1. Introducción

Desde los orígenes de la humanidad, siempre hemos mirado hacia el cielo, tratando de entender qué son esas luces que iluminan nuestras noches. Con miles de años de evolución, ahora comprendemos mucho sobre el cosmos, pero nuestro sueño de controlar y explorar más allá sigue vivo. Utilizando las herramientas que nos proporciona la tecnología actual, en este trabajo hemos recreado el funcionamiento de nuestro sistema solar.

Esta simulación no solo busca mostrar visualmente los planetas y sus órbitas, sino también proporcionar una experiencia interactiva y educativa. Aprovechando el poder de OpenGL y C++, hemos desarrollado un modelo que permite explorar el sistema solar desde diferentes perspectivas, utilizando una doble cámara que ofrece vistas para apreciar las orbitas y otra para visualizar el movimiento compartido de todo el sistema. Este proyecto combina nuestra pasión por la astronomía con las técnicas avanzadas de gráficos por computadora.

2. Descripción del proyecto

El proyecto consiste en una simulación del sistema solar, incluyendo los ocho planetas, asteroides y otros elementos adicionales que dependen de la cámara que se esté utilizando. La simulación permite cambiar entre dos cámaras, cada una ofreciendo una perspectiva única del sistema solar:

- **Cámara 1**: Proporciona una vista global del sistema solar, mostrando el rastro que dejan los planetas y el sol a medida que avanzan en el espacio. Esta vista es útil para observar el movimiento y la trayectoria de los cuerpos celestes en su conjunto.

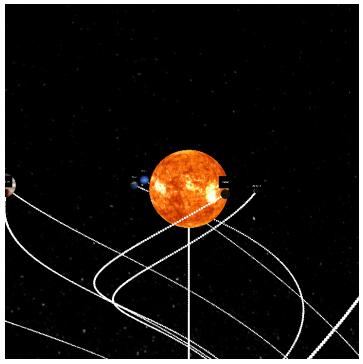


Figura 1: Cámara 1

- **Cámara 2**: Ofrece una vista más detallada de las órbitas que siguen los planetas alrededor del sol. Esta perspectiva permite apreciar mejor las trayectorias elípticas y las relaciones espaciales entre los distintos cuerpos del sistema solar.

2.1. Herramientas

Para el desarrollo de este proyecto, utilizamos varias herramientas y tecnologías clave. A continuación, se describen las más importantes:

- **OpenGL:** Una API (Interfaz de Programación de Aplicaciones) estándar para la creación de gráficos 2D y 3D. OpenGL es ampliamente utilizado para desarrollar aplicaciones gráficas debido a su flexibilidad y potencia.

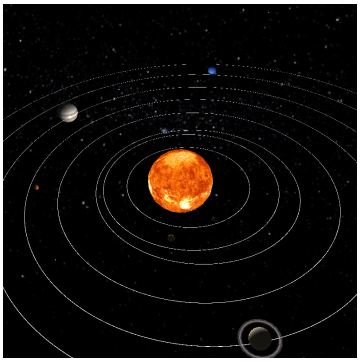


Figura 2: Cámara 2

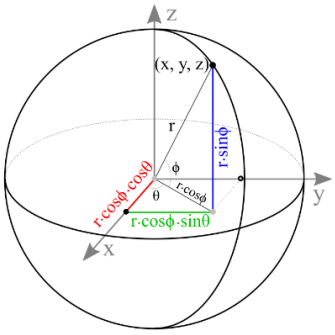
- **GLFW:** Una biblioteca para la creación de ventanas y el manejo de eventos de entrada (teclado, ratón, etc.). GLFW facilita la creación de contextos de OpenGL y la gestión de ventanas y entradas de usuario.
- **GLAD:** Una biblioteca de carga de funciones de OpenGL. GLAD ayuda a gestionar las extensiones de OpenGL, permitiendo acceder a todas las funciones necesarias de OpenGL en nuestro proyecto.
- **GLM (OpenGL Mathematics):** Una biblioteca de matemáticas para gráficos basada en la especificación GLSL (OpenGL Shading Language). GLM proporciona clases y funciones para manejar vectores y matrices, esenciales para las transformaciones y cálculos geométricos en gráficos por computadora.
- **stb_image:** Una biblioteca para la carga de imágenes. Utilizamos stb_image para cargar texturas en diferentes formatos (JPG, PNG, etc.), las cuales se aplican a los planetas y otros objetos en la simulación.
- **Solar System Scope[Sco23]:** Es una pagina web que contiene las texturas de todos los cuerpos celestes que conforman el sistema solar, estas texturas fueron aplicadas en la generacion de los planetas y el sol.

2.2. Matemáticas Utilizadas

Las matemáticas son fundamentales para la creación y automatización de procesos en gráficos por computadora. A continuación, explicamos las áreas clave donde las matemáticas jugaron un papel esencial en nuestro proyecto.

2.2.1. Creación de Esferas

Para crear las esferas que representan los planetas, primero investigamos la fórmula matemática para obtener los puntos en una esfera [Ahn23]. La fórmula utilizada es la siguiente:



El proceso comienza seleccionando una cantidad de subdivisiones para determinar la resolución de la esfera. El cálculo del paso se realiza utilizando la fórmula: $paso = 360/subdivisiones$ Durante los cálculos, el ángulo aumenta en incrementos de $\phi * paso$, lo que nos permite ajustar la resolución de las esferas según sea necesario. Este método nos asegura una distribución uniforme de los vértices en la superficie de la esfera.

2.2.2. Obtener Radio

para obtener el radio fue utilizado una ecuación de elipse basada en los valores de perihelio y afelio, que fueran usados para generar los componentes a y b de la elipse. Con eso, el cálculo desarrollado es una función que resulta en el radio basado en el ángulo de rotación.

$$r(\theta) = \frac{ab}{\sqrt{(b\cos(\theta))^2 + (a\sin(\theta))^2}}$$

dónde:

- a: mayor eje de la elipse
- b: menor eje de la elipse
- θ: Angulo de rotación

2.2.3. Orientacion a camara

En la camara 1 se necesita una propiedad para los objetos `Estela` la cual es que siempre estén mirando hacia la dirección a la camara y eso se logra de la siguiente manera

1. **Cálculo de la Dirección hacia la Cámara**: Se invierte la matriz de vista para obtener la posición de la cámara en el espacio global:

$$\vec{\text{camera}}\text{Pos} = \text{inverse}(\text{viewMatrix})[3]$$

La posición del objeto se extrae de la cuarta columna de la matriz de modelo:

$$\vec{\text{obj}}\text{Pos} = \text{orbitalModel}[3]$$

La dirección del objeto a la cámara se calcula como:

$$\vec{\text{direction}} = \frac{\vec{\text{camera}}\text{Pos} - \vec{\text{obj}}\text{Pos}}{\|\vec{\text{camera}}\text{Pos} - \vec{\text{obj}}\text{Pos}\|}$$

2. **Cálculo de los Vectores de Orientación**: `Right (derecha)`: Se calcula como el producto cruzado del vector $\vec{up} = (0, 1, 0)$ y la dirección hacia la cámara:

$$\vec{right} = \frac{\vec{up} \times \vec{direction}}{\|\vec{up} \times \vec{direction}\|}$$

`Up (arriba)`: Se recalcula el vector $\vec{up}$ para que sea perpendicular a la dirección hacia la cámara y al vector $\vec{right}$:

$$\vec{up} = \vec{direction} \times \vec{right}$$

3. **Construcción de la Matriz de Rotación**: Se construye una nueva matriz de rotación donde las primeras tres columnas corresponden a los vectores $\vec{right}, \vec{up}, \vec{direction}$:

$$\text{rotation} = \begin{pmatrix} \vec{right} & 0 \\ \vec{up} & 0 \\ \vec{direction} & 0 \\ 0 & 1 \end{pmatrix}$$

4. **Aplicación de la Rotación**: Se aplica la rotación al objeto, asegurándose de que su orientación sea correcta. La matriz de modelo final es:

$$\text{orbitalModel} = T(\vec{\text{objPos}}) \cdot \text{rotation}$$

donde $T(\vec{\text{objPos}})$ es la matriz de traslación que coloca el objeto en su posición calculada.

2.2.4. Orbitas

Para la generacion de las orbitas se necesito crear un nuevo objeto llamado **Orbita**, estos objetos hacen el calculo normal de un circulo $\cos\phi * radio$ y $\sin\phi * radio$, con la modificacion que el radio es variable, el radio se obtiene con la funcion explicada previamente de modo que cada planeta tiene su orbita eliptica bien graficada

2.3. Elementos en escena

En esta animación, se decidió no emplear un grafo de escena debido a la simplicidad de la organización requerida. En su lugar, la organización de los elementos depende de la cámara que se esté visualizando.

2.3.1. Cámara 1

En la Cámara 1 se puede visualizar la estela que dejan los planetas. Esto se logra mediante objetos auxiliares llamados **Estela**. Estos objetos son círculos con la capacidad de mirar siempre hacia la cámara, utilizando la fórmula explicada en la sección anterior. A estos objetos se les aplica una función de movimiento similar a la de cada planeta, pero con un desfase. Esto significa que si el planeta `Tierra` está en la posición $50°$, la primera estela estará en $49°$, la siguiente en $48°$, y así sucesivamente, creando un efecto de estela continua.

Cada cuerpo celeste tiene su propio conjunto de estelas, lo que resulta en un total de nueve grupos de estelas, correspondientes a los ocho planetas y el sol.

2.3.2. Cámara 2

En la Cámara 2 se utilizan objetos llamados **Orbita**, que están alineados con las trayectorias de los planetas. Esto permite visualizar claramente las órbitas que sigue cada planeta alrededor del sol.

2.4. Shaders

En nuestro proyecto utilizamos shaders basados en los proporcionados por LearnOpenGL [GL], pero con modificaciones significativas que permiten una iluminación dinámica en tiempo real.

2.4.1. Vertex Shader

La principal diferencia en nuestro vertex shader es la siguiente modificación:

```
FragPos = vec3(model * vec4(aPos, 1.0));
```

Esta línea de código actualiza la posición de los vértices después de que se hayan aplicado todas las transformaciones. Esto significa que, a medida que un objeto se mueve en el espacio, la iluminación se ajusta automáticamente para reflejar la nueva posición del objeto, proporcionando un efecto de iluminación más realista y dinámico.

2.4.2. Fragment Shader

Hemos realizado una modificación simple pero significativa en el fragment shader:

```
vec3 result = ((diffuse + specular) + vec3(0.1f)) * texColor.rgb;
```

Esta modificación asegura que incluso si un objeto está orientado de espaldas a la luz (es decir, no recibiendo luz directa), se le añade una iluminación mínima. Esto se logra incrementando la iluminación innata del objeto, lo que garantiza que todos los objetos tengan un nivel básico de iluminación, mejorando la visibilidad y la calidad visual de la escena.

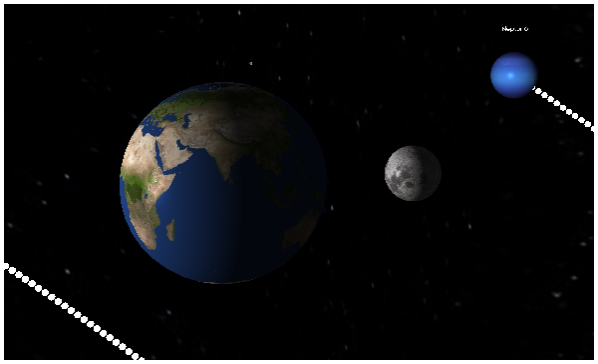


Figura 3: Tierra y Luna siendo afectados por iluminacion

2.4.3. Shaders sin iluminación

Pese a que queriamos generar el efecto que causa el sol al iluminar los objetos nos dimos cuenta que habían casos en los cuales este efecto no era necesario, las estelas y orbitas son elemenos para los cuales se crearon shaders simples, solo con las cualidades

de perspectiva y modelo, esto para que sean mas apresiables y no den un efecto raro dependiendo de donde se ven ya que son objetos que requerimos que esten lo mas fiel a su textura original.

3. Experimentos

Los resultados originales del programa eran buenos, pero presentaban un alto costo computacional. Aunque los componentes actuales son capaces de realizar cientos de cálculos por segundo, todavía tienen limitaciones. Por lo tanto, tuvimos que sacrificar un poco la calidad del resultado final para que el programa pudiera ejecutarse en computadoras menos potentes.

3.1. Optimización de Estela

La estela fue uno de los objetos que inicialmente causó problemas significativos. En la versión original, se crearon 500 objetos de círculos por planeta, lo que resultó en un gran consumo de memoria RAM y recursos (9 GB de RAM y un 77 % de uso en una tarjeta gráfica RTX 4060). Este problema era tan grave que el programa tardaba 30 segundos en iniciarse, debido a la carga y creación de todos los elementos de estela.

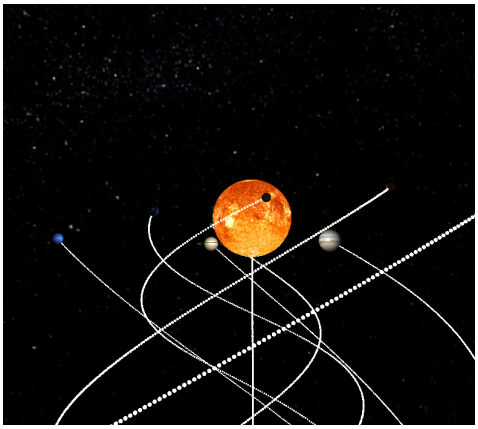


Figura 4: Orbitas de los planetas

Para solucionar esto, se implementó una opción más eficiente: en lugar de crear múltiples objetos de estela, se optó por crear un solo objeto de estela por planeta y renderizarlo 250 veces por frame. Esto resultó en una carga menor en la memoria RAM y redujo el uso de la gráfica al 35 %, mostrando una mejora notable entre las versiones.

3.2. Resolución de Objetos

Originalmente, todos los objetos se cargaban con una alta resolución, lo que significaba que estaban bien definidos. Sin embargo, esto generaba una mayor cantidad de cálculos en cada grupo de iluminación. Para optimizar esto, realizamos pruebas y ajustes en las resoluciones de los objetos hasta encontrar un equilibrio entre una buena apariencia visual y una resolución lo suficientemente baja para ahorrar cálculos en tiempo real de la iluminación.

4. Resultados Finales

Los resultados obtenidos cumplen con las metas y objetivos planteados en el curso. Se implementaron iluminación dinámica, transformaciones y perspectiva sin el uso de librerías de modelado externo, aplicando exclusivamente fórmulas matemáticas para la creación de los modelos. Además, se utilizaron dos tipos de shaders y dos cámaras para enriquecer la experiencia visual.

El programa demostró ser eficiente durante toda la ejecución, sin requerir una gran cantidad de memoria RAM. A pesar de ejecutarse a 144 Hz en la computadora de prueba, solo utiliza el 35 % de los recursos de la tarjeta de video.

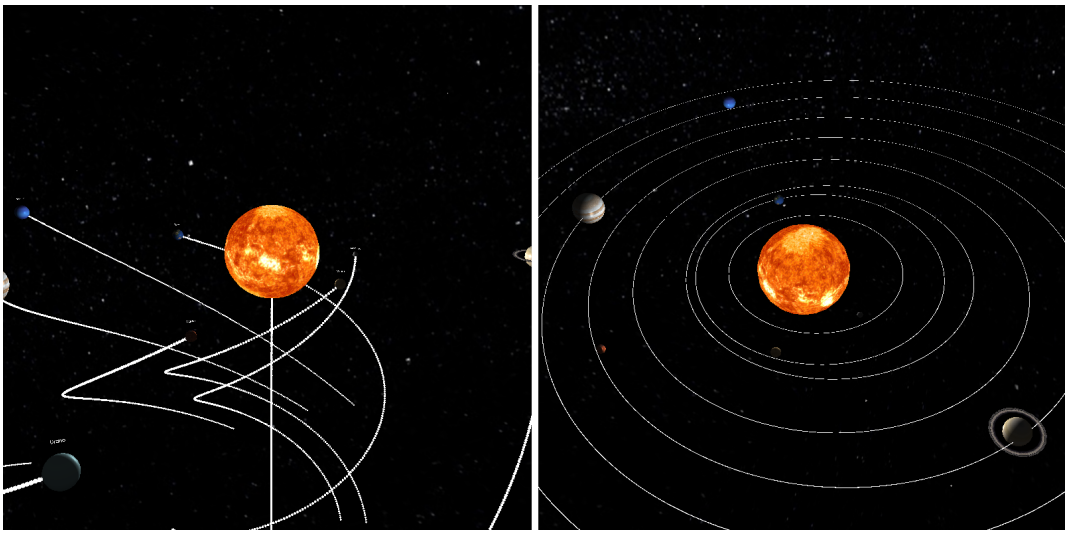


Figura 5: Resultado Final

Como se observa el programa logra ejecutar de buena manera ambas camaras en simultaneo y en la camara de la izquierda dentro de la navegación se puede apreciar los nombres de los planetas a medida que uno se desplaza por el entorno

5. Conclusiones

En conclusión, el desarrollo del trabajo fue una actividad gratificante ya que se lograron las metas sin obstaculos grandes, pudimos aprender el uso de la libreria `OpenGL` y sus derivados y concenptos de computación grafica con facilidad.

Referencias

[Ahn23] Song Ho Ahn. Opengl sphere. https://www.songho.ca/opengl/gl_sphere.html, 2023. Accessed: 2023-07-01.

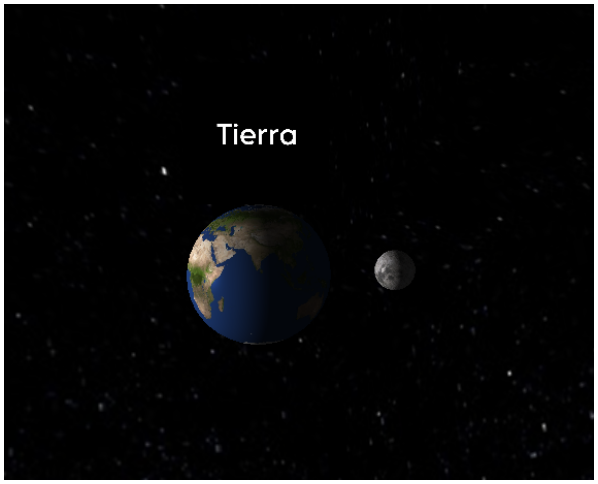


Figura 6: Planeta Tierra con su nombre

[GL] Learn Open GL. Learn open gl. https://learnopengl.com/Lighting/Basic-Lighting. Accessed: 2023-07-01.

[Sco23] Solar System Scope. Solar system textures. https://www.solarsystemscope.com/textures/, 2023. Accessed: 2023-07-01.