)

La cual indica que el incremento es proporcional a la frecuencia de refrescado y por tanto inversamente proporcional a la velocidad, en otras palabras a mayor velocidad menor incremento de Bezier y por tanto más puntos como deseamos, cabe aclarar que la división por 20 es porque en ese momento estaba probando el programa con una frecuencia de refrescado de 20 y lo tomé como referencia, 0.007 se obtuvo de manera experimental.

De igual manera la trayectoria depende de la altura del avión, por lo cual también se modifican los puntos de control en base a esta, a diferencia de la modificación anterior esta se trata de regular la extensión tanto en el eje x como en el eje y, podríamos decir que el ajuste anterior es para modificar la forma de la curva y este es como multiplicar el tamaño de la curva por un factor, esto es porque aunque cambiemos la altura, dada una velocidad del avión el comportamiento será el mismo según la velocidad terminal del paracaídas.

Aprovechando la altura de inicio habitual de lanzamiento se crean el factor alturaAvion/50.

P1(200, 50*(alturaAvion/50), 0)
P2(200 + (900 / (frecuencia_Refrescado * 2))*(alturaAvion/50), 44*(alturaAvion/50), 0)
P3(200 + (1800 / (frecuencia_Refrescado * 2))*(alturaAvion/50), 24*(alturaAvion/50), 0)
P2(200 + (2700 / (frecuencia_Refrescado * 2))*(alturaAvion/50), 0, 0)

Este factor también se aplica al incremento de Bezier solo que de manera inversa, la cantidad de puntos a mostrar es proporcional al tamaño de la curva así que si nuestra curva es más grande desearemos disminuir el incremento de Bezier para aumentar el número de puntos.

incrementoBezier = 0.007*(frecuenciaRefrescado/20)*(50/alturaAvion)

Para los movimientos se utilizan distintos timers, la finalidad de los timers es que en cada uno de ellos se colocan ciertos movimientos necesarios, es como dividir el programa en secciones, la primera sección es avión volando, la segunda sección es avión volando y paracaídas cayendo.

Así que en el primer timer se llamará siempre a sí mismo con la frecuencia de refrescado hasta que termine el algoritmo DDA, cuando terminó el algoritmo llama al segundo timer con los valores necesarios en las variables globales.

De la misma manera el segundo timer se llamará a sí mismo hasta que los dos movimientos estén finalizados, tanto la curva del paracaídas como el avance lineal del avión.

Iluminación

En cuanto a la iluminación hay que mencionar que hay varios componentes en ella, podemos mencionar la luz de ambiente, la luz difusa y la especular.

La luz de ambiente solo es la iluminación que existe en cierto entorno, por ejemplo en un cuarto iluminado de rojo la luz de ambiente será roja, se compone de una variable Ia que indica la intensidad y una constante Ka.

Los otros dos componentes dependen de las características del material sobre el que incide cierta fuente de luz, la luz difusa se refiere a según el color del objeto y la intensidad de una fuente de luz que tanto se nota el color está compuesta por una variable Il que representa la

intensidad de la fuente de luz y una constante Kd que indica que tanto absorbe la luz el objeto dependiendo del material se ve afectada por la ubicación de la fuente de luz con respecto a la cara, para saber qué tanto influye en la iluminación del objeto hay que obtener el ángulo de incidencia, lo cual es posible con el vector normal de la cara y el vector hacia la fuente de luz.

En cambio la especular es que tanta luz refleja nuestro objeto dependiendo desde qué lugar lo estemos viendo y de su coeficiente de reflección, está conformada por la misma Il que es la intensidad de la fuente de luz y una constante Ks que indica que tanto refleja el material, se ve afectada por el ángulo de visión con respecto al punto de visión y el vector de la luz que se refleja.

Ya que el color del objeto puede ser totalmente visible gracias a una combinación de la luz de ambiente y la luz difusa la suma Ka+Kd no puede ser mayor a 1, ya que si la luz de ambiente y la intensidad de la luz de la fuente son 1 (indicando 100%) entonces el grado de coloración máximo también es 1, la suma de luz de ambiente y luz difusa se puede entender como una escala de negro hasta el color del material, mientras que la luz especular se puede ver como una escala del color de el objeto hasta el color de la fuente de luz.

La ecuación para el cálculo de iluminación de cada cara está dada por:

I=Ia*Ka+Il*Kd(vectorNormal.vectorFuenteLuz)+Il*Ks(vectorRefleccion.vectorPuntoVision)^n

y se puede factorizar

I=Ia*Ka+Il*(Kd(vectorNormal.vectorFuenteLuz)+ Ks(vectorRefleccion.vectorPuntoVision)^n)

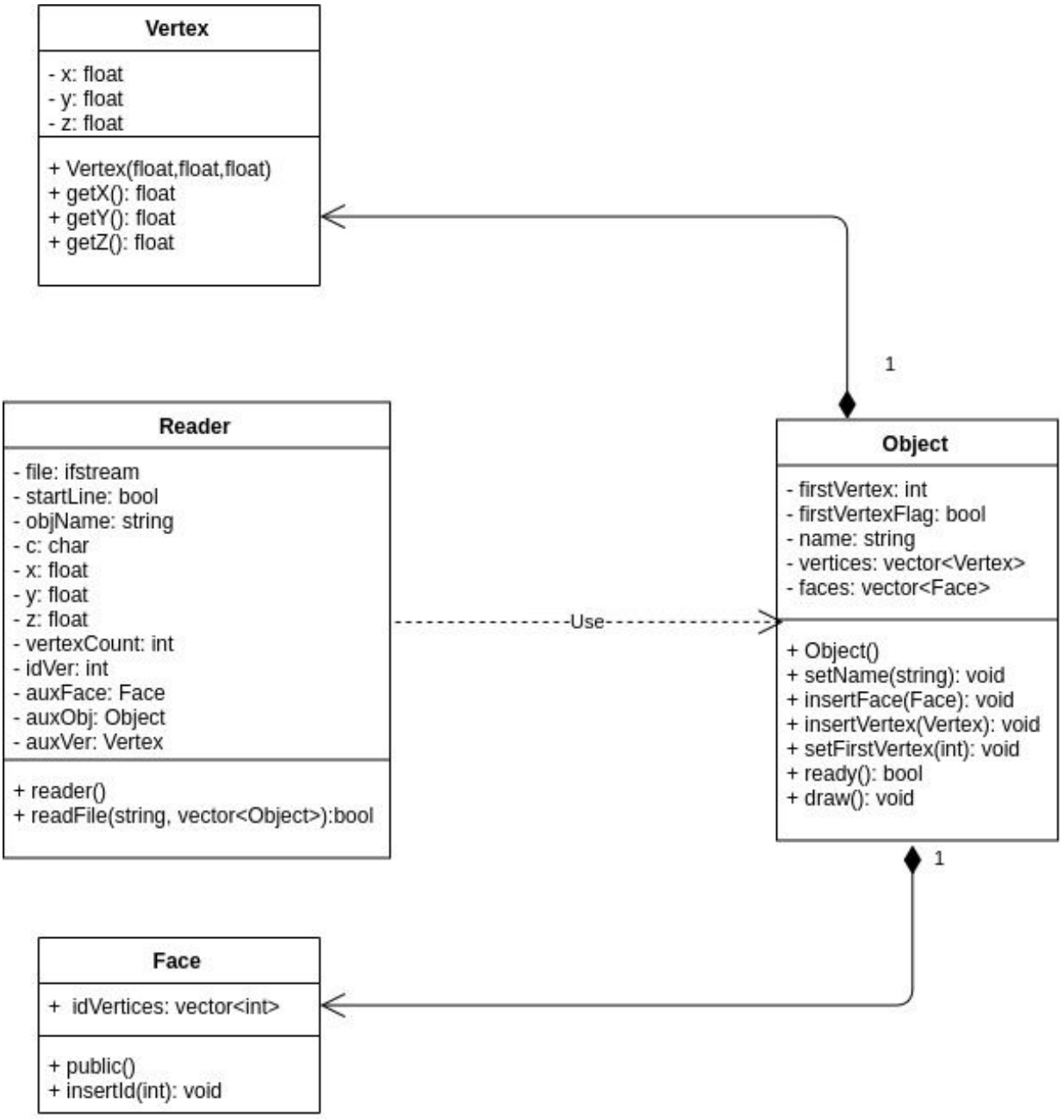
Es importante mencionar que el vector a la fuente de luz se calcula con el punto en el cual se encuentra la fuente de luz en el espacio y uno de los vértices de la cara, al vector dirigido a la fuente de luz se le resta el del vértice.

El vector al punto de visión se obtiene de manera similar, al vector hacia donde está nuestro PRP o cámara se le resta el de un vértice de la cara.

Un detalle importante es que todos los vectores son transformados a unitarios antes de hacer las operaciones ya que de esa manera arrojaran resultados máximos de 1 al realizar el producto punto.

Estructura del programa

Cuando se hace un modelo por computadora (una representación gráfica de un objeto) la información necesaria para su construcción se guarda en un archivo de extensión .obj, esta información consiste en varias líneas de texto las cuales contienen diferentes elementos, señalados siempre con un caracter al inicio de la línea que indica lo que esta contiene, un '#' indica un comentario, una 'v' indica un vértice, un 'o' indica una figura y una 'f' indica una cara. Para la realización de este programa como ya se había mencionado se siguió el paradigma orientado a objetos con el siguiente diagrama.



Se realizó el diseño de forma que la clase Object contiene una colección de la clase Vertex y otra de la clase Face, Vertex contiene tres flotantes que indican sus coordenadas en el

espacio además de los métodos de acceso para cada flotante, y `Face` contiene una colección de enteros los cuales indican los índices de los vértices que la conforman, además un método para insertar un nuevo índice.

Object también contiene una cadena donde se guarda el nombre de la figura contenida, un número del primer vértice de la figura (ya que la numeración de los vértices es compartida por las figuras del archivo una figura puede comenzar en el vértice 27 por ejemplo), el cual servirá para poder acceder al vértice correcto dentro de la instancia, después de restar este número (`firstVertex`) a los números de índice contenidos en la colección dentro de `Face` y una bandera que indica si este número ya fue asignado, Contiene un método para asignar el nombre (`setName`), uno para insertar una cara a la colección (`insertFace`), uno para insertar un vértice a la colección (`insertVertex`), uno para asignar el número del primer vértice introducido (`setFirsVertex`), otro para asegurarse que tiene los datos suficientes para ser insertado en una colección de objetos (`ready`) y finalmente uno para desplegarlo en pantalla (`draw`).

La clase `Reader` solo contiene una variable para poder abrir el archivo y distintos auxiliares para poder ensamblar algún objeto que será posteriormente insertado en la colección de objetos del programa, contiene un método para leer el archivo y almacenar los datos en una colección (`readFile`).

Para todas las colecciones mencionadas en las descripciones anteriores se utilizaron vectores, son como arreglos pero dinámicos, la desventaja de estos es que hay que instanciar un objeto de la clase que contiene el vector para luego insertarlo, ya que no se tenía ese espacio previamente reservado en el vector, empieza como un arreglo de cero elementos al cual se le pueden insertar más.

El lector funciona utilizando la clase reader para almacenar los modelos en un vector, de esta manera en la función init se les de todas las características iniciales.

Existe una función llamada reset que se ejecuta siempre que el avión es puesto al inicio de su recorrido, en ella se ajustan algunos parámetros necesarios para el funcionamiento de cada “round” como el hecho de que se resta 13 a la frecuencia de refrescado o el reacomodo de la segunda cámara a su posición inicial.

Al final se manda llamar una función llamada `callNextTimer`, que si el programa no está en pausa llamará la siguiente función timer indicada por una variable global.

Hay una función para el teclado la cual se encarga de hacer el cambio de cámara, poner en pausa el programa o cambiar la altura de el avión (solo si no está avanzando).

Las funciones de Bezier y DDA tienen cada una variables booleanas que indican si se acabó el movimiento o si es el primer valor, la primera es utilizada para cuando se acaba llamar al siguiente timer, la segunda es para ajustar los parámetros iniciales para las funciones de movimiento.

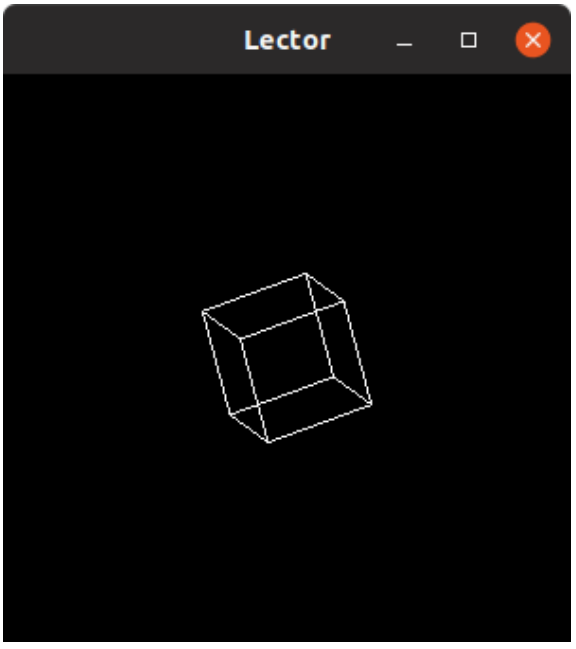
Todos los puntos que corresponden a las curvas son agregados a un vector de vértices con la finalidad de seguir dibujando las trayectorias del paracaídas en cada uno de los “rounds” y así poder compararlas.

En la función de dibujado está el código necesario para que se dibuje todo, esto incluye el fondo azul, el plano verde y el llamado a las funciones necesarias para dibujar los objetos que estén visibles.

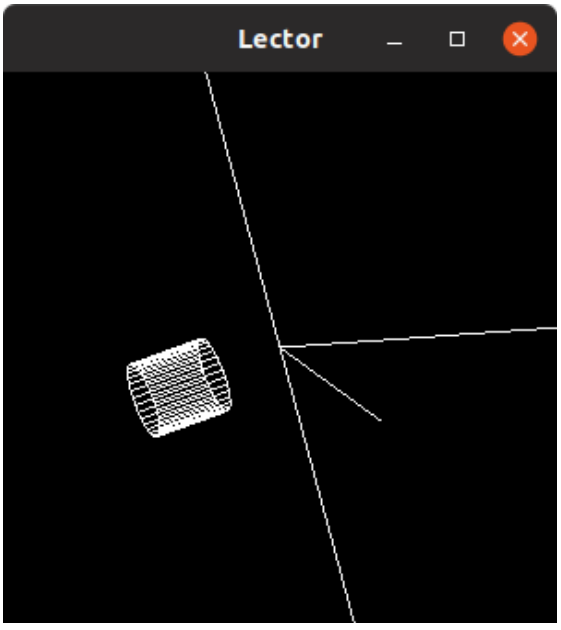
Problemas/ Experimentos

Para la versión uno había un problema, en algunos archivos leia completamente la información y en otros se quedaba ciclado antes de leer las caras, el problema consiste en que para la lectura de caras se aplica un ciclo while que lea entero por entero ya que la cantidad de vértices por cara es indefinida, pero algunos renglones no terminan solo con '\n' si no con "\r\n", por tanto tienen que estar los dos como condición de paro en el ciclo.

Después se realizaron pruebas con diverso archivos .obj en un principio todo iba bien con los archivos de una sola figura.

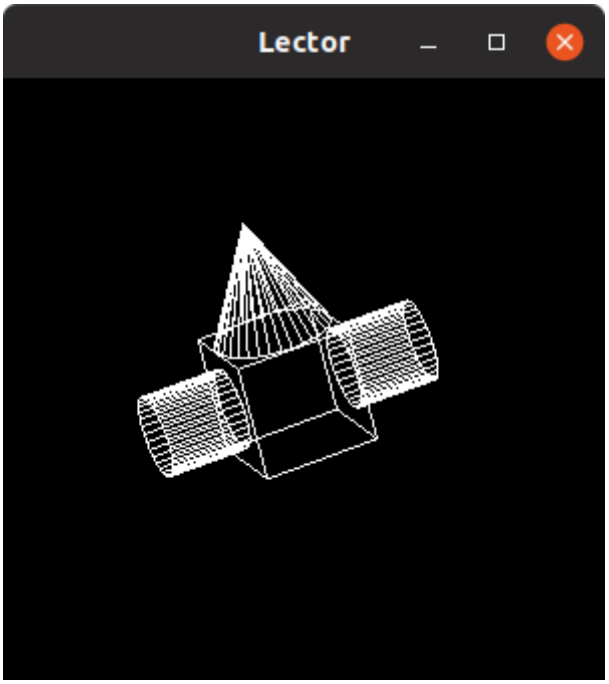


Pero si se leía un archivo que contuviera más de un objeto el dibujo simplemente no se mostraba de la manera correcta.



Después de revisarlo me di cuenta de que es porque los objetos en un archivo comparten numeración, un objeto puede terminar en el vértice 6 y el siguiente empezaría en el 7, sin embargo, en el programa los vértices se obtienen según su índice en el vector de vértices, pero ningún vector comienza en índice 7, por lo cual se introdujeron dos atributos a la clase Object, uno entero para guardar el índice del vértice en el cual empieza la figura y un booleano para saber si el número anterior ya fue asignado.el cual empieza.

Después de esto el programa funcionó correctamente.



Cuando estaba implementando la terminación de superficies visibles y la iluminación el me di cuenta de que por más que el avión se movia las caras nunca cambiaban, esto es porque estaba utilizando el vector que va del origen al PRP en lugar de tomar el vector de uno de los vértices de la cara al PRP, lo solucione obteniendo este último e implementando.

Comentarios finales

Uno de los puntos que se pueden mejorar de este proyecto es el hecho de que las curvas no son tan realistas, si bien se ven bien no están tan apegadas a la realidad, otro punto a tomar en cuenta es que el programa no está libre de errores, cuando se pulsa una tecla para la cual no hay instrucciones en la función del teclado el programa comienza a fallar, por alguna razón algunas variables cambian de valor y entra en ciertos condicionales en donde no debería entrar cuando hipotéticamente no debería hacer nada.

Por último solo quiero decir que si piensas modificar este proyecto ojala sufras tanto como yo jaja, bueno no, quien lea esto ojala que le saque el mayor provecho, tanto si está tratando de aprender o si lo quiere adecuar a sus necesidades.

Manual de usuario

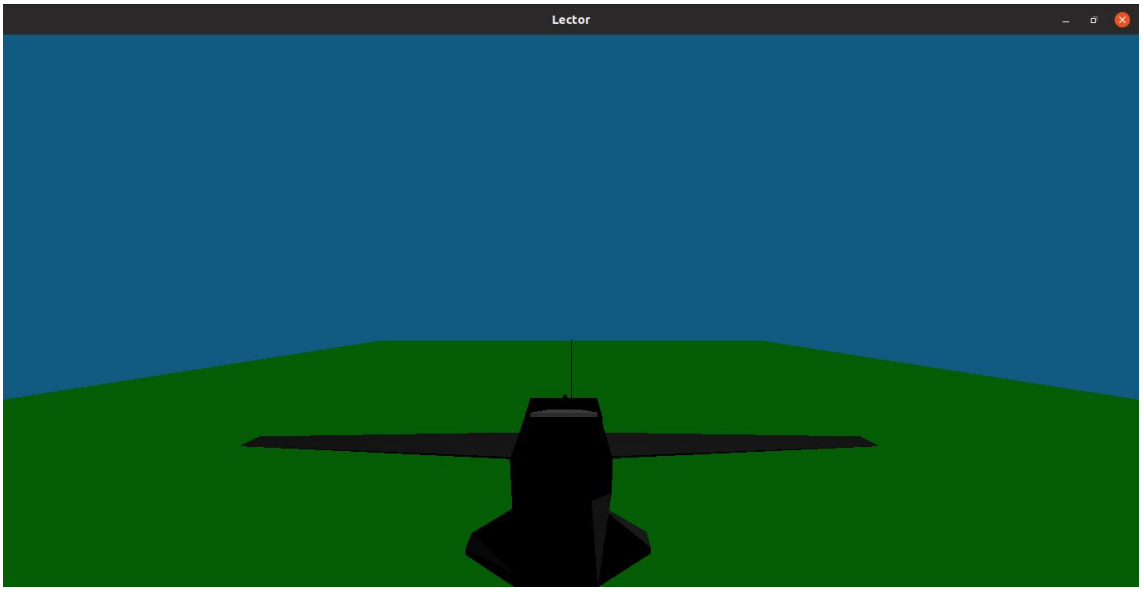
El programa está diseñado para mostrar 3 lanzamientos de paracaídas con una velocidad del avión mayor en cada ocasión, en cada uno de ellos podrás escoger la altura para el avión.

Para ejecutar el programa escribe el siguiente comando en la consola (debes estar en una computadora que corra una distribución de GNU/Linux).

./lectorM

El programa iniciará en esta pantalla:

En este momento tu puedes seleccionar la altura de la trayectoria del avión con las teclas ‘w’ y ‘s’, ‘w’ es para subir el avión y ‘s’ es para bajarlo, también puedes cambiar entre la cámara a distancia y la camara atrás del avión con la tecla ‘c’.



Puedes quitar la pausa del programa para comenzar el movimiento.

Nota: La cámara se puede cambiar en cualquier instante pero es importante poner en pausa el programa antes de hacerlo.

Una vez que tanto el paracaídas llegue al suelo como el avión al final de su recorrido el avión volverá al inicio de su recorrido, donde podrás repetir la selección de altura.

Referencias

OpenGL
Hewlett Packard Enterprise, “OpenGl - The Industry's Foundation for High Performance Graphics”. https://www.opengl.org/

admin, “How to Install OpenGL on Ubuntu Linux”. http://www.codebind.com/linux-tutorials/install-opengl-ubuntu-linux/, 2018

Imagen de paracaídas
Rory J Sánchez, “Aterrizar es lo importante: PARTE 1”. http://www.skydive.es/aterrizar-es-lo-importante-parte-1/