UNIVERSIDAD AUTÓNOMA DE SAN LUIS POTOSÍ

FACULTAD DE INGENIERÍA

ÁREA DE CIENCIAS DE LA COMPUTACIÓN

SISTEMAS INTELIGENTES

Clave única: 291423

Francisco Jacob Flores Rodríguez

GRAFICACIÓN POR COMPUTADORA [09 – 10 AM]

REPORTE DE PROYECTO FINAL DEL SEMESTRE

“REBOTE DE BALÓN CON CURVAS DE BÉZIER”

OMAR RODRIGUEZ GONZÁLEZ

Miércoles, 6 de enero del 2021

INTRODUCCIÓN

A lo largo de nuestra vida muy probablemente hemos visto diversas animaciones en forma de cortometrajes, películas, videos, entre otros, pero tal vez nunca nos hemos puesto a pensar en cómo se hace ese proceso. Uno de los pasos para estos procesos es el cargar archivos que contengan los objetos que se manejarán durante el proyecto. Es aquí en donde entran los archivos OBJ.

Como su nombre lo indica, el lector de OBJ leerá y guardará en memoria archivos con dicha extensión que manejen al menos vértices y caras. Las caras de un polígono están formadas por aristas, los cuales se infieren tomando en cuenta la información dada por la cara, y las aristas están formados por vértices. Además, puede haber vértices normales, vértices de textura, y más datos que son relevantes, pero en el caso del lector no tomaremos en cuenta todos estos elementos.

Todos estos datos que he mencionado están incluidos en los archivos con dicha extensión, delimitados por espacios, diagonales, o algún otro delimitador. Esto define la forma en que funciona el lector, ya que se toman en cuenta los delimitadores para cargar los datos a las instancias que a cada uno le corresponde. Es así como al tener la información cargada en memoria, se podrá dibujar al objeto en pantalla para hacer las respectivas transformaciones y desplazamientos esperados. Es por esto por lo que el lector es parte esencial del manejo de los archivos, ya que permitirán trabajar con los objetos deseados. Es evidente que hay técnicas más avanzadas para el manejo de dichos archivos, y objetos, pero este trabajo nos sirve para entender los elementos de un lector y cómo es que se manejan los archivos OBJ.

Por otro lado, estos objetos que se obtienen de dichos archivos nos funcionan para algo, pero ¿para qué? En este caso para animarlos de la forma en que lo necesitemos. Para esto se hará uso de otras técnicas importantes para el manejo de objetos, y su dibujado. Lo principal, y más importante en este caso, son las transformaciones.

Una transformación permite al objeto cambiar aspectos respecto a su posición, tamaño, y su movimiento en general. Hay 3 tipos de transformación: rotación, traslación, y escala. Estas transformaciones son primordiales a la hora de hacer alguna animación o trabajar con la composición de los objetos, que en este caso son en 3 dimensiones. Pero para esto se hace uso de otra herramienta muy importante: las curvas de Bézier, las cuales nos ayudarán a trasladar el objeto de un punto a otro con curvas, pero de forma sutil.

Ahora bien, el objetivo de este proyecto será hacer uso de las herramientas que se han mencionado. Específicamente, se hará una animación de rebote de un balón tomando un archivo OBJ que contenga la información del balón. Para esta animación se realizarán distintas transformaciones, y se hará uso de las curvas de Bézier. Esto ayudará a mejorar el nivel de comprensión sobre las problemáticas que engloban este tipo de proyectos, y actividades. Así, en lugar de utilizar herramientas que hacen todo esto sin que sepamos qué es lo que sucede, podremos comprender con mayor profundidad cuáles son los procesos detrás del manejo y ejecución de este tipo de trabajos.

DESARROLLO

Para la realización del proyecto, y el cumplimiento de los objetivos establecidos, es necesaria la comprensión de diversos aspectos que serían útiles para lo mencionado. Entre estos se encuentran:

- El lenguaje de programación que fue utilizado, que en este caso C++.
- Las bibliotecas y headers que se utilizaron.
- La plataforma o sistema operativo para el cuál será realizado el programa. En este caso una distribución de GNU/Linux, Xubuntu.
- Software de control de versiones. En este caso utilicé Git para llevar un control de lo que hice a lo largo del tiempo con el proyecto.

- Archivos OBJ. ¿Qué son, para qué sirven, y cómo se manejan?
- Transformaciones: rotación, traslación, y escala.
- Curvas de Bézier.
- Manejo de cámaras.

Habiendo establecido las herramientas utilizadas para la realización del proyecto, y su manejo, hay que definir o indicar cómo se utilizaron, por lo cual, sería mejor conocer de lo más general a lo más concreto.

LENGUAJE DE PROGRAMACIÓN: C++

En esta ocasión, hicimos uso de C++ como lenguaje para programar. Cabe indicar que nunca había trabajado tan a fondo con este lenguaje. Lo más complicado que había hecho para el momento en el que comenzamos con el proyecto fue probablemente el mítico “Hello, world”, y eso sin entender bien la estructura básica del lenguaje. Esto, evidentemente supuso una problemática para mí, ya que tendría que entender las bases e ir aprendiendo sobre la marcha. Aun así, me supuso un reto el tener que aprenderlo mientras iba desarrollando el proyecto, y utilizándolo de una forma algo más compleja.

Cabe agregar que no estuve solo en el proceso de aprendizaje, sino que el profesor nos guio con distintos procedimientos, lo cual ayudó a que el proceso fuera más rápido y digerible. Y así fui aprendiendo, haciendo uso además de los conceptos que ya conocía de la programación, por lo cual, me facilitó mucho su entendimiento. Si bien, en algunos aspectos se maneja una sintaxis distinta a otros lenguajes que había generado con anterioridad, hay muchas palabras reservadas que se comparten con dichos lenguajes.

Bibliotecas utilizadas

Aquí es donde entran las bibliotecas (no librerías) que se utilizaron para lograr lo cometido, las cuales fueron diversas. Es por esto por lo que no está de más definir para qué sirve cada una de las bibliotecas. Además, se hizo uso de bibliotecas que no se escuchan mucho, las cuales será esencial su entendimiento para la comprensión del proyecto, y el código específicamente.

A continuación, estarán listados los headers de las bibliotecas y para qué son de acuerdo con http://www.cplusplus.com/.

1. **<iostream>: Biblioteca de Streams de Entrada / Salida Estándar (Standard Input / Output Streams Library):** Header que define los objetos stream estándar de entrada/salida.
    - **C++98:** Incluir el header podría incluir automáticamente otros header, tales como <ios>, <streambuf>, <istream>, <ostream> y/o <iosfwd>.
    - **C++11:** Incluir el header también incluirá de forma automática <ios>, <streambuf>, <istream>, <ostream> y <iosfwd>.
2. **<string>:** Header que introduce tipos string, rasgos de carácter y un conjunto de funciones de conversión.
3. **<vector>:** Header que define la clase que contiene vector.
4. **<fstream> Streams de archivo (File streams):** Header que contiene las clases de stream de archivos.
5. **<armadillo>:** Es una biblioteca para álgebra linear y computación científica.

En nuestro caso tuvimos que instalar la biblioteca mediante la terminal de Xubuntu.

- **Instalación: sudo apt-get install libarmadillo-dev**

6. **<cmath> (math.h) Biblioteca numérica C (C numerics library):** Header que declara un conjunto de funciones para computar operaciones matemáticas comunes, y transformaciones.

7. **<random>:** Introduce facilidades para la generación de números aleatorios.

8. **<cstdlib> (stdlib.h) Biblioteca de Utilidades Generales Estándar C (C Standard General Utilities Library):** Define muchas funciones de propósito general, incluyendo manejo dinámico de memoria, generación de números aleatorios, comunicación con el entorno, aritméticas de enteros, búsqueda, ordenación, y conversión. Específicamente la utilicé para inicializar un generador de números aleatorios con srand().

9. **<ctime> (time.h) Biblioteca de Tiempo de C (C Time Library):** Incluye definiciones de funciones para obtener y manipular información de la fecha y tiempo. Específicamente utilicé la función time() para obtener la hora actual e inicializar el generador de números aleatorios con srand().

OpenGL

Para las siguientes bibliotecas primero hay que entender qué es OpenGL, por lo que será definido a continuación junto a su uso.

Como se menciona en el sitio web de OpenGL (https://www.opengl.org/about/) es el estándar de gráficos que más ha sido adoptado. Es el entorno principal para desarrollar aplicaciones de gráficos 2D y 3D portátiles interactivos. Desde su introducción en 1992, se ha convertido en la API (Application Programming Interface) de gráficos más utilizada en la industria. OpenGL incorpora un amplio conjunto de poderosas funciones de visualización, tales como: renderizado, mapeado de texturas, efectos especiales, entre otras.

Algunas de sus ventajas son:

- Es el estándar de la industria.
- Es estable.
- Es confiable y portable.
- Sigue evolucionando.
- Es escalable.
- Es fácil de usar.
- Está muy bien documentado.

Habiendo comprendido en grandes términos lo que es OpenGL podemos inferir que es la herramienta con la que se hace el dibujado de los objetos 3D que se obtienen de los archivos OBJ. Después de obtener los datos del objeto a partir del archivo (lo cual se explicará más adelante) se hace uso de las funciones definidas en las bibliotecas de OpenGL en C++. Permite dibujar por triángulos o polígonos (en el caso del objeto que utilicé fue con cuadros) especificando su color, entre otros parámetros. Además, permite su visualización, así como cambiar la perspectiva desde donde vemos los gráficos que manejamos, como otras funciones más complejas.

Después de esta breve explicación, ahora sí podemos explicar qué contiene cada uno de los siguientes headers de OpenGL.

10. **<GL/glu.h> GLU - Utilidades GL (GL Utilities):** Se refieren a un conjunto de funciones de utilidad que hacen que algunas de las operaciones de OpenGL más fácil de programar. Es un estándar en todas las implementaciones de OpenGL, y es generalmente pensado como una parte del estándar OpenGL.

11. **<GLFW/glfw3.h>:** Es el header de la API de GLFW 3. Define todos sus tipos y declara todas sus funciones.

    - **Instalación**

    - sudo apt-get install libglfw3-dev

    - sudo apt-get install libglew-dev

Habiendo definido todas los headers que se utilizaron en la realización del proyecto se logra una mayor comprensión de para qué son útiles en el proyecto. Es verdad que no le resta complejidad, pero así se facilita el proceso de comprensión.

SISTEMA OPERATIVO EN QUE SE TRABAJÓ

Para la ejecución del proyecto trabajamos en una distribución de GNU/Linux: Xubntu. Es un sistema operativo basado en Ubuntu, con la ventaja de que es más ligero. Aunque sea una distribución distinta, trabaja de la misma manera que otras distribuciones en cuanto al manejo de la terminal para compilar y ejecutar código.

Para compilar el código se trabajó con un Makefile, que compila todos los archivos de las carpetas que se indiquen de forma automática (aunque tardada porque cada vez que se ejecuta compila todo de nuevo). Después de esto simplemente se ejecutaba el programa principal (main), que tiene la extensión ".o". Esta extensión, según la respuesta de *community wiki* en https://askubuntu.com/questions/156392/what-is-the-equivalent-of-an-exe-file, es equivalente a la extensión ".dll" en Windows e indica un objeto que puede ser cargado en tiempo de ejecución.

¿Cómo utilizar el Makefile?

En la terminal, habiendo accedido al directorio en donde se encuentra todo lo necesario para que el programa funcione, ejecutar el comando "***make***", el cual compilará todos los archivos que se indiquen. En mi caso modifiqué el código original del Makefile, ya que trabajé con archivos en distintas carpetas.

¿Cómo ejecutar el programa?

Para ejecutar el programa simplemente hay que escribir en la terminal "***./main***".

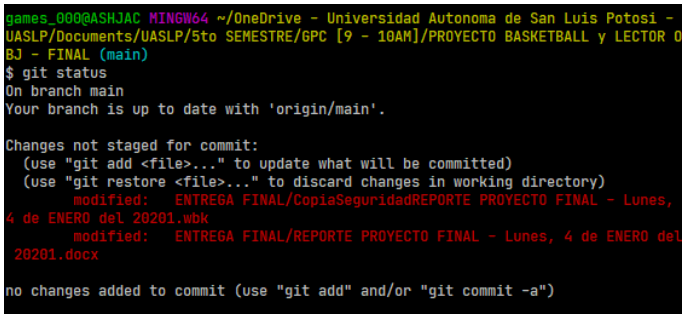
SOFTWARE DE CONTROL DE VERSIONES: GIT

Para llevar un control del trabajo que hice a lo largo del lapso que se nos dio para realizar el proyecto utilicé un software de control de versiones (que en este momento es el único que he utilizado) llamado Git. Git nos permite llevar cuenta del historial de los cambios que hayamos hecho, cuándo subimos estos cambios, qué agregamos, qué quitamos, volver a versiones anteriores, entre muchas más funciones. Es por esto por lo que me pareció factible, ya que fue un proyecto que tomó varios meses finalizarlo, y borrar cosas que ya no necesitaba en el momento, o hacer cambios grandes sin llevar este control de cambios me parece que es fácil cometer un error que lleve a más y más fallos para después terminar confundido por ya no saber qué está sucediendo.

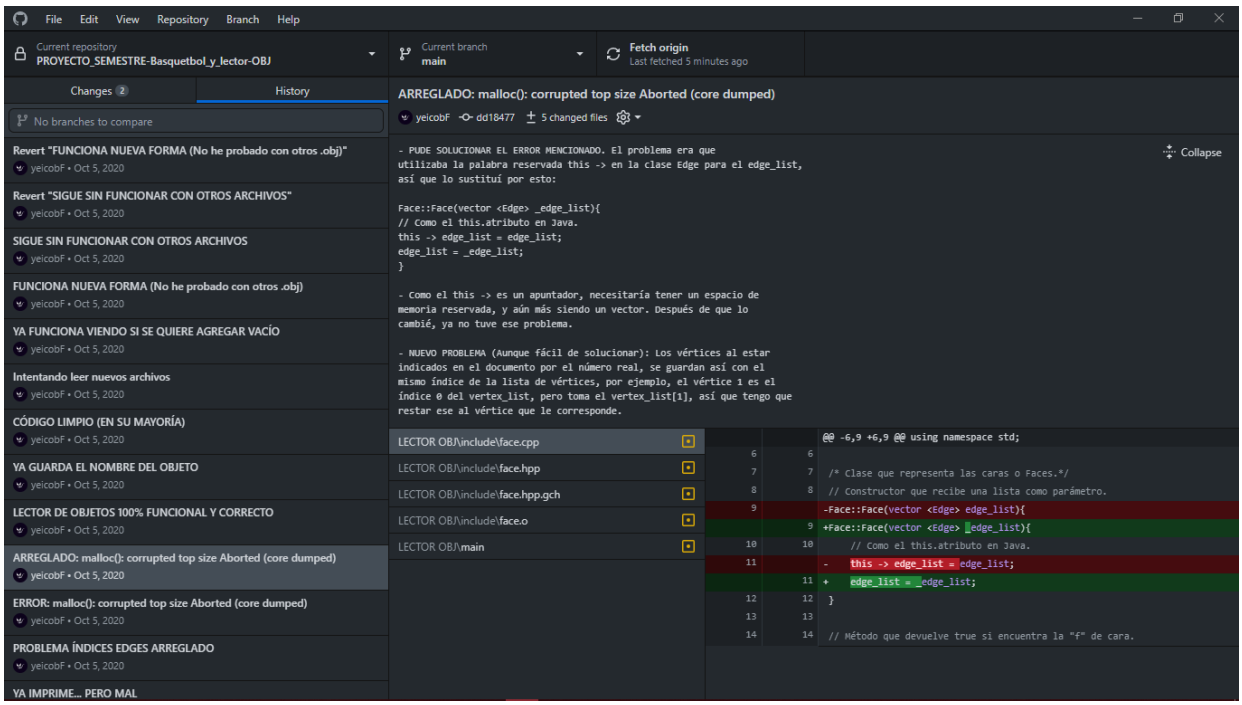
Además, en lo personal me gusta saber qué hice, qué errores encontré, cómo solucioné dichos errores, y más, por lo que es una herramienta que me viene muy bien. Estos cambios se quedan guardados en el servidor remoto en el repositorio que hayas creado en https://github.com/, pero los tienes que subir tú para que se vean reflejados.

Git como tal es el software que maneja todo lo mencionado, pero para hacer uso de él hay varias maneras, y las que utilicé fueron 2:

- **Git Bash:** Es un intérprete de comandos, en donde tienes todas las opciones disponibles de Git (que hayas instalado), por lo cual tienes una gran libertad para hacer uso de las herramientas de forma manual, logrando resultados más precisos con lo que quieras hacer.

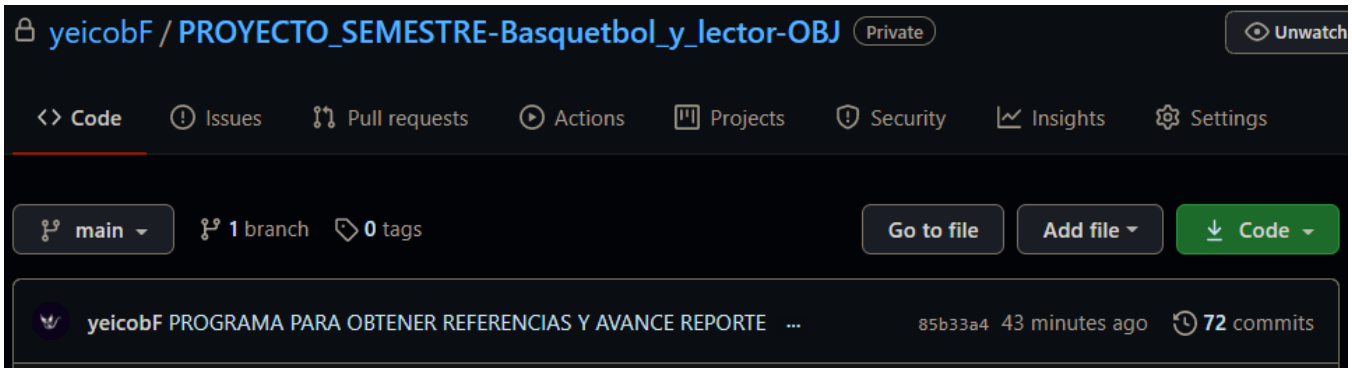


- **GitHub Desktop:** Es una interfaz gráfica de usuario que hace uso de Git, pero de forma mucho más sencilla pero más limitada al mismo tiempo. No tienes tantas opciones disponibles como haciendo uso de los comandos de manera directa. Aun así, es muy útil cuando no requieres de operaciones más complejas que subir y revisar los cambios que hayas realizado.



Haciendo uso de estas herramientas no solo me ayudó a guardar el programa e incluso trabajarlo desde otra computadora, sino que me ayudó a cuantificar y comprender con mayor facilidad los errores que cometía. Esto se debe a que en la mayoría de los cambios (en Git se les llama commits al contenido que se va subiendo) escribía un resumen de los errores que me había encontrado y de cosas que me faltaban por agregar o revisar, así que al

relatarlo me daba cuenta de lo que debía modificar y demás. Es cierto que rara vez volví a leer los escritos que hacía, pero aun así me pueden ser útiles en un futuro para saber cómo resolver ciertas problemáticas que algún día me encontré, o simplemente saber qué hice en cierto tramo del tiempo.



LECTOR DE ARCHIVOS OBJ

Entrando en materia en cuanto al desarrollo específico del proyecto y el funcionamiento del programa, no podemos dejar atrás al lector de archivos OBJ. Como bien indica el mismo nombre, su utilidad es recorrer todo un archivo OBJ y obtener sus datos para después guardarlo en memoria. Los datos que guarda mi lector son:

- Vértices con sus coordenadas (x, y, z).
- Los vértices que componen cada cara.
- Además, las aristas que unen a las caras se infieren a partir de los datos que se dan.

Eso es en rasgos generales. El problema con el lector es que si la fila que contiene los datos de las caras tiene espacios al final (después de los últimos datos), se empiezan a guardar los datos de forma errónea. Pero mientras no existan dichosos espacios al final de las filas mencionadas no hay problema. Probé con objetos de 3 y 4 aristas por cara y los lee de forma correcta mientras se respete lo dicho.

Explicación de un archivo obj

Ahora bien, los archivos OBJ tienen una estructura específica que indica cómo se maneja cada elemento del objeto, lo cual fue útil para la comprensión de estos y la realización del lector en sí. Es de esta forma como se logra el dibujado del objeto en 3D utilizando OpenGL, y su posterior manipulación.

Un archivo OBJ está organizado de diversas maneras, pero todas siguiendo una nomenclatura ya establecida, aunque a nosotros solo nos importan algunas para el proyecto. Estas van indicadas en cada línea por letras que definen cada tipo de datos:

- **o:** Nombre del objeto

```
o BasketballBall
```

- **g:** Nombre del grupo (también lo tomamos como nombre del objeto en el proyecto).

```
g cube
```

- **f:** Una cara del objeto.

```
f  1//2  7//2  5//2
f  1//2  3//2  7//2
f  1//6  4//6  3//6
f  1//6  2//6  4//6
f  3//3  8//3  7//3
f  3//3  4//3  8//3
f  5//5  7//5  8//5
f  5//5  8//5  6//5
f  1//4  5//4  6//4
f  1//4  6//4  2//4
f  2//1  6//1  8//1
f  2//1  8//1  4//1
```

- **v:** Un vértice con sus coordenadas (x, y, z, w (opcional)).

```
v  0.0  0.0  0.0
v  0.0  0.0  1.0
v  0.0  1.0  0.0
v  0.0  1.0  1.0
v  1.0  0.0  0.0
v  1.0  0.0  1.0
v  1.0  1.0  0.0
v  1.0  1.0  1.0
```

- **vn:** vértice normal.

```
vn  0.0  0.0  1.0
vn  0.0  0.0 -1.0
vn  0.0  1.0  0.0
vn  0.0 -1.0  0.0
vn  1.0  0.0  0.0
vn -1.0  0.0  0.0
```

TRANSFORMACIONES

Ya se ha definido la primera parte del proyecto, pero aún queda lo que logrará que se cumpla el propósito del proyecto: el movimiento de nuestro objeto. Pero ¿cómo es que se logra esto? No es trivial, ya que requiere del entendimiento y manejo de un concepto muy importante, técnicamente primordial, junto a sus cálculos y procedimientos. Este concepto se conoce como "***transformaciones***".

Según la información contenida en el sitio web http://prepa8.unam.mx/academia/colegios/matematicas/paginacolmate/applets/matematicas_VI_4/Applets_Geogebra/transfgeom.html, "las transformaciones geométricas son las operaciones que permiten crear una nueva figura *homóloga*, a partir de una previamente dada". Es decir, podemos modificar una figura que en este caso sería el objeto obtenido del archivo OBJ, para cambiar su composición geométrica.

En nuestro caso hicimos uso de 3 tipos de transformaciones geométricas básicas:

1. **Traslación:** Permite cambiar la posición de la figura. Se mueve el objeto.

2. **Escalación:** Permite cambiar el tamaño de la figura de acuerdo con una escala. Se puede hacer más grande o chica.

3. **Rotación:** Permite girar la figura.

Haciendo uso de las transformaciones básicas se crea una transformación compuesta. Estas transformaciones harán que el objeto presente movimiento y cambios en su estado (geométricamente hablando). Es así como nos acercamos cada vez más al entendimiento de cómo lograr el objetivo: simular los rebotes de un balón.

Curvas de Bézier

Ahora bien, ya sabemos que podemos mover el objeto, cambiarlo de tamaño, e incluso rotarlo, pero surge otra pregunta: ¿cómo sabemos hacia dónde moverlo, y cómo hacemos que tenga una trayectoria curva? Y es aquí en donde está la importancia de las curvas de Bézier.

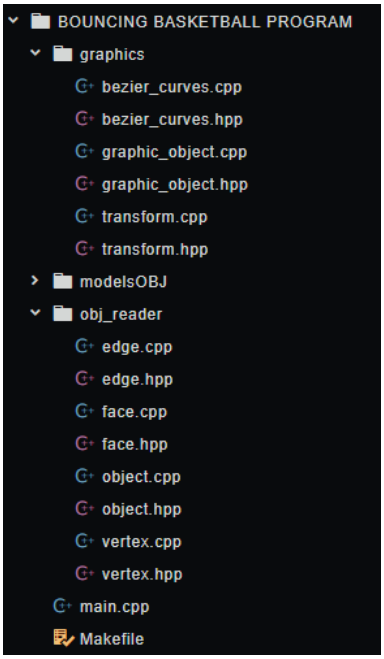
Las curvas de Bézier generan trayectorias que nos permiten que el objeto tenga un movimiento suave al tratarse de curvas, valga la redundancia. Estas se calculan haciendo uso de un punto inicial, dos puntos de control intermedios, y un punto final. Haciendo uso de dichos parámetros además de una matriz de Bézier, se calculan diversos puntos a lo largo de la trayectoria que se quiere seguir, para luego trasladar al objeto en cada uno de los puntos calculados. Entre más puntos, mayor suavidad habrá al mover al objeto.

ORGANIZACIÓN DEL PROGRAMA / PROYECTO – CARPETAS Y ARCHIVOS

Para realizar el programa se necesitaron muchos archivos para separarlo por clases. Estos archivos son los headers (.hpp), y los que implementan los métodos de los headers (.cpp). Pero planteándolo así habría terminado con muchos archivos y todos en el mismo lugar, lo cual sería un desorden total. Es así como se me ocurrió la idea de dividir los archivos por carpetas que tengan significancia sobre los archivos que están ahí, y así tener un mejor control sobre el contenido.

Específicamente los separé en dos carpetas que definiré a continuación junto con los archivos:

1. **graphics:** En esta carpeta se encuentran los archivos que tienen que ver con el manejo del ámbito gráfico: dibujado del objeto, movimiento, cálculo del movimiento.
    - **transform**
    - **bezier_curves**
    - **graphic_object**
2. **obj_reader:** En esta carpeta se encuentran los archivos de las clases necesarias para el manejo del lector OBJ.
    - **vertex**
    - **edge**
    - **face**
    - **object**



Además, puse el archivo OBJ que utilicé en la carpeta:

3. **modelsOBJ**

En la carpeta principal se encuentra el main.

ESTRUCTURAS DE DATOS UTILIZADAS

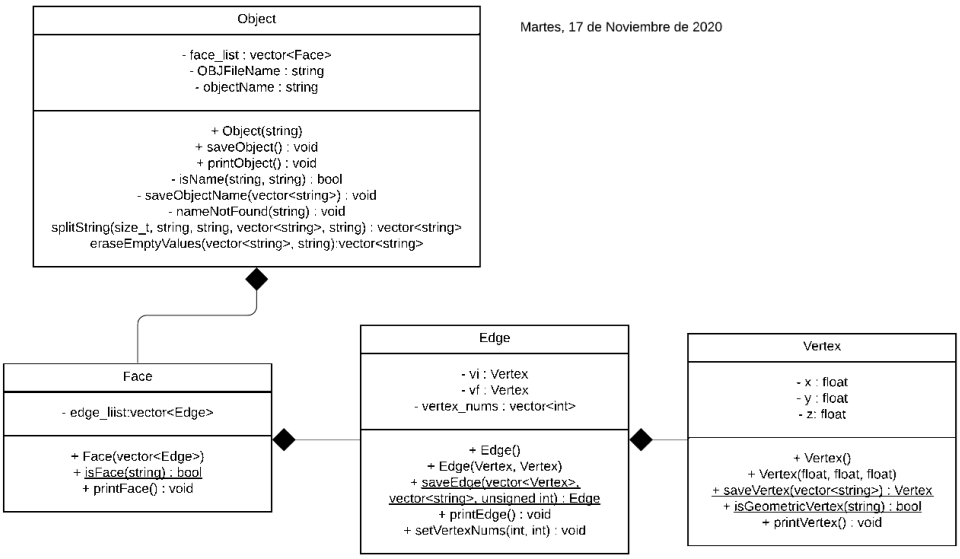
- **Vector:** Un vector es una estructura de datos similar a los arreglos. A diferencia de los arreglos, tienen la capacidad de crecer o decrecer dinámicamente. También es algo parecida a una lista por esta última característica mencionada. Se utiliza para guardar la lista de caras, de vectores y de vértices.

CLASES Y MÉTODOS

Diagramas de clases

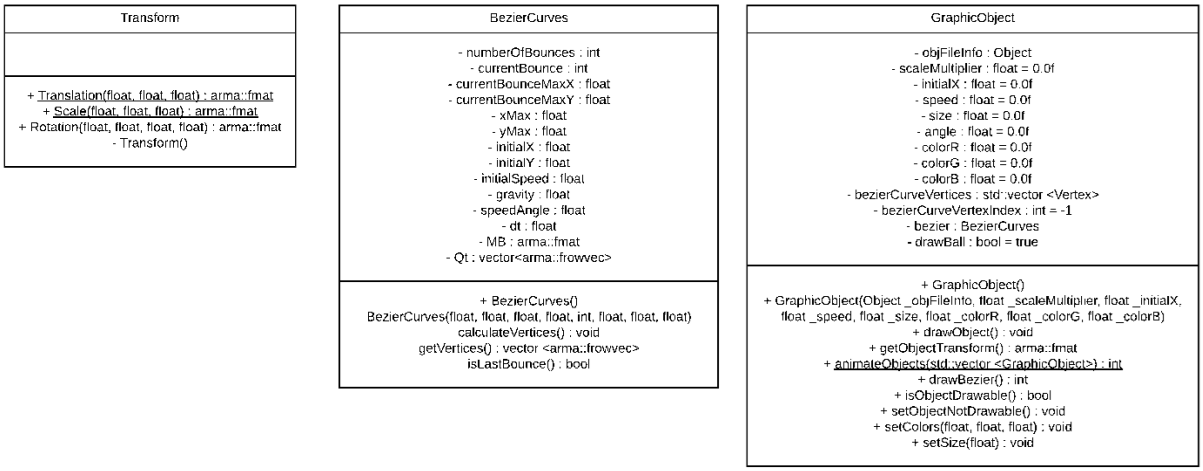
Para definir de una mejor forma las clases, y conocerlas de forma estructurada y relacional, es bueno tener en cuenta los diagramas de clases. Estos los realicé en Lucidchart, cuyo sitio web es: **https://lucid.app/.**

Lector de archivos OBJ



En el diagrama de clases se muestran las 4 clases que funcionan para trabajar con los archivos OBJ. Las flechas que se ven en el diagrama son un tipo de Aggregation, específicamente Composition. Esta indica que el hijo no puede existir independientemente del padre. En este caso la generalización es el Objecto, pero como no manejé herencia, esta es la forma en la que supe representar la relación que hay entre las clases, pero probablemente esté incorrecto.

Gráficos



Explicación clases y sus métodos

Para cumplir una buena encapsulación y un manejo con mayor orden de las instancias el proyecto debe ser dividido en clases que sean coherentes con lo que se está manejando. Es por esto por lo que decidí hacer diversas clases específicas para cada elemento en concreto. Hay métodos que regresan diversos tipos de datos, e incluso manejé tipos estáticos, los cuales no requieren de una instancia para ser accedidos.

Con la finalidad de comprender mejor la estructura de mi proyecto, definiré las clases y los métodos que cada una de estas manejan a grandes rasgos. No iré a lo más detallado porque lo importante es que se entienda qué se hace, más no cómo se hace (en esta ocasión), ya que eso está indicado en el código del programa.

Lector de archivos OBJ

- ***Vertex*:** Esta clase manejará los vértices, es decir, sus coordenadas.
    - ***Vertex():*** Constructor de vértice sin parámetros para no tener problemas en clase Edges.
    - ***Vertex(float xi, float yi, float zi):*** Constructor de vértice que recibe coordenadas.
    - ***static Vertex saveVertex(std::vector <std::string> values):*** Método estático para guardar valores de vértice y regresarlos.
    - ***static bool isGeometricVertex(std::string str):*** Método que regresa true si la cadena que se envió se refiere a un vértice en el archivo OBJ.
    - ***void printVertex():*** Método que imprime las coordenadas del vértice.
    - ***void setVertex(arma::frowvec _vertex):*** Método para establecer un vértice recibido por parámetro.
    - ***arma::frowvec getVertex():*** Método para obtener el vector de coordenadas del vértice.
    - ***arma::fcolvec getHomogeneousCoordinates():*** Método para obtener las coordenadas homogéneas del vértice.
- ***Edge*:** Esta clase manejará las aristas del objeto. Guardará los números de vértice de las aristas y sus vértices como tal.
    - ***Edge():*** Constructor para no tener problemas en otras clases.
    - ***Edge(Vertex va, Vertex vb):*** Constructor de aristas con sus vértices correspondientes.
    - ***static Edge saveEdge(std::vector <Vertex> v_list, std::vector <std::string> values, unsigned int actualIndex):*** Método estático que guarda una arista. Se puede invocar sin instanciar al objeto.
    - ***Vertex getStartVertex():*** Método para obtener el primer vértice de la arista.
    - ***void printEdge():*** Método para imprimir la arista y sus vértices.
    - ***void setVertexNums(int v1, int v2):*** *Método para establecer los números de los vértices de la arista.*
- ***Face*:** Esta clase manejará las caras del objeto y guardará las aristas que forman las caras.

- ***Face(std::vector <Edge> _edge_list):*** Constructor que recibe una lista de aristas.
- ***static bool isFace(std::string str):*** Método que verifica si la cadena mandada indica una cara en el formato de un archivo OBJ.
- ***std::vector <Edge> getFaceEdges():*** Método que regresa las aristas de una cara.
- ***void printFace():*** Método que imprime el número de aristas en una cara, e imprime cada una de las aristas.

- ***Object*****:** Es la clase que manejará al objeto con todos sus elementos, es decir, sus caras, aristas, y vértices.
  - ***bool isName(std::string str, std::string* notEnterValues):*** Método que verifica si la línea actual lee un nombre.
  - ***void saveObjectName(std::vector <std::string> values):*** Método que guardará el nombre del objeto.
  -
  - ***void nameNotFound(std::string OBJFileName):*** Método que asigna el nombre el del archivo al objeto si no se especifica el nombre en el archivo.
  - ***std::vector <std::string> splitString(size_t pos, std::string delimitador, std::string linea, std::vector <std::string> values, std::string notEnterValues):*** Método que parte una cadena dada y regresa un vector de tipo string.
  - ***std::vector <std::string> eraseEmptyValues(std::vector <std::string> values, std::string valueToErase):*** Método que quitará los espacios en blanco o cosas que estorben en un vector.
  - ***Object():*** Constructor vacío para no tener problemas al compilar y correr.
  - ***Object(std::string _OBJFileName):*** Constructor que recibe el nombre de un archivo OBJ.
  - ***void saveObject():*** Método que guardará la información del archivo OBJ del objeto en memoria.
  - ***void printObject():*** Método que imprimirá los datos del objeto: Nombre, caras con sus aristas y vértices, y el número total de caras del objeto.
  - ***std::vector <Vertex> getFacesVertices():*** Método que regresa todos los vértices del objeto.

Gráficos

- ***Transform:*** Clase que maneja las transformaciones: traslación, escalación, y rotación. Técnicamente es una clase estática (o abstracta, depende del concepto que se maneje) porque no se requiere instanciar la clase para poder acceder a sus métodos. No es necesaria la instancia, ya que solo queremos los métodos, pero no ningún atributo o algo más.
  - ***Transform():*** Constructor privado para evitar que la clase se instancie.
  - ***static arma::fmat Translation(float tx, float ty, float tz):*** Método para hacer una traslación, mover el objeto***.***

- ***static arma::fmat Scale(float sx, float sy, float sz):*** Método para hacer una escalación, cambiar de tamaño el objeto.

- ***static arma::fmat Rotation(float ax, float ay, float az, float angle):*** Método para rotar el objeto.

- ***BezierCurves:*** Clase que calcula y regresa las coordenadas de las curvas de Bézier.

  - ***BezierCurves():*** Constructor vacío para no tener problemas al compilar.

  - ***BezierCurves(float _initialX, float _initialY, float _initialSpeed, _speedAngle, int _numberOfBounces, float gravity, float _yMax, float _dt):*** Constructor que se va a utilizer para establecer el valor inicial de los atributos de las curvas de Bézier.

  - ***void calculateVertices():*** Método para calcular los vértices de la curva de Bézier.

  - ***std::vector <arma::frowvec> getVertices():*** Método para obtener los vértices de la curva de Bézier calculada.

  - ***bool isLastBounce():*** Método para revisar si ya se hizo el último rebote del objeto. Este método puso haber estado en otra clase, pero al final siendo que manejé los rebotes en la clase de las curvas de Bézier, lo dejé ahí.

- ***GraphicObject:*** Clase que maneja al objeto en el aspecto gráfico / visual. Engloba el dibujado del objeto, la visualización en pantalla, el movimiento de las cámaras, entre otros elementos más.

  - ***GraphicObject():*** Constructor vacío para no tener problemas con el compilador.

  - ***GraphicObject(Object _objFileInfo, float _scaleMultiplier, float _initialX, float _speed, float _size, float _colorR, float _colorG, float _colorB):*** Constructor que recibe los atributos y características del objeto gráfico a manejar.

  - ***void drawObject():*** Método que dibuja al objeto en pantalla haciendo uso de OpenGL.

  - ***arma::fmat getObjectTransform():*** Método para obtener la transformación de un objeto.

  - ***static int animateObjects(std::vector <GraphicObject> object_list):*** Método para hacer todo el proceso gráfico: dibujado, animación, y más. Es como el main pero de la clase, porque en este método se ejecuta todo lo importante para el manejo de los objetos y ventanas.

  - ***int drawBezier():*** Método que dibuja a los objetos tomando en cuenta las curvas de Bézier.

  - ***bool isObjectDrawable():*** Método que revisa si el objeto aún se puede dibujar. Esto se vuelve true cuando ya se hicieron los rebotes que se indicaron en el constructor.

  - ***void setObjectNotDrawable():*** Método que establece que el objeto ya no se podrá dibujar.

  - ***void setColors(float _colorR, float _colorG, float _colorB):*** Método para cambiar el color del objeto.

  - ***void setSize(float _size):*** Método para cambiar el tamaño del objeto.

REBOTES Y CÁLCULOS PARA EL MOVIMIENTO DEL OBJETO

Como ya he mencionado en diversas ocasiones, el objetivo del proyecto es mostrar un balón en pantalla y que este rebote. Al inicio tenía en mente implementarlo utilizando fórmulas físicas utilizando parámetros como velocidad, tiempo, alturas, y más, pero al final lo implementé de una forma distinta. Tomando esto en cuenta, indicaré el enfoque que utilicé para hacer los cálculos de las curvas.

1. Se reciben los parámetros en el constructor:
    - **_initialX:** x inicial.
    - **_initialY**: y inicial.
    - **_initialSpeed:** velocidad inicial.
    - **_speedAngle:** ángulo de velocidad.
    - **_numberOfBounces:** número de rebotes totales.
    - **gravity:** gravedad.
    - **_yMax:** y máxima, es decir, la altura máxima.
    - **_dt:** Incremento entre punto y punto en la curva para hacer el cálculo.
2. Se establece el número de rebote actual.
3. Se calcula la x máxima hasta donde llegará el último rebote.
4. Después, al calcular los vértices de cada curva se obtendrá la x máxima de cada rebote utilizando como referencia el número de rebote actual.
5. Se definen los 4 puntos de control.
6. Se calcula el punto máximo en y del siguiente rebote, el cual no se basará en una fórmula física, sino en el número de rebote actual. Disminuirá su altura de forma uniforme. Cada rebote perderá la misma altura.
7. Después se calculan los puntos de la curva.

Es así como calculé las trayectorias y los rebotes, tomando como referencia el número actual de rebote, pero sin tomar en cuenta factores como la velocidad, gravedad, y más. A pesar de esto, me habría gustado darle un enfoque más preciso, pero lamentablemente me fue difícil conseguirlo.

RESULTADO FINAL Y EJECUCIÓN DEL PROGRAMA

Después de todo el trayecto que recorrí durante los últimos meses llegué al punto en que terminé el programa, pero sentí que algo le faltaba. Al principio solamente generaba un balón y rebotaba con lo que le indicaba, pero le faltaba algo. De esta manera es como llegué a la conclusión de que agregaría otro balón, aunque no era suficiente. Luego pensé en que sería interesante que al presionar una tecla se generara un balón, luego otro, y así sucesivamente. Es así como al final implementé algo que no había pensado desde el inicio: generar un balón por la izquierda al presionar la tecla “enter” y otro a la derecha al presionar “backspace”, la tecla de retroceso. Además, los balones aparecerían tomando como referencia dos balones iniciales que son constantes, pero cambiando su tamaño y su color.

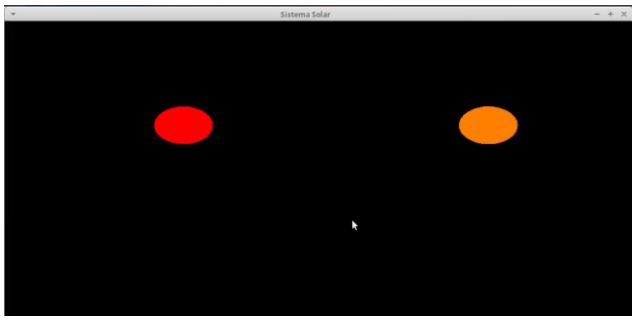
Además, la cámara se puede cambiar para ver todo desde una distinta perspectiva. Esta se puede invertir, ver desde arriba, o volver a la cámara original dependiendo de qué letra presiones. Específicamente se obtiene presionando las teclas:

- **“I”:** *Invertir cámara.*
- ***“D”:*** *Ver desde arriba.*
- ***“O”:*** *Volver a la perspectiva original.*

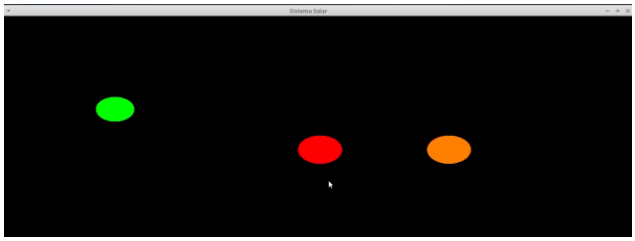
Tal vez no sea lo más innovador y complejo del mundo, pero me pareció algo interesante que le daba más vida al proyecto de alguna forma.

Capturas de pantalla

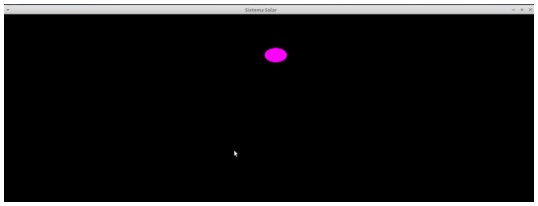
Dejando de lado esto, adjuntaré capturas de pantalla del resultado final del programa. Como las imágenes no caben en la parte inferior de la hoja, las agregaré en la siguiente página.



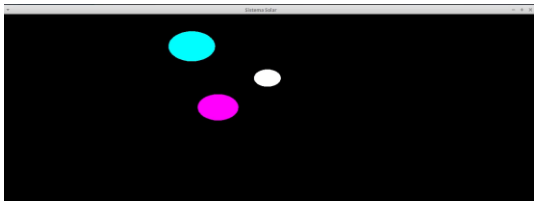
Los dos balones generados al inicio de forma constante



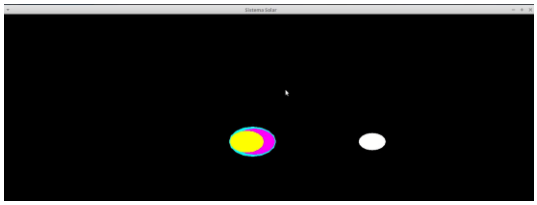
Presioné “enter” y se generó un balón verde por la izquierda.



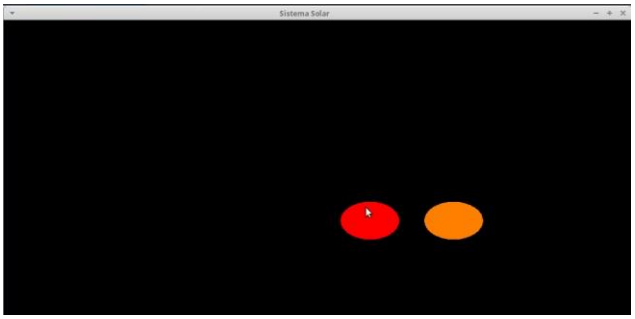
Presioné “backspace”, por lo que se generó un balón rosa por la derecha.



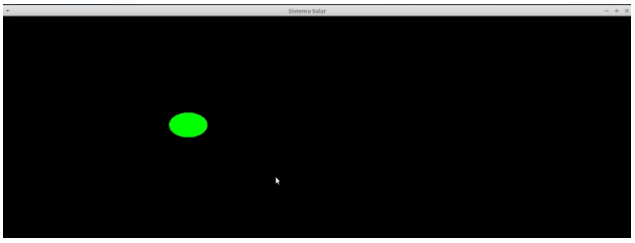
Generé más balones.



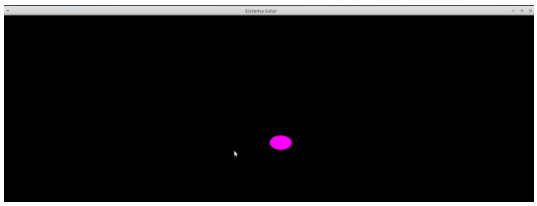
Vista desde arriba.



Los balones a punto de tocar el piso.



Presioné la tecla “I”, por lo que se invirtió la cámara.



Presioné “D”, por lo que se ve el balón desde abajo.

PRUEBAS

Las pruebas son esenciales porque nos permiten saber qué errores cometimos, qué salió bien, qué no, y así sucesivamente. Hay ocasiones en que creí que había hecho las cosas bien, pero al momento de ejecutar el programa no era así. Es por esto por lo que a continuación, adjuntaré algunas de las pruebas que realicé.

```
int main(){
    // Utilizar el constructor con el nombre del archivo.
    Object obj = Object("modelsOBJ/OrangeCartoon_basketball_ball_OBJ.obj");
    obj.saveObject();
    // std::cout << "\n Terminó de guardar el objeto.\n";
    obj.printObject();
}
```

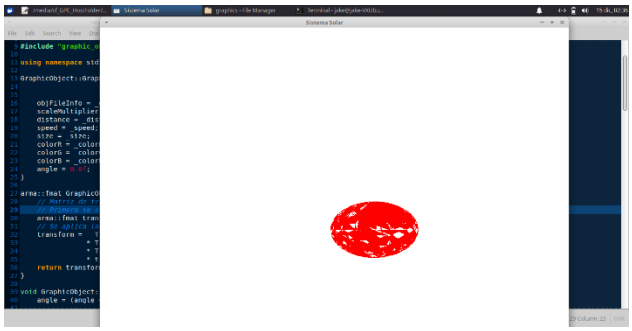
Para comenzar con las pruebas primero debemos indicar la carpeta en donde se encuentra el archivo OBJ y el nombre del archivo. Es así como se guardará en memoria y se podrá manipular.

```
STRE-Basquetbol_y_lector-OBJ/LECTOR OBJ$ make
g++ -I./   -c -o include/object.o include/object.cpp
g++ main.o include/face.o include/edge.o include/vertex.o include/object.o -o main -l glfw -l
GLEW -l GL -l GLU -l armadillo
jake@Jake-XUbuntu20:/media/sf_GRAFICACION_POR_COMPUTADORA_[9-10AM]/PROYECTO_SEMESTRE-Basquetbo
l_y_lector-OBJ/LECTOR OBJ$ ./main
```

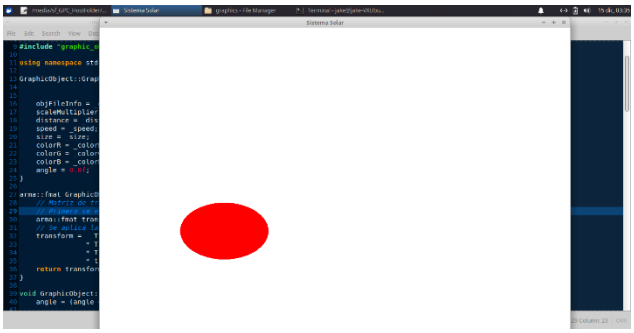
En las siguientes imágenes se encuentra la prueba que hice con el archivo OBJ que utilizaré para realizar el proyecto. Después de guardar la información con Object::saveObject(), la muestra con el método de impresión. De esta forma sabemos que se guardaron los datos de forma correcta. Específicamente se muestran el número de cara, el total de aristas, y los vértices que forman a cada arista.

```
- CARA 644

-> NÚMERO de ARISTAS en esta cara: 4

        -> ARISTA: 1

- VÉRTICES: 618, 620

Vertice inicial:
 -> x = -0.010843, y = -0.346438, z = -0.14129

Vertice final:
 -> x = -0.010843, y = -0.311769, z = -0.206161


        -> ARISTA: 2

- VÉRTICES: 620, 621

Vertice inicial:
 -> x = -0.010843, y = -0.311769, z = -0.206161

Vertice final:
 -> x = 0.010835, y = -0.311769, z = -0.206161


        -> ARISTA: 3

- VÉRTICES: 621, 619

Vertice inicial:
 -> x = 0.010835, y = -0.311769, z = -0.206161

Vertice final:
 -> x = 0.010835, y = -0.346438, z = -0.14129


        -> ARISTA: 4
 -> x = 0.010835, y = -0.208317, z = -0.309613

Vertice final:
 -> x = -0.010843, y = -0.208317, z = -0.309613

----------------------------


 - CARA 648

 -> NÚMERO de ARISTAS en esta cara: 4

        -> ARISTA: 1

 - VÉRTICES: 626, 628

Vertice inicial:
 -> x = -0.010843, y = -0.143491, z = -0.344263

Vertice final:
 -> x = -0.010843, y = -0.073151, z = -0.3656


        -> ARISTA: 2

 - VÉRTICES: 628, 629

Vertice inicial:
 -> x = -0.010843, y = -0.073151, z = -0.3656

Vertice final:
 -> x = 0.010835, y = -0.073151, z = -0.3656


        -> ARISTA: 3

 - VÉRTICES: 629, 627

Vertice inicial:
 -> x = 0.010835, y = -0.073151, z = -0.3656
 -> x = -0.010829, y = -0.361721, z = -0.098207


        -> ARISTA: 2

 - VÉRTICES: 617, 616

Vertice inicial:
 -> x = -0.010829, y = -0.361721, z = -0.098207

Vertice final:
 -> x = 0.010821, y = -0.361721, z = -0.098207


        -> ARISTA: 3

 - VÉRTICES: 616, 663

Vertice inicial:
 -> x = 0.010821, y = -0.361721, z = -0.098207

Vertice final:
 -> x = 0.010804, y = -0.368339, z = -0.070605


        -> ARISTA: 4

 - VÉRTICES: 663, 662

Vertice inicial:
 -> x = 0.010804, y = -0.368339, z = -0.070605

Vertice final:
 -> x = -0.010809, y = -0.368473, z = -0.070009

----------------------------

 - ESTE OBJETO TIENE 662 CARAS -
```

Además de las pruebas indicadas con anterioridad, a lo largo del lapso en el que trabajé con el proyecto, realicé múltiples pruebas para ver que lo que había hecho funcionaba. Obtuve como resultado muchos fallos, pero también resultados satisfactorios que me guiaron para el resultado final del proyecto.



Esto fue cuando logré que se dibujara el balón, pero mostrando fallos en el dibujado.



Esto fue de cuando logré finalmente que el balón se dibujara sin ningún tipo de hueco o fallo.

Como en muchos proyectos, y más en los que requieren una cantidad de tiempo y trabajo considerable, se requieren hacer muchas pruebas para comprobar que lo que se ha hecho funciona. Es por eso por lo que, al menos en mi caso hice demasiadas pruebas, ya que a veces lo que había hecho funcionaba, pero otra cosa ya no, ocurrían errores de segmentación y más, por lo que si adjuntara capturas de pantalla y documentara todas las pruebas el documento sería muy extenso.

Aun así, gracias a todas las pruebas que realicé llegué al resultado final del programa, el cual fue satisfactorio para mí. Es verdad que pude haber hecho algunas cosas de maneras distintas, pero al final resultó como lo deseaba.

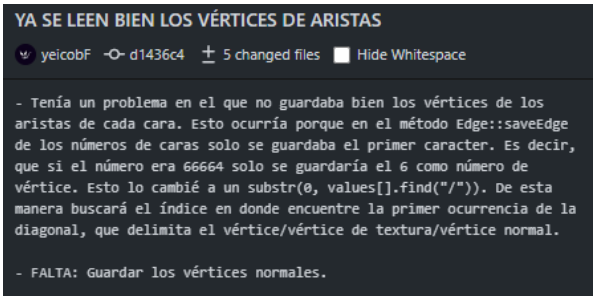
PROBLEMAS ENCONTRADOS Y EXPERIMENTOS

Los problemas son muy habituales a la hora de realizar un proyecto, que en este caso se enfocó en la programación. Son diversos los factores por los que suceden estos fallos, al igual que son numerosos los errores y problemas. Es por esto por lo que a continuación, indicaré algunos de los problemas con los que me encontré y sus soluciones. Siendo certero, hay problemas que no les encontré solución de forma específica, pero cambié el enfoque o simplemente tanteé hasta que lo que quería quedaba de otra mejor forma.

Lector OBJ

En la realización del lector me encontré con diversos problemas, entre ellos el que no funciona con todos los archivos OBJ. Esto me hizo optar a hacer un repositorio en Git (como mencioné con anterioridad) para mantener un control de los cambios que he realizado y los problemas con los que me iría encontrando.

- No guardaba correctamente los números de vértices de las aristas. Esto ocurría porque en el método que realizaba estas operaciones solamente se guardaba el primer carácter de la subcadena del vértice. Lo solucioné cambiando la implementación a cuando encontrara la ocurrencia de la diagonal.



- Si al final de una fila de datos hay un espacio, este genera problemas.
- Problemas con la impresión, ya que se guarda la "f" de las caras en el vector de values (el cual es de tipo string). Esto lo solucioné poniendo el valor que no se debía ingresar en cada caso (en este caso la f), así no la agregaría al vector y solo agregaría los valores.
- Al correr el programa, cuando entra a hacer el setVertexNums(); lanza este error:

malloc(): corrupted top size

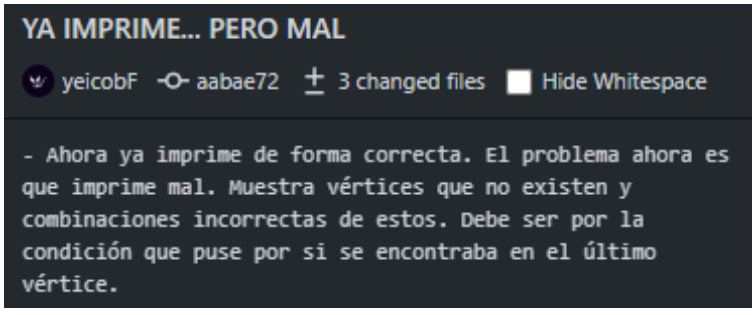
Aborted (core dumped)

El problema era que utilizaba la palabra reservada this -> en la clase Edge para el edge_list, así que lo sustituí por esto:

```
Face::Face(vector <Edge> _edge_list){

// Como el this.atributo en Java.

// this -> edge_list = edge_list;

edge_list = _edge_list; }
```

Como el this -> es un apuntador, necesitaría tener un espacio de memoria reservada, y aún más siendo un vector. Después de que lo cambié, ya no tuve ese problema.

- Adjunto una captura de pantalla de algunos errores con los que me encontré (no son muy específicos en el título, pero sí en la descripción):



Gráficos

También me encontré con muchos problemas y complicaciones a la hora de trabajar con el aspecto gráfico, ya que es algo con lo que no me logré aclarar en muchos aspectos. Una de las cuestiones con las que si bien, no tuve fallos en cuanto a programación, pero sí en cuanto a resultado. Esto son las curvas de Bézier.

- **Curvas de Bézier**

Las curvas de Bézier conceptualmente no son del todo difíciles de entender, pero cuando se trata de la programación te puedes encontrar con muchas sorpresas. Mi mayor problema fueron los puntos de control, los cuales entiendo su propósito, pero al cambiar los valores mi objeto salía disparado con un ángulo pequeño, por lo que el salto se veía con falta de naturalidad.

```
arma::frowvec P1 = {initialX, 0, 0};
// Tuve problemas con los puntos de control. No los supe emplear del todo bien.
// Los balones avanzabann pero regresaban. Intenté arreglarlo de muchas
//  formas, pero al final mejor lo dejé así.
arma::frowvec P2 = {(float)(currentBounceMaxX * 0.75), currentBounceMaxY, 0};
arma::frowvec P3 = {(float)(currentBounceMaxX * 0.75), currentBounceMaxY, 0};
arma::frowvec P4 = {currentBounceMaxX, 0, 0};
```

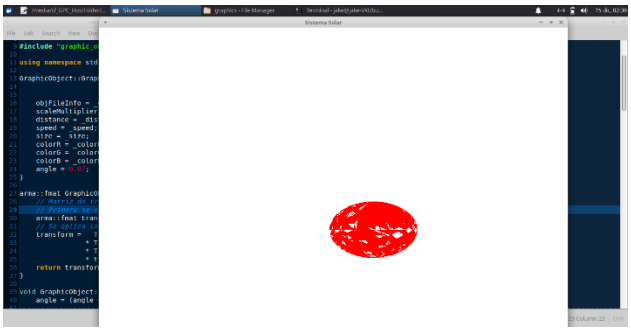
Como se logra ver en la captura de pantalla los puntos de control están definidos, pero no como yo quisiera. Al inicio tenía que el punto de control 2 (P2) fuera ¼ del punto final de la curva actual, pero al momento de correr el programa no daba el resultado que yo deseaba. Esto me obligó a tener que trabajar por tanteo hasta llegar a un resultado que me causara mayor satisfacción.

Aun así, el resultado no me agradó del todo, pero finalmente el programa funciona simulando rebotes, así que mi solución simplemente fue cambiar valores por tanteo hasta llegar a lo que más me pareciera. Es cierto que es poco profesional hacer algo así, pero después del tiempo que invertí revisando dicho problema solo me quedó optar por esa solución.

Muy probablemente eso fue fruto de algún parámetro u operación que haya hecho mal, pero como ya indiqué anteriormente, el programa funciona y eso era lo más importante en este caso.

- **Dibujado de objetos**

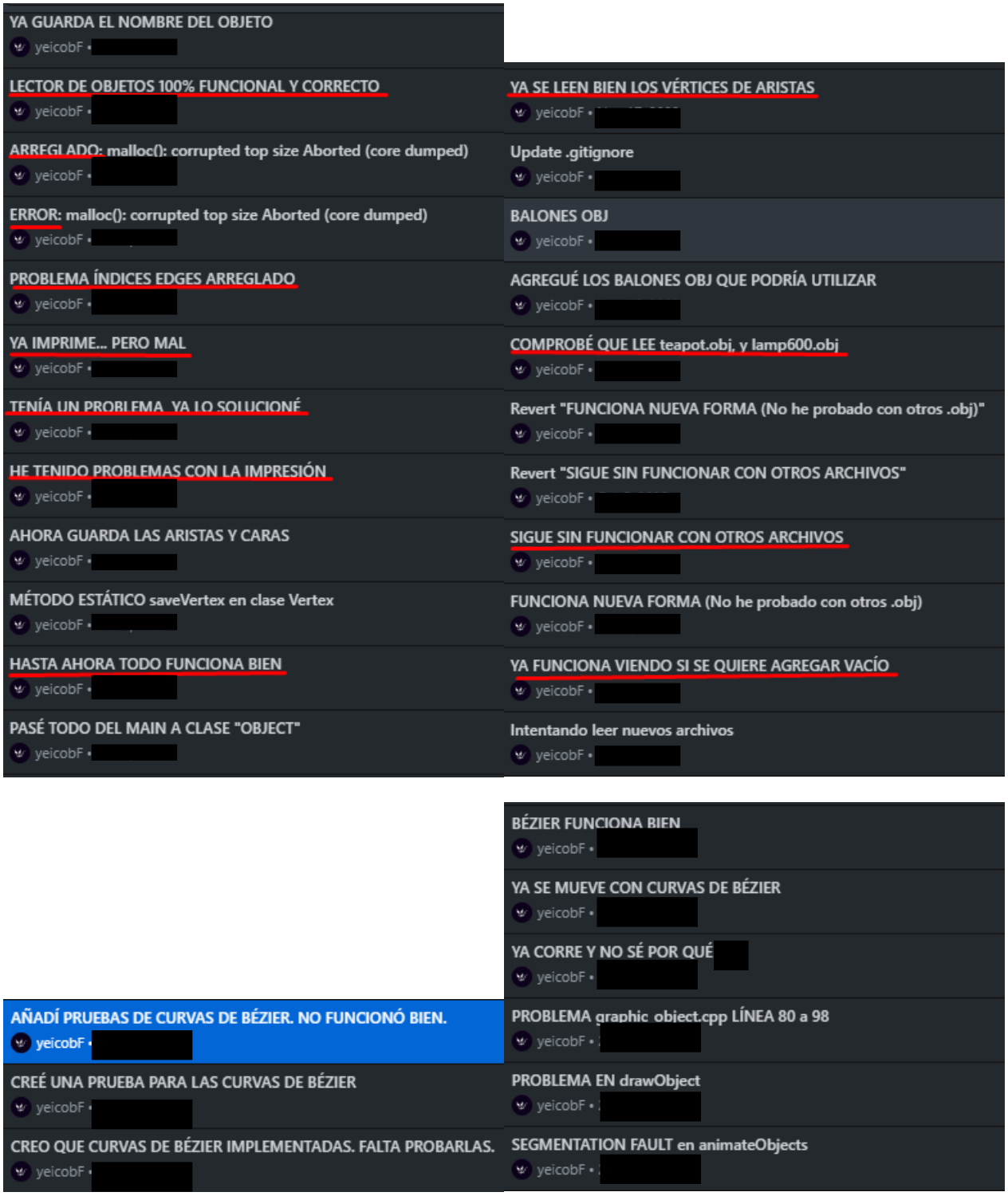
Como último problema que indicaré en esta sección de forma textual está el dibujado de objetos. Este fue complicado en un inicio, ya que el objeto no se dibujaba correctamente. Se dibujaba con huecos y de formas extrañas. Como era de esperar, estuve investigando mi código para encontrar el error, lo cual fue gracioso cuando lo encontré.



```
/* values[actualIndex].substr(0, values[0].find("/") indica que buscará
    el número completo hasta encontrar la diagonal, ya que en las líneas
    de texto que indican el número de vértices a relacionar los aristas,
    estos se delimitan con una diagonal.*/

int indexV1 = stoi(values[actualIndex].substr(0, values[0].find("/"))) - 1, indexV2 = 0;
// Si se llegó al índice máximo hacer último arista con primer vértice.
if(actualIndex == (values.size() - 1)){
    // Como llegamos al último valor, relacionar con el primer vértice.
    /* El índice del segundo vértice = el vértice en el primer índice de
    values, en donde se guardaron los vértices pero restando 1 para tener
    su índice de la lista de vértices.*/
    indexV2 = stoi(values[0].substr(0, values[0].find("/"))) - 1;
    e = Edge(v_list[indexV1], v_list[indexV2]);
}
else{
    /* Si el primer vértice está dentro del límite,
        asignar el segundo con vértice que sigue. */
    indexV2 = stoi(values[actualIndex + 1].substr(0, values[0].find("/"))) - 1;
    e = Edge(v_list[indexV1], v_list[indexV2]);
}
```

Fue de esos fallos que son diminutos, pero te cuestan mucho tiempo y dolor de cabeza. El problema era que, en mi lector de objetos a la hora de leer los vértices de cada cara para formar las aristas, solo leía el primer carácter cuando los vértices eran 3 números, 3 caracteres hasta llegar al delimitador, el cual era una diagonal. Afortunadamente después de cambiar esto el dibujado funcionó correctamente y ya no tuve problemas en ese aspecto. Además, tuve que dibujar con polígonos en lugar de triángulos. Adjunto la parte del código del problema.

Y no solo tuve los problemas que mencioné tuve muchos más, pero la gran mayoría fueron solucionados. Aun así, tengo constancia de muchos de estos fallos gracias al repositorio que creé en Git, y a los cambios que subía constantemente. Adjuntaré capturas de pantalla de algunos de los fallos, aunque también de cuando sucedían cosas buenas.



Como se logra ver en las capturas, me encontré con diversos errores, pero luego los solucionaba y así sucesivamente, repitiendo el mismo proceso una y otra vez. Un bucle del que muchos programadores no pueden escapar.

CONCLUSIÓN

El desarrollo de proyectos de esta escala (que no es grande, sino que requiere de tiempo y aprendizaje constante) requiere de mucho interés de quien lo está desarrollando, además de inversión de tiempo tanto para aprender como para avanzar. Me parecen muy interesantes este tipo de trabajos porque nos van preparando para lo que en algún momento nos vayamos a encontrar, además de que nos ayuda a definir qué es lo que nos gusta y lo que realmente queremos.

Los gráficos por computadora me sorprendieron bastante, y en el lapso en que realicé el proyecto aprendí muchas cosas que nunca me imaginé que aprendería. Esto es muy positivo, ya que me gusta aprender cosas nuevas y ampliar mi conocimiento sobre las que ya conozco. En este caso se representaría como la programación, la cual ya conocía y ya había mantenido contacto, pero nunca había tenido un acercamiento tan cercano a lo que son los gráficos por computadora.

Todo el contenido que aprendí de alguna forma me será útil en el futuro, siga trabajando con este tipo de proyectos o no, ya que me llevo el conocimiento del tema y de lo nuevo que aprendí de la programación. Pero, a pesar de que me parezcan muy interesantes los gráficos y cómo se generan, cómo se utilizan y demás, por ahora no me atrae del todo trabajar en un área del rubro que involucre un manejo tan directo de estos. Me agradan por lo que son, y por su nivel de complejidad, pero por ahora no me siento del todo atraído. Aun así, es de los proyectos que más me han gustado por todo lo que englobó su realización.

En conclusión, me llevé mucho conocimiento nuevo, complementé el que ya tenía, y me agradó realizar este trabajo. Los gráficos a computadora se ven muy bonitos por fuera, ya sea en películas, videojuegos, entre otros, pero al aumentar el nivel de profundidad nos encontramos con elementos y procesos muy complejos que se han logrado gracias al avance tecnológico. Por esta razón al inicio del semestre quería hacer un proyecto de mayor complejidad porque no la conocía, pero ahora que conozco más que hace unos meses me doy cuenta de que no es nada sencillo trabajar con estas herramientas. Y, me doy cuenta de que aquí aplica muy bien el refrán “no juzgues a un libro por su portada”, pues fue lo que hice al inicio, y ahora me doy cuenta de que las cosas eran totalmente distintas. Por esto hay que ser analistas con lo que hagamos, y plantearnos objetivos certeros.

REFERENCIAS

1. James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes, Computer Graphics: Principles and Practice, ADDISON-WESLEY PUBLISHING COMPANY, 1990

2. M. Pérez, “¿Cuáles son las Partes de un Informe/Reporte?”. https://www.lifeder.com/partes-informe/#:%7E:text=Las%20partes%20de%20un%20informe%20o%20reporte%20m%C3%A1s%20destacadas%20son,incluir%20anexos%20y%20p%C3%A1ginas%20preliminares.&text=Los%20hechos%20expuestos%20en%20un,realizado%20previamente%20por%20el%20autor, 2020

3. “Modelos de 3D Gratis”. https://www.turbosquid.com/es/Search/3D-Models/free, (s. f.)

4. D. Chakravorty, “The Most Common 3D File Formats”. https://all3dp.com/3d-file-format-3d-files-3d-printer-3d-cad-vrml-stl-obj/#:%7E:text=We%20simply%20explain%20the%20most,OBJ%2C%20FBX%2C%20COLLADA%20etc , 2020

5. “2. Linear interpolation”. https://en.khanacademy.org/computing/pixar/animate/ball/v/a2-quick, (s. f.)

6. “3. Bezier curves”. https://en.khanacademy.org/computing/pixar/animate/ball/v/animation3, (s. f)

7. M. Winter, "How to change background color of an OpenGL surface?". https://stackoverflow.com/questions/53535984/how-to-change-background-color-of-an-opengl-surface, 2018

8. P., "Hello world: drawing a square in OpenGL". http://www.dgp.toronto.edu/~ah/csc418/fall_2001/tut/square, (s. f.)

9. A. González, "Clase Lista C++ Estándar". http://profesores.elo.utfsm.cl/~agv/elo326/list.pdf, 2001

10. J. W., "Declaring vectors in a C++ header file". https://stackoverflow.com/questions/4230345/declaring-vectors-in-a-c-header-file, 2010

11. saghi, ".cpp and .hpp". https://www.sololearn.com/Discuss/1644964/cpp-and-hpp, 2019

12. Alex, "11.11 — Class code and header files". https://www.learncpp.com/cpp-tutorial/class-code-and-header-files/, 2007

13. "C++ Files". https://www.w3schools.com/cpp/cpp_files.asp, (s. f.)

14. Sayan Mahapatra, "Why "using namespace std" is considered bad practice". https://www.geeksforgeeks.org/using-namespace-std-considered-bad-practice/#:~:text=The%20statement%20using%20namespace%20std%20is%20generally%20considered%20bad%20practice.&text=In%20the%20worst%20case%2C%20the,to%20resolve%20identifier%20name%20conflicts., 2020

15. "C++ String to float/double and vice-versa". https://www.programiz.com/cpp-programming/string-float-conversion#:~:text=help%20of%20examples.-,C%2B%2B%20string%20to%20float%20and%20double%20Conversion,convert%20string%20to%20long%20double%20., (s. f.)

16. "Initialize a vector in C++ (5 different ways)". https://www.geeksforgeeks.org/initialize-a-vector-in-cpp-different-ways/, 2020

17. "Passing vector to a function in C++". https://www.geeksforgeeks.org/passing-vector-function-cpp/, 2017

18. pp_pankaj, "Difference Between Vector and List". https://www.geeksforgeeks.org/difference-between-vector-and-list/, 2020

19. "C++ sin()". https://www.programiz.com/cpp-programming/library-function/cmath/sin, (s. f.)

20. "Naming conventions". https://manual.gromacs.org/documentation/5.1-current/dev-manual/naming.html, (s. f.)

21. "<iostream>". http://www.cplusplus.com/reference/iostream/, (s. f)

22. Nitin Sharma, "Difference between Header file and Library". https://www.tutorialspoint.com/difference-between-header-file-and-library, 2020

23. "<string>". http://www.cplusplus.com/reference/string/, (s. f)

24. "<vector>". http://www.cplusplus.com/reference/vector/, (s. f)

25. "<fstream>". http://www.cplusplus.com/reference/fstream/?kw=%3Cfstream%3E, (s. f.)

26. "Armadillo". http://arma.sourceforge.net/, (s. f.)

27. “<cmath> (math.h)”. http://cplusplus.com/reference/cmath/?kw=cmath, (s. f.)

28. "OpenGL". http://www.cs.tufts.edu/research/graphics/resources/OpenGL/OpenGL.htm, (s. f.)

29. "OpenGL Overview". https://www.opengl.org/about/, (s. f.)

30. Marcus Geelnard, Camila Löwy, "glfw3.h". https://www.glfw.org/docs/3.3.2/glfw3_8h_source.html, 2020

31. "glfw3.h File Reference". https://www.glfw.org/docs/3.3.2/glfw3_8h.html, 2020

32. "<random>". https://www.cplusplus.com/reference/random/?kw=%3Crandom%3E, (s. f.)

33. "<cstdlib> (stdlib.h)". https://www.cplusplus.com/reference/cstdlib/?kw=%3Ccstdlib%3E, (s. f.)

34. user54905, "What is the equivalent of an “exe file”?". https://askubuntu.com/questions/156392/what-is-the-equivalent-of-an-exe-file, 2012

35. "Transformaciones Geométricas". http://prepa8.unam.mx/academia/colegios/matematicas/paginacolmate/applets/matematicas_VI_4/Applets_Geogebra/transfgeom.html, (s. f.)

ANEXO – MANUAL DE USUARIO

REQUISITOS PARA QUE EL PROGRAMA FUNCIONE

Para ejecutar el programa se necesita cumplir con varios requisitos, los cuales serán totalmente necesarios. De otra forma el programa no va a funcionar. A continuación, se encontrarán listados.

1. **Sistema operativo distribución de GNU/Linux**. En mi caso utilicé Xubuntu, aunque probablemente funcione correctamente en muchas más distribuciones.

2. **Armadillo:** Biblioteca para álgebra linear y computación científica.

    - INSTALACIÓN DESDE LA TERMINAL: **sudo apt-get install libarmadillo-dev**

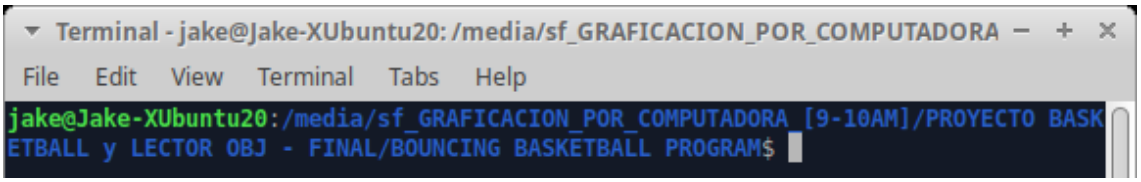
3. **OpenGL:** Entorno principal para desarrollar aplicaciones de gráficos 2D y 3D portátiles interactivos

    - INSTALACIÓN DESDE LA TERMINAL:
        1. **sudo apt-get install libglfw3-dev**
        2. **sudo apt-get install libglew-dev**

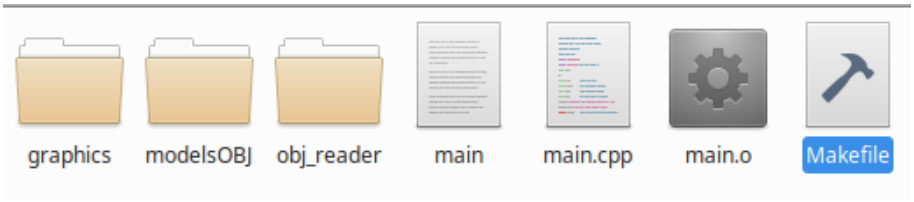
Habiendo cumplido los requisitos ya podría ser compilado y ejecutado el programa.

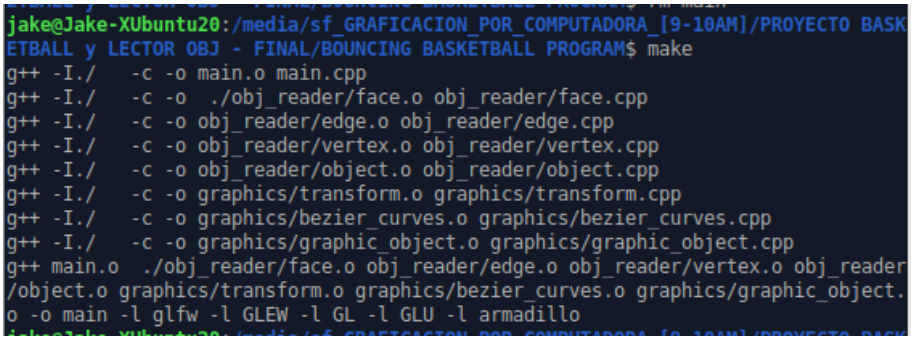
INSTRUCCIONES PARA COMPILAR Y EJECUTAR EL PROGRAMA

1. Desde la terminal, entrar al directorio en donde se encuentra el programa. Es decir, en donde se encuentra el archivo "***main.cpp***" y las carpetas con los archivos de código fuente.

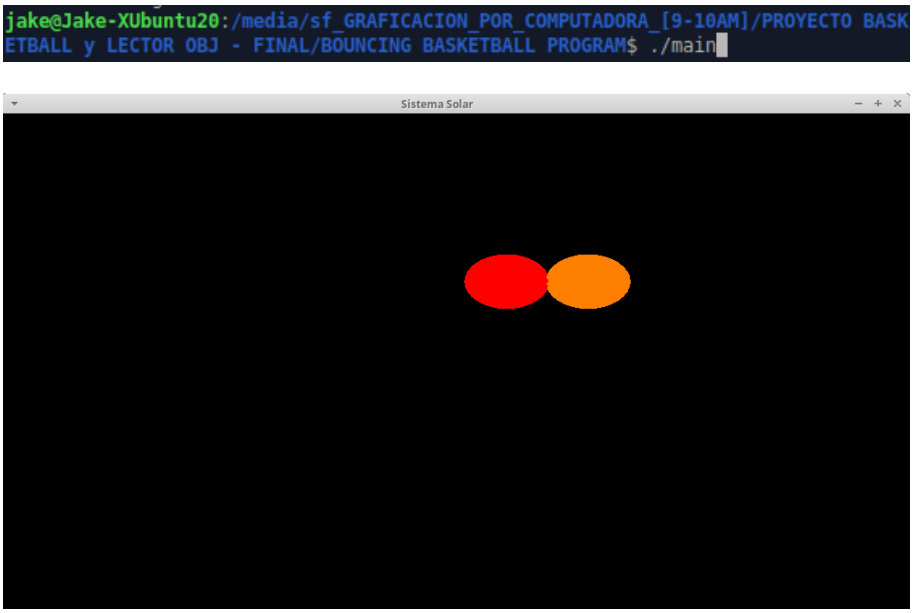


2. Ejecutar el comando "***make***". Esto hará que se ejecute el archivo "***Makefile***" y se compile el programa en su totalidad, permitiendo que el programa se pueda ejecutar.





3. Ahora que ya se ha compilado el programa, ejecutar el comando "***./main***". Esto hará que el programa comience a ejecutarse. Se abrirá una ventana con el programa en ejecución.



Problemas al compilar o errores de segmentación

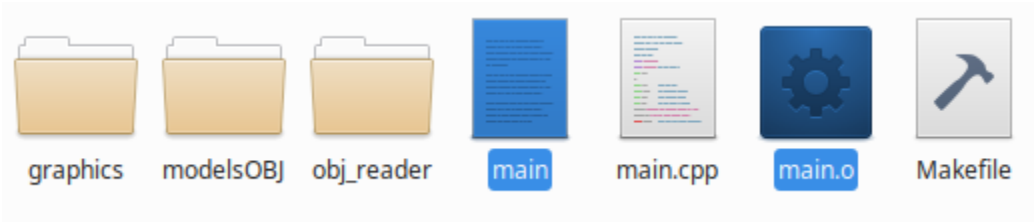
Si tienes problemas al compilar o ejecutar, tales como "***Segmentation fault (core dump)***" o algún otro, puede que se haya dañado algo que impida que el programa funcione aunque este esté correcto. Lo que yo recomiendo porque me ha funcionado, es borrar todos los archivos con la extensión "***.o***" que haya en los directorios, y el archivo con el nombre de "***main***" (no el de "***main.cpp***"). Esto se puede lograr con el siguiente comando en la terminal:

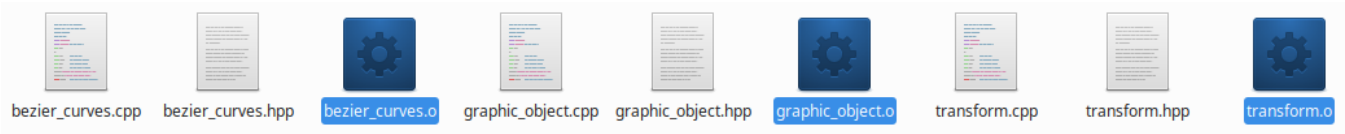
- **rm *.o */*.o main**

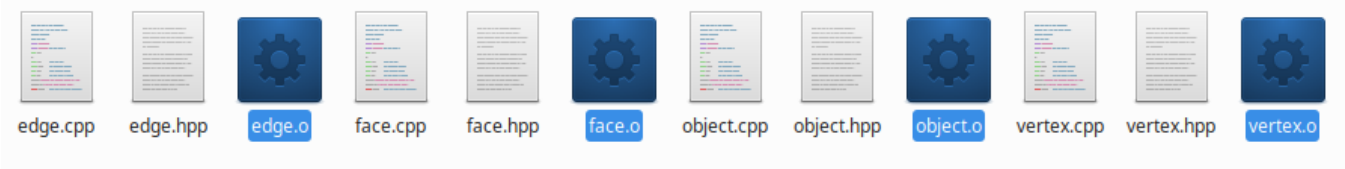
Lo que hará será borrar todos los archivos con dicha extensión (incluyendo los de las carpetas), y el "main".

Después de esto se tendrán que repetir los pasos de la sección anterior. Si no se soluciona, lo más seguro es que se haya modificado el código fuente y se haya cometido algún error. Eso ya depende de quien hizo dichos cambios.

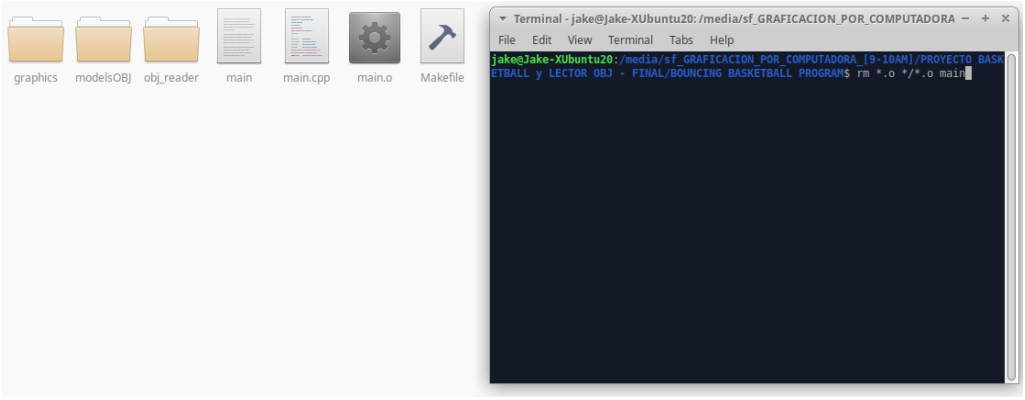
Antes de ejecutar el comando

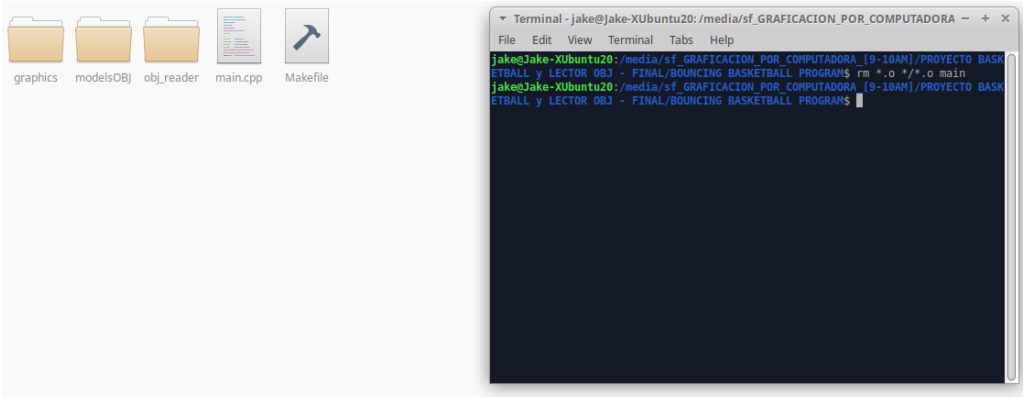


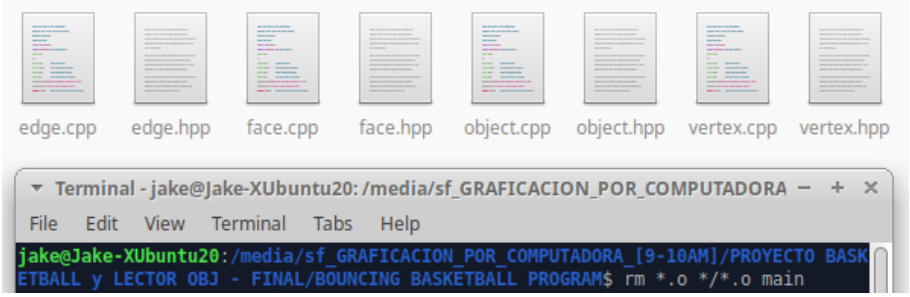




Después de ejecutar el comando







¿CÓMO INTERACTUAR CON EL PROGRAMA?

Depende de qué quieras hacer.

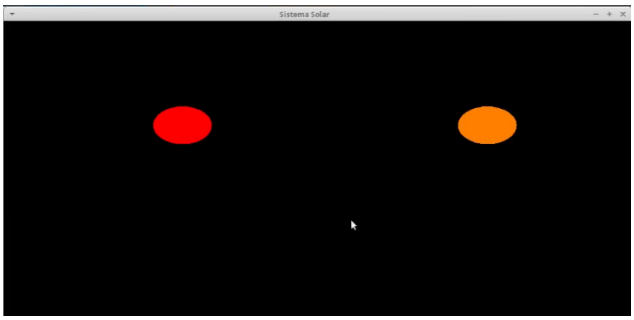
- Si quieres cambiar los parámetros con los que se crean los balones, deberás modificar directamente los constructores de las instancias de la clase ***GraphicObject*** y poner los que más te parezcan. Esto se encuentra en el archivo "***main.cpp***", que es en donde comienza la ejecución del programa. No es necesario modificar nada más. Cuando hayas finalizado deberás volver a compilar el programa. Esto lo lograrás siguiendo los pasos de la sección anterior. Para un mejor entendimiento adjuntaré una captura de pantalla de lo que se deberá modificar, que es lo que está seleccionado.

```
/*
GraphicObject(Object _objFileInfo, float _scaleMultiplier,
                        float _distance, float _speed, float _size,
                        float _colorR, float _colorG, float _colorB)
*/
/* Se pasa como parámetro el objeto con la información del archiv OBJ en
    memoria. */
GraphicObject basketball = GraphicObject(objBasketBall, 0.5f,
                                         -0.9f, 2.0f, 0.5f,
                                         1.0f, 0.0f, 0.0f);
// Se agrega el objeto a la lista de los que se dibujarán.
object_list.push_back(basketball);

GraphicObject basketball2 = GraphicObject(objBasketBall, 0.5f,
                                         0.9f, 2.0f, 0.5f,
                                         1.0f, 0.5f, 0.0f);
```

- Si quieres interactuar con el programa en ejecución encontrarás cómo hacerlo en la siguiente sección.

INTERACTUAR CON EL PROGRAMA EN EJECUCIÓN

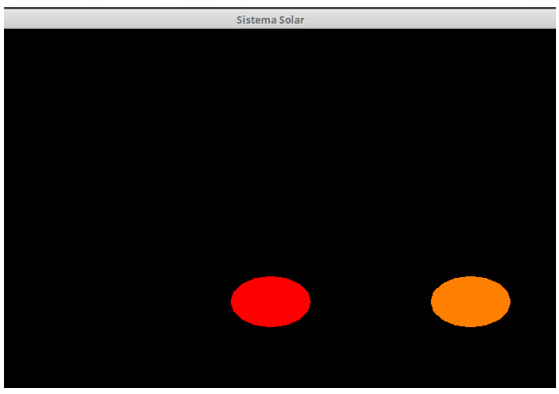
Una vez que se hayan seguido todos los pasos, incluso habiendo ejecutado el programa, te encontrarás con una ventana en donde se verán dos objetos esféricos, específicamente dos balones (aunque no lo parezca). Estos dos primeros objetos son constantes, es decir que siempre se crearán al iniciar el programa. Esto es con el objetivo de que se vea la funcionalidad original, y después que el usuario interactúe con el programa. Cabe señalar que se puede interactuar desde que se comienza a ejecutar.



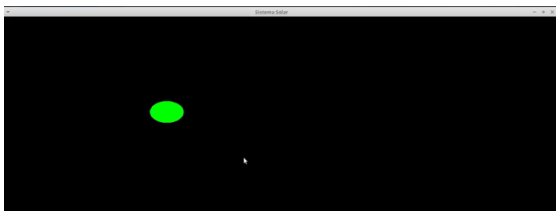
Teclas para interactuar con el programa

Inmediatamente podrás interactuar con el programa utilizando las teclas del teclado (valga la redundancia), que son las siguientes:

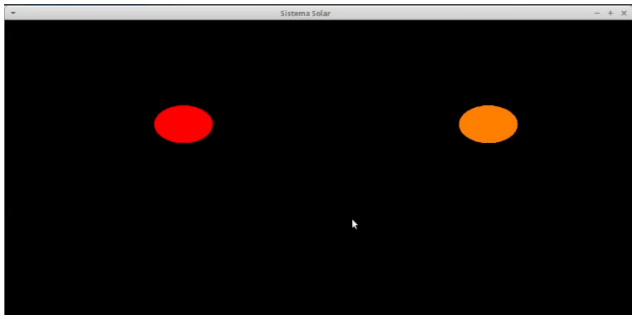
- **"D":** Cambia la cámara para que puedas observar lo qué sucede desde arriba.



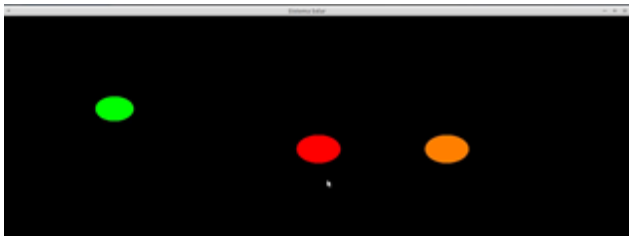
- **"I":** Invierte la perspectiva desde donde estás observando la animación. Esto solo funciona si se encuentra en la perspectiva inicial.



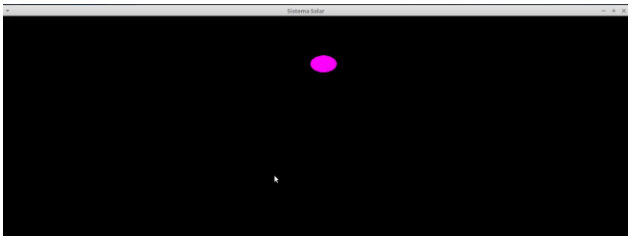
- **“O”:** Vuelve a la perspectiva original, la perspectiva en donde inició la animación.



- **“ENTER”:** Crea un nuevo balón que será generado desde la parte izquierda de la perspectiva original. Este será de un color aleatorio (se pueden repetir), y de un tamaño aleatorio.



- **“BACKSPACE” (Tecla de retroceso):** Crea un nuevo balón que será generado desde la parte derecha de la perspectiva original. Este será de un color aleatorio (se pueden repetir), y de un tamaño aleatorio.



- **“ESCAPE” (Esc, tecla de escape):** Se cerrará la ventana terminando el programa como consecuencia.

Y esto es todo lo que se necesita saber para compilar, ejecutar, e interactuar con el programa.