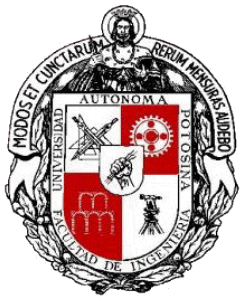
Universidad Autónoma De San Luis Potosí

Facultad De Ingeniería



Ciencias De La Computación

Ingeniería En Computación

Graficación por Computadora A

Reporte Final Proyecto

Profesor: Omar Rodríguez González

Elaborado por:

Cardona Becerra Luis Eduardo

Hernández Cuevas Jeneffer Stephane

Introducción

En este reporte se presentara una explicación del proyecto realizado en la materia de Graficación de Computadoras A, basado en los temas vistos en clase a lo largo del semestre.

Es importante mencionar que el proyecto fue desarrollado con el propósito de modelar el movimiento a través de una curva y representarlo de manera gráfica para plasmar los conocimientos que adquirimos a lo largo del curso.

El proyecto abarca principalmente los temas de movimiento curvo a través de una trayectoria, en este caso para representar dicha trayectoria se implementó el movimiento en un objeto que simula un balón de futbol soccer.

La simulación del proyecto se visualizará de manera gráfica en la pantalla de escritorio representaciones en 3D convertidas a 2D para poder ser captado por nosotros, para esto se hizo el uso de proyecciones, en este caso la proyección en paralelo es la implementada. El programa está basado principalmente en la lectura de archivos obj, al extraer el archivo dentro del programa se hace el uso de la clase Bezier que es la encargada del modelado de las curvas y las clases de Escalación, Rotación y Traslación para poder lograr el movimiento a través de ellas. Todo lo ya mencionado se hizo visible gracias a OpenGL para poder hacer el modelado en un gráfico 3D utilizando las coordenadas x, y, z para proyectar.

A lo largo del semestre se trataron varios temas, el objetivo del proyecto es desarrollar los temas principales haciendo uso de distintas bibliotecas y herramientas ya existentes. Se trataron los temas más básicos para poder lograr la compresión de cómo funciona los modelados más complejos utilizados en plataformas importantes con un mayor desarrollo, de esta manera entendemos las bases y de donde viene todo lo que hay actualmente en televisión y videojuegos.

Este proyecto también incluye el desarrollo de nuestro proyecto dando a conocer el uso de las distintas clases implementadas y de cómo agregamos OpenGL para poder graficar los objetos que requeríamos, se explica los errores y problemas que tuvimos y como los resolvimos en caso de que se retome este proyecto o se requiera basar en él y puedan surgir este tipo de percances.

El lenguaje utilizado en el desarrollo del proyecto fue C++.

Desarrollo

¿Qué es y para que usamos OpenGL?

Es un conjunto de librerías que son utilizadas a través de lenguajes de programación para conseguir un interfaz software entre las aplicaciones y el hardware gráfico. La librería está formada por unas 250 instrucciones diferentes que se utilizan para especificar los objetos y las operaciones necesarias para desarrollar aplicaciones interactivas tridimensionales. Los modelos se deben construir partiendo de un pequeño conjunto de "primitivas geométricas" como puntos, líneas y polígonos.

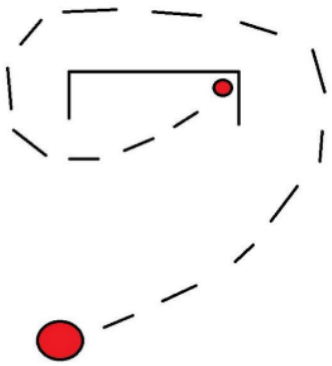
OpenGL, permite:

- Construir formas geométricas a partir de primitivas.
- Ubicar los objetos en el espacio tridimensional y seleccionar el punto de vista de la escena.
- Aplicar el color a los objetos, ya sea mediante una asignación explícita de la aplicación, a partir de las condiciones de iluminación o mediante la utilización de texturas.
- Convertir la descripción matemática de los objetos y la información sobre el color en pixeles de la pantalla, proceso que se llama rasterización.

¿Por qué desarrollar este proyecto?

De acuerdo a los temas que íbamos a tratar en el curso decidimos optar por el movimiento de un balón inspirado en los tiros libres del futbol, en este tipo de tiros es importante saber que se necesita lograr cierta trayectoria y tomar varios factores en cuenta de acuerdo al tiro.

Hemos optado por una trayectoria en donde el balón inicie frente a una portería y se mueva alrededor de ella elevándose para finalmente entrar.



Para esta trayectoria hacemos el uso de 3 curvas de Bezier unidas para lograr una simulación seguida teniendo un punto inicial y uno final y dos intermedios haciendo uso de las coordenadas x, y, z. Se hizo el uso de las siguientes curvas:

No. Curva	X	Y	Z
1	0.0	0.0	0.8
1	0.85	0.1	0.2
1	0.98	0.15	-0.5
1	0.4	0.38	-0.75
2	0.4	0.38	-0.75
2	-0.1	0.5	-0.9
2	-0.6	0.5	-0.9
2	-0.6	0.4	0.1
3	-0.6	0.4	0.1
3	-0.5	0.3	0.95
3	0.06	0.3	0.4
3	0.3	0.35	0.1

Explicación de cómo está estructurado el proyecto:

El proyecto usa el paradigma de la programación orientada en objetos, ya que es una forma de programar mas parecida a la vida real en la cual puedes clasificar mejor los objetos y todo lo que lo conforma, por esta razón la preferencia de dicho paradigma, pero sabemos que hay diferentes lenguajes de programación con este modo de programación, en cuanto a esto nos dio la tarea el profesor de que fuera en el lenguaje de programación C++ ya que este nos iba a facilitar mas mediante diferentes bibliotecas que utilizaríamos, de aquí viene el siguiente punto a considerar lo cual son las librerías, cabe señalar que se utilizan las más comunes para el propio uso de cualquier tipo de programa como lo son:

1. **iostream**
2. **string.h**
3. **vector**

Estas son las más comunes, la iostream para la entrada y salida, la string para uso de cadenas, la vector para usar listas de objetos o cualquier tipo de dato. Pero unas que juegan un papel más importante son:

1. **GLFW/glfw3.h**
2. **GL/glu.h**
3. **armadillo**

Estas librerías desarrollan tareas más importantes, en el caso GLFW/glfw3.h y GL/glu.h conforman la utilización de gráficos en base a OpenGL la cual se explicará más adelante en este documento, de ahí prosigue la que se considera que nos ayudó a optimizar muchas acciones elementales la cual es la librería armadillo, es una librería de algebra lineal la cual en su mayoría nos ayudó a hacer operaciones con matrices o con vectores, mediante diferentes métodos que están implementada en ella, si no fuera por ella tendríamos que estructurar formas de desarrollar las operaciones paso por paso, sin lugar a dudas una librería indispensable. Según las clases que íbamos implementando se necesitaban agregar sus correspondientes cabeceras.

Explicación de todas las clases/funciones

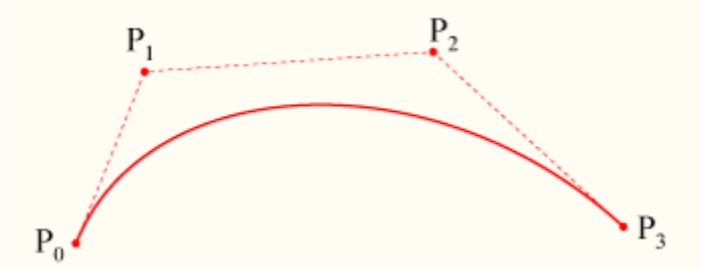
El proyecto esta estructurado por varias clases, todas indispensables para una tarea en específico, enseguida se muestra las diferentes clases que se implementaron para este proyecto.

- Bezier
- Escalacion
- Face
- Obj
- Traslacion
- Vertex
- VertexC

En este apartado se explica a detalle el funcionamiento de cada clase.

Bézier.

Es la clase que se encarga de calcular los diferentes puntos que puede tomar una curva según cuatro puntos que la conforman, los cuales son: el punto inicial, el punto final y dos puntos intermedios que son para dar la curvatura deseada.



En cuanto a cómo fue programada se utilizaron los siguientes atributos:

- dt
- MHB
- Pb1, Pb2, Pb3, Pb4
- vcurveB

En los cuales **dt** es un parámetro que nos sirve para hacer la curva compuesta por puntos, estos puntos varían en su cantidad, si dt es un valor muy pequeño dicha curva va contener muchos puntos, por otra parte si es un valor muy grande los puntos de dicha curva serán pocos, otra cosa a tomar en cuenta de dt es que solo puede variar entre 0 y 1.

Después tenemos **MHB** este parámetro el cual contiene datos que conforman una matriz que fue implementada por el desarrollador de este método, la cual es necesaria para las operaciones. De allí siguen los puntos **Pb1, Pb2, Pb3, Pb4,** los cuales son puntos de control de la curva como se mencionó anteriormente.

Y por último tenemos **vcurveB**, es un vector que contiene todos los puntos de la curva.

Esos fueron los atributos de esta clase, ahora se enlistan los métodos de esta clase:

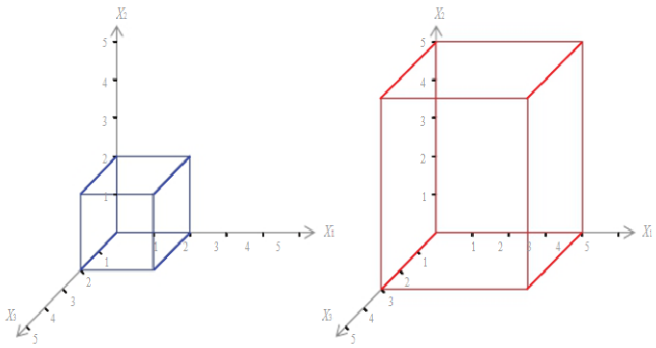
- Bezier()
- Bezier(Pb1, Pb2, Pb3, Pb4)
- calc_curve()
- printGrap()

El método **Bezier()** es el constructor por defecto el cual nos crea el IDE, este método no contiene nada, de allí tenemos una sobrecarga de ese mismo constructor el cual solo define la matriz de Bézier y establece los puntos de la curva también se manda llamar el método de **calc_curve()** , el cual calcula la los puntos de la curva y además los va a agregando al vector que es una lista de los puntos de esa curva.

Por último, tenemos printGrap **(),** llamado así porque pinta gráficamente un objeto, este método recorre la lista de caras y de vértices.

Escalación

Esta clase tiene la tarea de hacer cualquier tipo de modelo a una escala mas pequeña o mas grande, en nuestro caso del proyecto la utilizamos para que nuestra esfera o pelota tuviera un tamaño mas pequeño del que se venía originalmente.



Esta clase contiene los siguientes atributos

- Vertex v1
- Matriz ME
- colVec VE
- colVec VR

El atributo **v1** es un objeto de la clase vértice que contiene sus correspondientes coordenadas, **ME** la matriz de escalación la cual se utiliza para calcular la operación entre el vértice y la misma matriz para obtener los nuevos vértices. De ahí obtenemos los datos colVec, que son usados ya que armadillo los necesita para hacer las operaciones de matrices, **VE** es un vértice y **VR** es el vértice resultante.

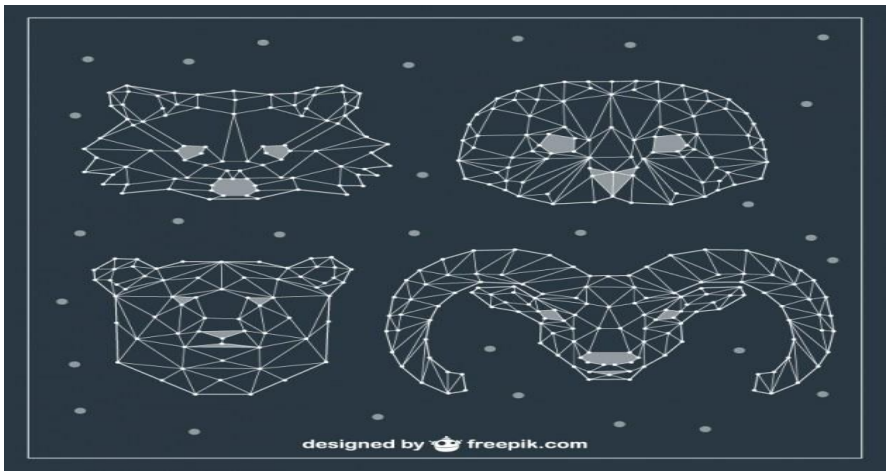
A continuación, los métodos de esta clase:

- Escalacion(esc, vertice)
- Multi(matriz, colvec)

En esta clase solo se utiliza solo utiliza dos métodos, **Escalacion(esc, vertice)** que en su constructor contiene esc que es el numero a escalar el objeto y el vértice que se va a escalar, en este método se define la matriz de escalación en base a la escala de esc, y se manda llamar el método de Multi el cual solo ejecuta la multiplicación de la matriz y el tipo de dato colVec que contiene el vértice.

Face

Esta clase es una clase muy importante para la formación de caras que conforman nuestros polígonos en nuestro modelo obj, a grandes rasgos esta clase se encarga de definir objetos con características propias de un triángulo en nuestro caso.



Los atributos de esta clase son:

- indVer1, indVer2, indVer3.
- NorA, NorB, NorC, NorAN, NorBN, NorCN, D.
- magnitudN, magnitudL.
- Vector veNor;
- Vector vecL, vecLN;

Comenzando con **indVer1, indVer2, indVer3,** son los índices de los vértices que conforman esta cara son tres porque son los necesarios para hacer una cara triangular. Después tenemos los valores que conforman un vector normal de esa cara los cuales son los valores de **A, B, C,** también tenemos esos mismos valores, pero con la normal normalizada que son los **AN, BN, CN, y D,** que es un valor extra para usarlo con la ecuación del plano, pero no tiene relevancia en este proyecto. Por otra parte esta **vecN** que este es un tipo de vector de armadillo, pero definido con **A, B, C.** Por último, tenemos un vector especial que se forma con la ayuda de un foco y un vértice de la cara el cual es **vecL**, y ese mismo vector, pero normalizado que es **vecLN.**

También en esta clase están los métodos:

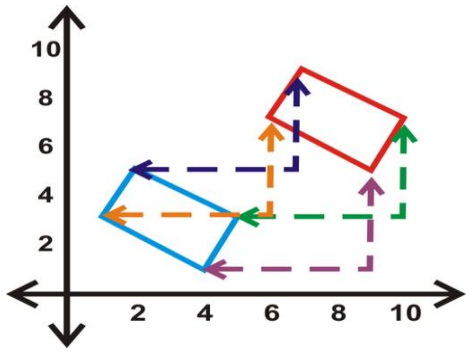
- Face()
- Face(indVer1, indVer2, indVer3)
- SetFace()
- SetNormales()
- getFace1()
- getFace2()
- getFace3()
- getAN()

- getBN()
- getCN()
- getveNor()

Estos métodos tienen el constructor por defecto que solo lo utilizamos para inicializar los índices de los vértices, hay una sobrecarga de ese método el cual ese si define los índices de los vértices que le corresponde a su respectiva cara. Después de eso poseemos diferentes getters y setters que nos ayudan a obtener o definir atributos de objetos.

Traslación

Este método tiene una destacada acción en todo el programa, el cual es el efecto del movimiento. La traslación como su palabra lo dice es una traslación o movimiento del objeto que en nuestro caso son todas las caras que lo conforman.



Esta clase posee los siguientes atributos

- Vertice v1
- Mat MT
- colVec VT
- colVec VR

v1 es un objeto del tipo vértice que contiene como atributos las coordenadas, enseguida que mencionemos esa clase se verá más a detalle, pero además posee una matriz **MT** que es la adecuada para hacer la operación de traslación y tenemos dos tipos de datos colVec vectores necesarios para llevar a cabo la operación de la matriz. Los métodos que conforman esta clase son los siguientes

- Traslacion(trasX, trasY, trasZ)
- Muti(MT, VR)

Estos son dos métodos para obtener una nueva traslación, **Traslación (trasX, trasY, trasZ)** es el método constructor lo pusimos con tres parámetros que son la traslación en X, en Y y en Z, aquí

definimos nuestra MT según esos tres últimos parámetros, y mandamos llamara a **Multi()** y esta nos hace obtener los nuevos resultado de en VR que son los valores a los cual multiplicar nuestros vértices para obtener ese nuevo lugar para el modelado.

Vertex

Es una clase muy importante, ya que cada objeto creado a partir de esta clase posee todas las coordenadas de dicho objeto, en si es como el esqueleto de nuestros modelos. Los siguientes atributos la caracterizan:

- CX
- CY
- CZ

Son en cuestión solo las coordenada en X, Y, y Z de dicho objeto. Acerca de sus métodos son los siguientes:

- Vertex()
- Vertex(X, Y, Z)
- setVertex(x,y,z)
- getVertexX()
- getVertexY()
- getVertexZ()

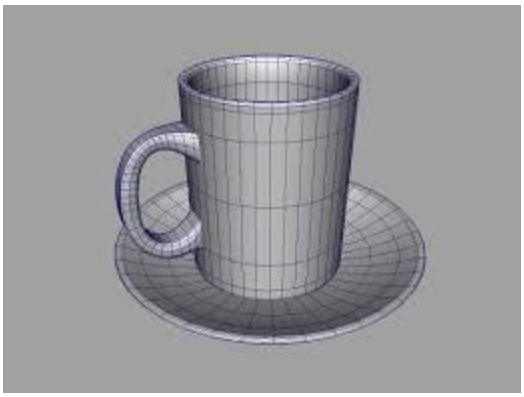
Estos métodos en el caso del constructor por defecto no hacen más que inicializar los valores de las coordenadas, la sobrecarga del constructo es la que las vuelve a establecer, pero ahora con sus verdaderos valores, enseguida de eso tenemos algunos getters y setters para la definir y obtener atributos.

VertexC

Este método hace lo mismo que el anterior la única diferencia es que lo implementamos ya que el profesor de la materia nos lo proporciono junto con la clase Bezier para añadirla en nuestro trabajos o proyectos.

Obj

Esta clase la podemos considerar la mas importante ya que esta es como si fuera nuestro lector de archivos Obj, ya que gracias a ella podemos tener una lista de vertices y lista de caras las cuales conforma nuestro modelo Obj.



Sus atributos son los siguientes:

- lista Caras
- lista Verti
- nomArch

Como se mencionó anteriormente posee una lista de caras y otra lista de vértices que son los principales atributos de esta, también tiene un identificador de tipo cadena para saber cual es el nombre del archivo Obj. Ahora mostraremos los métodos de esta clase:

- Obj(nomArch)
- leeArchivo(tipo)
- llenaListaVer(números)
- llenaListaFac(numerosF, listCar, vector foco)
- calcNormales

Son 5 métodos importantes comenzando con el constructor por defecto el cual solo nos hace que el nombre con el que lo mande llamar se le asigne a nuestro atributo nomArch, seguimos con el método **leeArchivo()**, este método se encarga de crear una cadena obteniendo todos los valores de todas las líneas del archivo obj siempre y cuando cumplan la condición de que empiezan con el carácter el cual le pasamos como parámetro cuando se llama a este método, sigue el **llenaListaVer()** con la cadena que se obtiene del método anterior se leen de tal manera de identificar cuando corresponde a la coordenada en X, en Y y en Z, y establecerlo en una lista de objetos de la clase Vertex y así hacer una lista de vértices que contenga sus correspondientes coordenadas. Después sigue hacer los mismo, pero con las caras, leer las caras del archivo Obj y guardar los índices de los vértices que forman esas caras, de allí obtener una lista de caras y proseguir con el método **calcNormal ()** para que nos arroje las normales de cada cara, junto con esto obtener el vector L el cual es el vector que va del foco al vértice. Con estos métodos resumidos conformamos nuestro modelo.

Problemas/Experimento

Los problemas fueron encontrados conforme íbamos haciendo nuevas incorporaciones en el proyecto, una de los primeros impedimento que tuvimos fue acerca del lector de archivos obj, el cual era de que teníamos que recorrer línea por línea detectando los dígitos, y no tomar en cuenta los espacios entre cada conjunto de dígitos, para esto tuve primero la opción de identificar los números con un método llamado isdigit, el cual me dice si está en un carácter de número, pero por alguna extraña razón no funcionaba como deseaba, entonces lo mejor que pude hacer para esto fue que guardara primero todos los números en una cadena, pero esa cadena tenia una particularidad la cual era que solo contenía un carácter de espacio entre cada número, esto lo hacía más genérico la búsqueda, me di cuenta que las cadenas las podía utilizar con iteradores para saber si en ese momento de un tal índice se encontraba un carácter en específico, y esto ayudo a que funcionara el lector.
Después de allí tuvimos que dibujar los polígonos con las funciones de opengl, en cuanto a eso tuvimos un detalle que se nos pasó por alto , el cual fue el de que al momento de importar un modelo obj de referencia desde blender teníamos que importarlo con la opción de caras triangulares, ya se nos pasó y cuando dibujábamos las caras de una esfera dicha esfera se miraba con huecos, lo cual era porque no eran caras triangulares y el lector no las leía bien.
Una vez hecho todo esto tocaba usar la cámara para verlo a través de diferentes punto de vista. Aquí no había problemas hasta que nos tocó implementarlo en nuestros editores, desde el principio utilizamos la cámara en modo perspectivo, pero no pudimos poder abarcar la región del dibujo con esta cámara, así que implementamos mejor la paralela, ya que mediante esta no necesitaba una cierta región para solo ser vista, toda la región se podía notar.

Otro percance que tuvimos fue al momento de querer saber las teclas que presionábamos para poder cambiar la vista y es que al principio el ejecutable nos detectaba algo que no era pero lo solucionamos haciendo uso de las variables globales y quitando ciertas partes que nos imprimían texto para no obstruir en la lectura de las teclas.
Por último, tuvimos un problema el cual era cuando tuvimos que implementar la iluminación, en el momento de sacar las normales , en nuestro caso el profe nos paso un modelo el cual nos iba a dar de referencia para ver si estaba bien nuestro cálculos, entonces tomamos fotos a sus resultado y afortunadamente está bien nuestro programa, pero cuando lo añadimos al proyecto, por alguna razón tomaba mal las normales y daba resultados incorrectos, era algo extraño ya que se basaban en el mismo código, pensé que si probaba con otro modelo obj y comparándolo con los dos programas debería salir igual, y fue así funcionaba, solo que con el modelo del profe no se cumplía.

Conclusiones/Comentarios Finales

En cuanto a mi opinión Luis Eduardo Cardona Becerra, pienso que me dejo una gran enseñanza el hecho de desarrollar este proyecto, ya que por el lado de la programación aprendí formas alternativas de resolver problemas, pero uno de los aprendizajes mas satisfactorios es la hecho de comprender los fundamentos básicos de los gráficos por computadora, de pequeño yo veía los gráficos de computadora aplicados en los videojuegos pero nunca creía que detrás de ellos conlleva un gran uso de las matemáticas, que van desde las más sencillas hasta aquellas en las que tienen que intervenir matrices y vectores. Yo cuando conocí esto de la algebra lineal en el departamento no le vi una aplicación tan importante hasta que llegué a esta materia de gráficos por computadora, hasta hace poco comprendía un poco de cómo se hacían los juegos en 2d, a través de pixeles, pero también me puse la cuestión, pero uno en 3d como se hacía, y lo descubrí que era gracias a los polígonos. Otros detalles importantes para destacar que me pareció fascinante fueron los que en realidad muchas de lo que vemos en un monitor en realidad no es lo que en nuestro cerebro percibe, por ejemplo cuando vemos que un objeto está muy cerca del monitor lo juzgamos que está cerca de nosotros, pero en realidad no lo es ya que solo lo estamos agrandando mas el objeto, otra cosa que llamo mi atención con respecto a la cámara es de que cuando vemos un objetos solo dibujamos las caras visibles por nuestro ojo, por ejemplo vemos a una persona de frente, cuando la dibujamos no dibujamos su parte trasera, otra y al igual increíble es la iluminación y que esta la podemos interpretar de que hay un foco arriba de un objeto y que por eso se ve con mas claridad un objeto, y esto no pasa en realidad en los gráficos por computadora lo que en realidad pasa es de que dependiendo un punto en el espacio que en nuestro caso representa el foco solo a partir de su distancia a la superficie es la claridad del color del objeto que vamos a pintarle. Sin lugar a duda los gráficos por computadora es una ciencia de la computación increíble el profe nos mencionó que solo esto es lo básico y que lo demás conlleva un grado más grande de complejidad sin embargo su aplicación hace ver la magia de la computación.

Mis conclusiones como Jeneffer Stephane Hernández Cuevas son que al inicio del curso no sabía exactamente de qué se trataba la materia, el profesor nos dio una breve explicación de que se iba a tratar y hasta se nos explicó de manera breve como Pixar hacia modelados gráficos muy interesantes en sus películas animadas y que para llegar a eso se necesitaba una base y saber cómo surge todo.

Gracias a las herramientas que usamos y los temas que tratamos logramos desarrollar un proyecto en el que entendimos el cómo se mueven los objetos de manera gráfica, aunque parece fácil y el resultado final pareciera que no conlleva dificultad, es todo lo contrario ya que para lograr un movimiento tan simple en la vida real, gráficamente se llevan a cabo estudios de cómo

va a ser el comportamiento de los objetos después que sucede cierta acción, para esto es importante usar la física y hacer uso de las matrices que son las que nos permiten hacer la representación de los movimientos.

Es muy interesante conocer la estructura del movimiento y de cómo están definidos los objetos, ya que en esta ocasión el lector obj nos permitía conocer sus vertices, aristas y caras, y si queríamos moverlos se tenía que mover cada cara de una por una, de esta manera ya empezaba a ser compleja la idea ya que teníamos que realizar operaciones con cada cara y esto va a ser así cada que implementemos un algoritmo nuevo al programa, tal es el caso de la iluminación que también se debe de aplicar a cada superficie que obtengamos y volver a graficar ya con los elementos.

Es por esto que el hecho de pensar en que películas y videos animados que hacen el uso de tecnología se vuelve todo un mundo muy complejo porque prácticamente estas simulando la realidad y una vez que sabes de donde viene todo es difícil dejar de pensar en lo que hay detrás para que el espectador crea que lo que ve es real.

En general la graficación por computadora es tan interesante como compleja y en lo personal llegue a batallar en muchas ocasiones por la complejidad de operar cada pequeña parte a la que quería cambiar su movimiento o lo que se requería, más sin embargo aprendí mucho de esta materia y pude conocer algo más en cuanto a la programación y darme cuenta de que me gustó mucho esta parte de los gráficos.

Referencias

Ramiro Alcocer, “oocities” http://www.oocities.org/valcoey/opengl.html, 2001

Dr. Conrad Sanderson, Dr. Ryan Cutien, “Armadillo” http://arma.sourceforge.net/, 2016

Sonar Systems, “YouTube” https://www.youtube.com/watch?v=L2aiuDDFNIk&t=326s, 2015

James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes, Computer Graphics PRINCIPLES AND PRACTICE, IBM Editorial Board, 1990

Michel Jiménez, “blog”, https://michellejimenezv.wordpress.com/acerca-de/, 2014

Ricardo, “Blog”, http://grafi-ricardo.blogspot.com/2012/04/glulookat.html, 2012