Simulación de carácter catastrófica del Sistema Solar en OpenGL

Wladimir Albornoz, *estudiante, Usach* y Andrés Barrera, *estudiante, Usach*

Resumen—En el siguiente informe se detallan los aspectos preliminares para el proyecto de la asignatura de computación Gráfica, en este se presenta el modelamiento de sistema solar con un cinturón de asteroides, todo esto en un sistema de software. Otras implementaciones realizadas en este trabajo fueron la aplicación de texturas en el fondo de la escena utilizando la técnica de SkyBox y la implementación de un menú para facilitar la navegación entre las funcionalidades del programa y el experimento. Fue implementado en OpenGL bajo el sistema operativo GNU/Linux.

Index Terms—Sistema solar, OpenGL, rotación, traslación, escalamiento,objetivos, herramientas a utilizar, plan.

1. Introducción

La idea principal del proyecto es generar una simulación de la muerte del sol y la absorción de algunos planetas producto de esto, fundamentalmente si se rompiera la resistencia existente, debido a que el sol se transformará en una gigante roja dentro de 10 mil millones de años aproximadamente [1].
Esta simulación tiene un carácter científico, ya que está demostrado que cuando se concrete el fin del sol terminaría absorbiendo a los planetas que se encuentren cercanos a el, se toman como base las diferentes visiones e información que se tiene del espacio-tiempo.
Debe considerarse que lo que se está presentando toma como base parte de un proyecto ya implementado [2], cabe destacar que esto resulta mas complejo debido a que la base está realizado con aspectos, métodos, módulos y herramientas diferentes, teniendo que considerar un tiempo extra para injertar de manera perfecta esto a el proyecto que se está realizando.

2. Objetivos

2.1. Objetivo General

Implementar gráficamente la simulación catastrófica del sistema solar y que se aprecie lo mas cercano a la realidad posible.

2.2. Objetivos Específicos

- Conocer y relacionar términos de la computación gráfica para el desarrollo del proyecto [3].
- Entender y aprender a utilizar OpenGL para la implementación y representación de elementos de la computación gráfica.
- Determinación de herramientas útiles para el desarrollo.
- Descubrir librerías que puedan ayudar tanto en la codificación como en la implementación.
- Lograr dominar el lenguaje y las herramientas a utilizar.

3. Descripción de la problemática

3.1. Motivación

Resulta muy interesante la oportunidad de generar una simulación gráfica el sistema solar [4] conocido (figura 1), mas detalladamente del fin de el sol como se conoce, debido a la importancia que este sistema tiene para toda la humanidad; la mayor motivación para el desarrollo es en efecto poder implementar el proyecto, además con ello se pretende aprender y a la vez poner en práctica conceptos de la computación gráfica como la rotación, traslación, escalamiento y el modelado en 3 dimensiones.

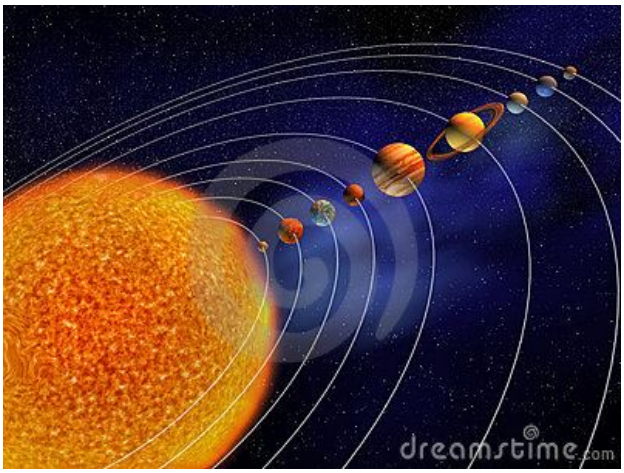

Figura 1: Sistema solar conocido

3.2. Definición de la problemática

La mayoría de las implementaciones gráficas conocidas muestran simulaciones de la realidad de los planetas y del sistema solar, sus trayectorias y las diferencias entre las velocidades de rotación de cada uno; pero ¿que sucedería si las órbitas de los planetas aumentara y sumados al aumento de la masa del sol, este atrajera a todos los planetas existentes en el sistema conocido?, el fin de el sol y de los planetas como los conocemos es lo que se desea implementar.
Skybox es una técnica que permite que una escena se vea más grande y más impresionante, envolviendo al usuario con una textura que es posible apreciar en una cámara de 360° [5].

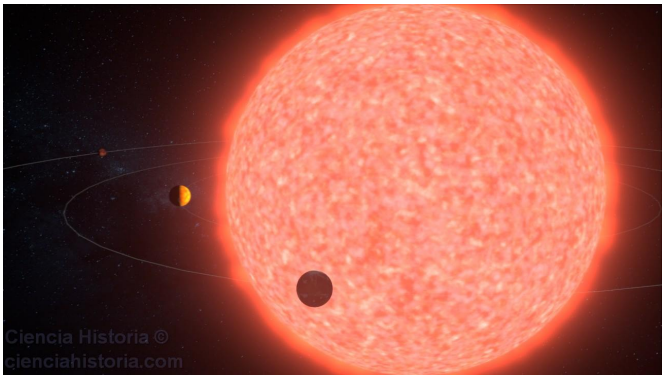

Figura 2: El Sol convertido en gigante roja llegando casi a la órbita de Venus, la Tierra tiene un futuro incierto

4. Descripción de la solución propuesta

Tomando como base el proyecto que solo muestra una representación del sistema solar, se pretende que la representación de las trayectorias de los planetas sea automática, con esto se refiere a que se vea el movimiento continuo de la trayectoria de los planetas y que eventualmente de a poco los planetas comiencen a cambiar su trayectoria, los cuales sumados al aumento del diámetro del sol, como se muestra en 2, empiecen a ser absorbidos por este, lo cual los hará desaparecer [6].
Para ello habría que:

- Modificar algunas funciones relacionadas con la traslación de estas diferentes figuras (diferentes planetas)
- Editar y acondicionar las formulas de elipse.
- Aumentar el radio del sol(*escalamiento*).

5. Modelamiento

5.1. Función de la elipse

A continuación en la figura 3 se puede ver una elipse para posteriormente analizar su función matemática:

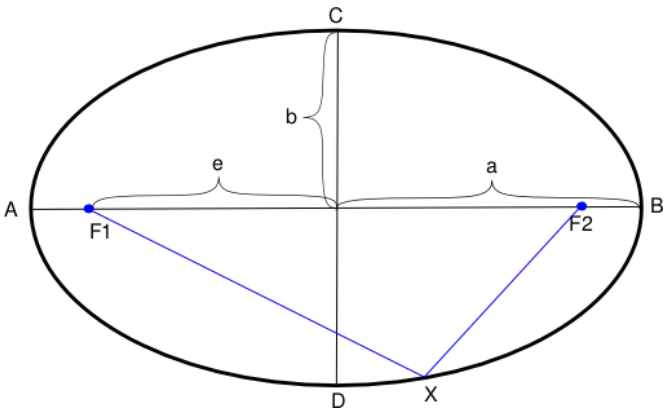

Figura 3: Elipse

Se utiliza la función de la elipse:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2}$$

Donde $a > 0$ y $b > 0$, las coordenadas de las abscisas. La elipse se encuentra centrada en el origen (h, k = 0). La ecuación de la elipse modela la ubicación de la órbita de los planetas.

5.2. Transformaciones 3d

Se utilizan una serie de trasnformaciones en el desarrollo del problema, estas seran mostradas a continuación:

5.2.1. Rotación

Claramente como se desea implementar un sistema solar, un detalle importante a destacar es la rotación que tiene cada planeta; la rotación está definida por la siguiente matriz:

$$\begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$$

En vista de que se trabaja en tres dimensiones es necesario definir las matrices para cada tipo de rotación que se pueden obtener desde la matriz de rotacion general de 2x2:
Rotación sobre el eje x:

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

Rotación sobre el eje y:

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

Rotación sobre el eje z:

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

Se describe como rotación en los planos xy, yz y zx, pues resulta más fácil asociar la rotación en 3 dimensiones de esa forma. La órbita de los planetas se genera aplicando una rotación sobre el eje y (En el origen, teniendo el solo como referencia) con un ángulo $0° \leq \theta \leq 360°$, el cual aumenta en cada frame de la escena.

5.2.2. Traslación

La matriz que describe la traslación de los objetos en la escena es la siguiente:

$$T = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} v_x \\ v_y \\ v_z \\ 1 \end{bmatrix} = \begin{pmatrix} v'_x, & v'_y, & v'_z & 1 \end{pmatrix}$$

La traslación permite saber el radio que tendrá la órbita del planeta respecto al sol. Es en torno a esta transformación que se aplica la rotación, por lo tanto para cada frame de la escena se aplica un pequeño aumento en la rotación respectiva.

5.2.3. Escalamiento

La matriz que describe el escalamiento de los objetos en la escena es la siguiente:

$$T = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} v_x \\ v_y \\ v_z \\ 1 \end{bmatrix} = \begin{pmatrix} v'_x, & v'_y, & v'_z, & 1 \end{pmatrix}$$

El escalamiento permite simular el fin del sol, ya que según la ciencia [1] este empezaría a aumentar su radio a medida que pasan los años absorbiendo los primeros 3 planetas **Mercurio**, **Venus** y probablemente la **Tierra**. Ya alcanzando el limite de tamaño, su atmósfera empezará a contraerse hasta transformarse en una enana blanca. De esta manera el proceso de escalamiento nos permitirá simular el comportamiento del sol en esta teoría científica.

5.3. Distancia entre dos puntos

Esta fórmula se utiliza para determinar la distancia entre 2 puntos en un plano en 2D, aunque el sistema está representado en 3 dimensiones. Los planetas se mueven respecto al eje y y la posición de los mismos está determinada sólo por sus coordenadas en el eje x y en el eje z:

$$x_{new} = \frac{\sqrt{X^2_{final} + X^2_{inicial}}}{100}$$

$$z_{new} = \frac{\sqrt{Z^2_{final} + Z^2_{inicial}}}{100}$$

la distancia total entre el punto final y el inicial es dividida por un número aleatorio con ello se genera la traslación de un objeto desde un punto hacía otro; lo que se traslada es la viewport.

6. DISEÑO DEL PROBLEMA

A continuación, se especifica un diseño preliminar mas un diseño detallado del problema y de la solución

6.1. Diseño preliminar

La formulación del modelo en base al sistema solar y a la teoría del fin del sol en aproximadamente 10.000 millones de años aprox. es un modelo, debido a que los cálculos realizados son a escala respecto al sistema solar real, es decir, X pixeles son equivalentes a algunos años luz. Los tamaños de los planetas también son a escala, los tamaños de cada uno son equivalentes a la vida real, así también su color y movimiento.
Dado esto, en pocas palabras seria de la siguiente forma:

- Calcular la trayectoria de los planetas alrededor del sol
- Implementar un menú que permita navegar entre las distintas funcionalidades presentes en el programa
- Aplicar la técnica de SkyBox en la escena

6.2. Diseño detallado

OpenGl es el principal entorno de desarrollo, para la creación de aplicaciones graficas portátiles e interactivas en 2D y 3D. La API se suele utilizar para interactuar con una unidad de procesamiento gráfico (GPU), para lograr la aceleración por hardware de renderizado. Desde su introducción en 1992, OpenGl se ha convertido en la Interfaz de Programación de Aplicaciones (API) gráficas más usada por la industria, brindando miles de aplicaciones a una gran variedad de plataformas de computación. A su vez, OpenGL fomenta la innovación y velocidad del desarrollo de aplicaciones incorporando una alta variedad de formas de rendering, mapeo de texturas, efectos especiales y otras poderosas funciones de visualización [7].
Se utilizo el Entorno de Desarrollo Integrado (IDE) llamado NetBeans, el cual permite detectar rápidamente errores de sintaxis. Permite también iniciar en modo debug, herramienta con la cual es posible realizar un seguimiento del valor de cada variable en cada frame de la aplicación. Además, resulta mucho más fácil el proceso de compilación, ya que es un proceso mas gráfico que la ejecucion en una consola, además no discrimina el sistema operativo a ocupar.

OpenGL trabaja con una pila de matrices, las cuales representan cada objeto que se muestra en la escena. A continuación se detallará el trabajo y la sintaxis de cada una de ellas.

```
void mercurio (float x, float y, float z) {
     glPushMatrix();
     //Texturas
     //Transformaciones
     glRotate(theta, 0, 1, 0);
     glTranslate(x, y, z);
     //...
     glPopMatrix();
}
```

Extracto de código que nos permite demostrar cuando se crea una nueva matriz
Las transformaciones se aplican en un orden predefinido, es decir, primero se realiza la traslación (Desde el origen hacía el punto (x, y, z)) para posteriormente realizar la rotación sobre el eje deseado.Esto es apoyado por la literatura y la cátedra.

Las transformaciones geométricas en OpenGL están implementadas, a continuación se mostraran en mas detalle cada una de ellas:

6.2.1. *Rotación*

Para rotar, en openGL se utiliza la función **glRotate**, la cual rota en torno a un eje a elección, utilizando transformaciones geométricas. La función tiene la siguiente forma:

```
void glRotatef (GLfloat  angle,
                GLfloat  x,
                GLfloat  y,
                GLfloat  z );

void glRotated (GLdouble  angle,
```

```
                GLdouble  x,
                GLdouble  y,
                GLdouble  z );
```

Donde $angle$ es el ángulo que se le aplicará y (x, y, z) es sobre el eje que se aplicará la rotación.
Resulta importante destacar que, tanto el ángulo de rotación como las coordenadas, pueden ser variables de tipo **float** y **double**.
Dicha función no utiliza la misma matriz para realizar la transformación geométrica que se mencionó anteriormente, la matriz que utiliza es una variación:

$$\begin{pmatrix} x^2(1-c)+c & xy(1-c)-zs & xz(1-c)+ys & 0 \\ yx(1-c)+zs & y^2(1-c)+c & yz(1-c)-xs & 0 \\ xz(1-c)-ys & yz(1-c)+xs & z^2(1-c)+c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Donde: $c = s = \sin\theta$ y $\cos\theta$.

6.2.2. Traslación

Para trasladar en openGl se utiliza la función **glTranslate**, la cuál está determinada por un vector que se proyecta desde el origen hasta la coordenada (x, y, z).
La función tiene la siguiente forma:

```
void glTranslatef (GLfloat  x,
                   GLfloat  y,
                   GLfloat  z);

void glTranslated (GLdouble  x,
                   GLdouble  y,
                   GLdouble  z);
```

(x, y, z) es hacía dónde se quiere mover el objeto. Esta función utiliza la misma matriz descrita en la sección 6.2.1

6.2.3. Escalamiento

Para escalar en openGl se utiliza la función **glScale**, la cuál está determinada por un vector que se permite escalar desde el origen hasta la coordenada (x, y, z).
La función tiene la siguiente forma:

```
void glScaled(GLdouble  x,
              GLdouble  y,
              GLdouble  z);

void glScalef(GLfloat  x,
              GLfloat  y,
              GLfloat  z);
```

(x, y, z) son los valores en los que se escalará el objeto. Esta función utiliza la misma matriz descrita en la sección 6.2.1

6.2.4. Vista

Para fijar la vista en un punto determinado desde dónde se observará la escena, en openGL se utiliza la función **gluLookAt**.
La función tiene la siguiente forma:

```
void gluLookAt (GLdouble eyeX,
                GLdouble eyeY,
                GLdouble eyeZ,
                GLdouble centerX,
                GLdouble centerY,
                GLdouble centerZ,
                GLdouble upX,
                GLdouble upY,
                GLdouble upZ);
```

Donde:

- **centerX, centerY, centerZ**: Es la coordenada (x, y, z) que especifica el punto de referencia hacía el cual se posicionará el punto de vista.
- ***upX, upY, upZ***: Es el vector que determina la dirección.
- **eyeX, eyeY, eyeZ**: Es la coordenada (x, y, z) en la que se posicionará el punto de vista. Se debe entender el punto de vista como un objeto más en la escena. Al verlo de esta manera, podemos concluir que ésta tiene asociada una matriz, para así poder modificar la orientación y/o ubicación del punto de vista en la escena.
  En otras palabras, se pueden aplicar transformaciones geométricas (Rotación y traslación) para situar la vista donde se requiera. Esto se puede representar con lo siguiente:

$$M = T * R$$

Siendo M: matriz de la viewport, T: matriz de traslación y R: matriz de rotación.
La imagen a continuación, puede aclarar un poco mas las cosas:

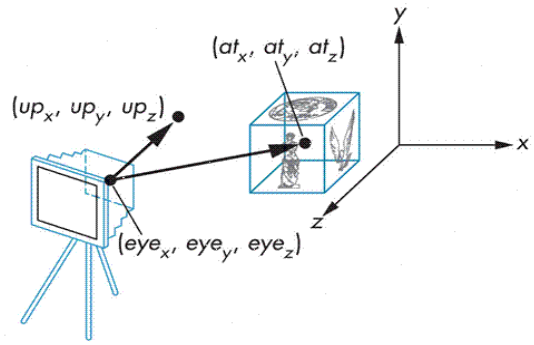

Figura 4: Representación gráfica del punto de vista en la escena

6.2.5. Órbita

La función para empezar la rotación y darle un sentido, en openGL se conoce como **getOrbitStartPoint**, esta función recibe como parámetro el ángulo de rotación del planeta, el vector en dónde se almacenarán las coordenadas luego de la rotación, y la posición en (x, y, z) respecto al origen desde la cuál se comienza a realizar la órbita.

```
void getOrbitStartPoint (double angulo,
                double *coordenadas,
                float x, float y, float z) {
     coordenadas[0] = cos(angulo)*x +
         sin(angulo)*z;
     coordenadas[1] = y;
     coordenadas[2] = -sin(angulo)*x +
         cos(angulo)*z;
}
```

Esta función realiza el producto de la multiplicación de la matriz de rotación respecto al eje y por el vector de la posición inicial, la cual corresponde al planeta correspondiente.

6.2.6. loadSkyBox

La idea detrás de la técnica Skybox, es renderizar un cubo gigante con una textura particular, posicionando la escena y la viewport en el centro [8].
Se utilizará la técnica basada en texturas (mapeo de la textura en un cubo), se aplica un tipo especial de textura en el cubo. Esta textura, se crea de forma que para cada cara del cubo, se posicione la textura que le corresponde y que al interior del cubo las esquinas estén perfectamente alineadas, para poder asi crear la sensación de una textura continua.

Un ejemplo de esta técnica es la siguiente imagen:

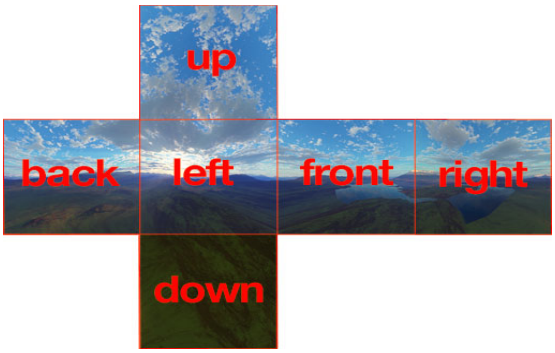

Figura 5: Ejemplo de una textura que se pueden asignar a las caras de un palco cúbico, con caras etiquetadas

El procedimiento que se realiza, es cortar cada segmento en una imagen diferente y, en el interior del programa, cada una se trabaja como textura independiente. Posterior a eso, se crea un cubo gigante y se le aplica la textura, el resultado se muestra a continuación:

Figura 6: Ejemplo de una textura para una cúpula celeste hemisférica

El resultado es un cubo, la aplicación de esta técnica al proyecto es poder darle la forma continua al sistema solar, sin tener un fin a la vista.
Se utiliza el formato de imagen (TGA), ya que es la forma más óptima de ahorrar los costos de renderizando [3], utilizando un formato de imagen que es de fácil lectura y de una calidad menor.

6.3. Menu de opciones

Como prototipo, se tienen las siguientes opciones:

1. **Catástrofe**: Inicia el fin del Sol
2. **Iniciar movimiento**: Inicia la animación de los planetas orbitando.
3. **Detener movimiento**: Detiene la animación de los planetas orbitando.
4. **Salir**: Salir del sistema solar.

Cabe mencionar que se ha completado el 100 % de los requerimientos dados al inicio del problema, se espera que el resultado sea apreciado en su totalidad y que este sea de ayuda para algún proyecto a futuro.

7. Metodología y herramientas a utilizar

7.1. Metodología

La metodología a utilizar es el análisis y diseño estructurado ya que con esta metodología se tiene un orden lógico en los pasos a seguir para el desarrollo correcto del proyecto.

7.2. Herramientas

Preliminarmente se tienen las siguientes herramientas para el desarrollo:

7.2.1. Equipos

Se posee 2 notebooks con similares características, procesadores Intel core I3 y I5, CPU de 2.4 GHz, 8 Gb de memoria Ram y distribución de Linux Ubuntu 14.04 LTS.

7.2.2. Herramientas de programación

Preliminarmente se establecen las siguientes herramientas para la programación, tales como:

- Netbeans IDE 8.0.2
  Entre algunas cosas destaca la rápida observación de errores de sintaxis y fácil compilación.
- OpenGL
  Principal entorno de desarrollo, para la creación de aplicaciones gráficas portátiles e interactivas en 2D y 3D [7].

8. Análisis de resultados

8.1. Validación de resultados

Como se especificó en la presentación preliminar de la idea, para realizar esta aplicación se toma como base un proyecto anterior del ramo de computación gráfica, el cual se llama Representación del sistema solar en OpenGL". De este código se toman las funciones principales para generar la representación.
En el código se representa Plutón que no se encontraba representado, además se comienza generando todos los planetas alineados tras el Sol para mantener un orden mas lógico en un principio.
Lo principal novedad que se trabajó fue la muerte del sol.
Se toma como base científica las distancias normalizadas de los planetas y sus velocidades de rotación.

8.2. Rendimiento preliminar

En esta etapa final del proyecto, el equipo de trabajo se enfoca en implementar las funciones más importantes del proyecto base además de aprender sobre la utilización de la herramienta OpenGL. A la base tomada para el desarrollo se le eliminan una serie de funciones que en realidad no serán necesarias por lo tanto esto entregara un código mas compacto y fácil de comprender.
Preliminarmente no se identifican problemas relacionados a la presentación en pantalla de la aplicación al ser ejecutada, la velocidad de movimiento (rotación) es más que suficiente para identificar los movimientos de los planetas.
El rendimiento gráfico de la aplicación es prudente(por la sencillez), rápido(por como se aprecia) y optimo(por como se realiza).

9. Plan de Trabajo

Se realiza una carta Gantt para darle fechas limites a cada tarea que este proyecto conlleva. Se adjunta para no entorpecer el modelo de este documento y para que la carta Gantt se pueda analizar de mejor manera.

10. Experimentos

En la imagen se puede observar los planetas alineados, esto muestra el punto de partida del programa. En la imagen se puede observar un

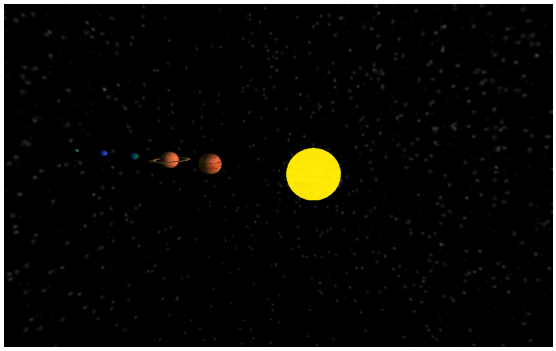

Figura 7: Experimento con las técnicas que se mencionaron aplicadas

momento de la órbita de los planetas, cuando estos ya están orbitando y trasladándose a través de su propia elipse y su propia velocidad.

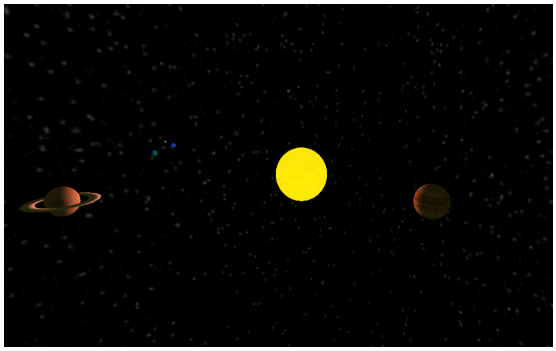

Figura 8: Experimento que muestra las posiciones de los planetas luego de rotaciones

11. Conclusiones

Como inicio para un buen proyecto, siempre se deben llevar a cabo investigaciones que puedan determinar cual tema abordar y que herramientas utilizar para un optimo desempeño durante este.
El tema abordado nos llenó de optimismo para trabajar en el, sumado a los conceptos de computación gráfica aprendidos en el aula más la investigación pertinente, nos motivó a dar el 100 % de nuestro esfuerzo para hacer de este un gran proyecto.

Referencias

[1] R. G. Baker, "The sun-earth connect 1: A fractional -matrix of solar emissions compared to spectral analysis evidence of solar measurements and climate proxies," 2015.
[2] S. P. J. Riveros, "Sistema solar," 2008.
[3] J. Foley, *Computer graphics : principles and practice*. Reading, Mass: Addison-Wesley, 1995.
[4] Astromia.com. (Diciembre 15, 2014.) Astronomía educativa: Tierra, sistema solar y universo. [Online]. Available: http://www.astromia.com/solar/nubeoort.htm
[5] Atspace. (Diciembre 15, 2014.) Tutorial 25 - skybox. http://ogldev.atspace.co.uk/www/tutorial25/tutorial25.html.
[6] E. Siegel. (Enero 2, 2014.) The far future of our solar system. https://medium.com/starts-with-a-bang/the-far-future-of-our-solar-system-31d05b036596.
[7] OpenGL. (Diciembre 15, 2014.) Opengl overview. [Online]. Available: https://www.opengl.org/about/
[8] J. YANG. (Diciembre 15, 2014.) Treasure hunting. http://www.cs.ucla.edu/~jdyang/CS274C_project_report.pdf.