Sistema Solar Interactivo 3D:
Visualización y Aplicación de Conceptos de Computación Gráfica

Joiner Antonio Davila Saiz

Universidad Manuela Beltran

Facultad de Ingeniería

Computación Gráfica

2025

1. DESCRIPCIÓN DEL PROYECTO

1.1 Tema Seleccionado

Sistema Solar Interactivo en 3D

Se ha desarrollado una aplicación web interactiva que presenta una visualización tridimensional del sistema solar, incluyendo el Sol y los ocho planetas principales: Mercurio, Venus, Tierra, Marte, Júpiter, Saturno, Urano y Neptuno.

1.2 Objetivos del Proyecto

Objetivos Educativos:

- Proporcionar una herramienta visual e interactiva para el aprendizaje sobre astronomía básica
- Facilitar la comprensión de conceptos como órbitas planetarias, rotación axial y escalas relativas
- Mostrar información detallada sobre cada planeta mediante interacción directa

Objetivos Técnicos:

- Demostrar la aplicación práctica de conceptos fundamentales de computación gráfica
- Implementar un sistema de renderizado 3D en tiempo real utilizando tecnologías web
- Crear animaciones fluidas y sistemas de interacción usuario-objeto
- Aplicar técnicas de iluminación, sombreado y materiales en entornos 3D

1.3 Alcance del Proyecto

El proyecto abarca:

- Creación y renderizado de objetos 3D (geometrías esféricas para planetas y estrellas)
- Sistema de iluminación con múltiples fuentes de luz
- Animaciones orbitales y rotacionales
- Sistema de interacción mediante raycasting
- Interfaz de usuario con controles personalizables
- Panel informativo interactivo
- Diseño responsive y optimizado para rendimiento

2. HERRAMIENTAS Y TECNOLOGÍAS UTILIZADAS

2.1 Stack Tecnológico

Frontend:

- **HTML5**: Lenguaje de marcado para la estructura de la página web
- **CSS3**: Hojas de estilo en cascada para el diseño visual y layout
- **JavaScript ES6+**: Lenguaje de programación para la lógica de la aplicación

Librerías Principales:

- **Three.js versión r128**: Biblioteca JavaScript de código abierto para renderizado 3D
    - Utilizada para: creación de escenas 3D, manejo de cámaras, renderizado WebGL, geometrías, materiales y animaciones
    - CDN: `https://cdnjs.cloudflare.com/ajax/libs/three.js/r128/three.min.js`
- **OrbitControls**: Extensión de Three.js para control de navegación de cámara
    - CDN: `https://cdn.jsdelivr.net/npm/three@0.128.0/examples/js/controls/OrbitControls.js`

2.2 Conceptos de Computación Gráfica Aplicados

2.2.1 Modelado 3D

- **Geometrías primitivas**: Uso de `THREE.SphereGeometry` para crear planetas y el Sol
- **Transformaciones**: Aplicación de rotaciones, traslaciones y escalado mediante matrices de transformación
- **Jerarquía de objetos**: Organización de objetos en una escena jerárquica

2.2.2 Iluminación

- **Luz ambiental** (`THREE.AmbientLight`): Iluminación general uniforme en toda la escena
- **Luz puntual** (`THREE.PointLight`): Simulación del Sol como fuente de luz puntual
- **Sistema de sombras**: Implementación de sombras proyectadas utilizando `shadowMap`

2.2.3 Materiales y Texturización

- **Material Phong** (`THREE.MeshPhongMaterial`): Para planetas, proporciona iluminación basada en el modelo de Phong
    - Propiedades: color, shininess, specular
- **Material Basic** (`THREE.MeshBasicMaterial`): Para elementos que no requieren iluminación calculada
    - Utilizado en: Sol (efecto emisivo), órbitas, estrellas

2.2.4 Animación

- **Animación basada en tiempo**: Cálculo de posiciones planetarias usando funciones trigonométricas
- **Rotación continua**: Uso de `requestAnimationFrame` para loop de animación a 60 FPS
- **Interpolación**: Transiciones suaves mediante amortiguación (damping) en controles

2.2.5 Proyección y Cámara

- **Cámara perspectiva** (`THREE.PerspectiveCamera`): Proyección que simula la visión humana
    - Campo de visión: 75 grados
    - Planos de recorte: near (0.1) y far (10000)
- **Controles de órbita**: Sistema de navegación que permite rotación, zoom y pan

2.2.6 Detección de Colisiones

- **Raycasting**: Algoritmo para detectar intersecciones entre rayos (desde la cámara) y objetos 3D
- Utilizado para: detección de clics en planetas

2.2.7 Efectos Visuales

- **Partículas**: Sistema de estrellas de fondo usando `THREE.Points`
- **Transparencias**: Uso de materiales con opacidad para efectos visuales
- **Efectos de emisión**: Material emisivo para el Sol

3. DESCRIPCIÓN DE LAS INTERACCIONES IMPLEMENTADAS

3.1 Sistema de Navegación

3.1.1 Rotación de Cámara

- **Método**: Click izquierdo del mouse + arrastrar
- **Funcionamiento**: El sistema `OrbitControls` calcula la rotación de la cámara alrededor de un punto central basado en el movimiento del mouse
- **Características**:
    - Amortiguación habilitada para movimiento fluido
    - Restricciones de distancia (min: 30, max: 500 unidades)
    - Actualización continua en el loop de animación

3.1.2 Zoom

- **Método**: Rueda del mouse (scroll)
- **Funcionamiento**: Ajuste dinámico de la distancia entre la cámara y el centro de la escena
- **Límites**: Implementados para evitar zoom excesivo o alejamiento total

3.1.3 Pan (Desplazamiento Lateral)

- **Método**: Click derecho + arrastrar
- **Funcionamiento**: Movimiento paralelo de la cámara manteniendo su orientación

3.2 Interacción con Objetos

3.2.1 Selección por Clic

- **Tecnología**: Raycasting de Three.js
- **Proceso**:
    1. Conversión de coordenadas del mouse (pantalla) a coordenadas normalizadas (-1 a 1)
    2. Proyección de un rayo desde la cámara a través de las coordenadas del mouse
    3. Detección de intersecciones con objetos en la escena

4. Identificación del objeto más cercano intersectado

5. Activación de panel informativo con datos del planeta

3.2.2 Panel de Información

- **Trigger**: Clic en cualquier planeta
- **Contenido**: Nombre y descripción detallada del planeta
- **Interacción**: Botón de cierre para ocultar el panel
- **Diseño**: Panel flotante centrado con fondo semitransparente y efectos visuales

3.3 Controles Personalizables

3.3.1 Control de Velocidad de Animación

- **Tipo**: Slider (range input)
- **Rango**: 0x a 5x velocidad normal
- **Funcionamiento**: Modifica la variable `animationSpeed` que multiplica todas las velocidades de rotación orbital
- **Feedback visual**: Muestra el valor actual multiplicado (ej: "1.0x")

3.3.2 Toggle de Órbitas

- **Tipo**: Checkbox
- **Funcionalidad**: Activa/desactiva la visibilidad de las trayectorias orbitales (anillos visuales)
- **Aplicación**: Iteración sobre todos los objetos de órbita para cambiar propiedad `visible`

3.3.3 Toggle de Etiquetas

- **Tipo**: Checkbox
- **Funcionalidad**: Muestra/oculta las etiquetas con nombres de los planetas
- **Implementación**: Similar a las órbitas, modifica la propiedad `visible` de las etiquetas

3.4 Animaciones Automáticas

3.4.1 Rotación Orbital

- **Algoritmo**: Posición calculada mediante funciones trigonométricas

- **Fórmula**:

```
x = cos(ángulo) × distancia
z = sin(ángulo) × distancia
```

- **Velocidad**: Variable por planeta, ajustable mediante control de velocidad
- **Actualización**: Cada frame del loop de animación

3.4.2 Rotación Axial

- **Implementación**: Incremento continuo de `rotation.y` en cada frame
- **Velocidad**: Independiente de la velocidad orbital, proporcional a `animationSpeed`

3.4.3 Rotación de Anillos de Saturno

- **Especial**: Saturno incluye anillos que rotan independientemente
- **Efecto**: Rotación suave alrededor del eje Z para efecto visual dinámico

3.5 Cómo las Interacciones Mejoran la Experiencia del Usuario

1. **Exploración Libre**: Los controles de navegación permiten examinar la escena desde cualquier ángulo, facilitando la comprensión espacial
2. **Aprendizaje Activo**: La interacción por clic transforma la visualización en una herramienta educativa interactiva
3. **Personalización**: Los controles permiten adaptar la visualización a las preferencias del usuario
4. **Inmersión**: Las animaciones continuas crean una sensación de dinamismo y vida en la escena
5. **Accesibilidad**: La interfaz clara y los controles intuitivos hacen la aplicación accesible para usuarios de todos los niveles

4. DESCRIPCIÓN DE LOS ELEMENTOS VISUALES

4.1 Texturas y Materiales

4.1.1 Sol

- **Material**: `MeshBasicMaterial` con propiedades emisivas

- **Color base**: Amarillo (#ffff00)
- **Efecto emisivo**: Color emisivo amarillo con intensidad 0.5 para simular autoluminiscencia
- **Efecto de brillo**: Capa adicional semitransparente naranja alrededor del Sol para efecto de halo
- **Tamaño**: Radio de 5 unidades

4.1.2 Planetas

- **Material**: `MeshPhongMaterial` para iluminación realista
- **Propiedades**:
    - **Color**: Específico para cada planeta basado en características reales
        - Mercurio: Gris marrón (#8c7853)
        - Venus: Amarillo dorado (#ffc649)
        - Tierra: Azul (#6b93d6)
        - Marte: Rojo (#c1440e)
        - Júpiter: Beige (#d8ca9d)
        - Saturno: Dorado (#fad5a5)
        - Urano: Cian (#4fd0e7)
        - Neptuno: Azul oscuro (#4b70dd)
    - **Shininess**: 30 (brillo medio)
    - **Specular**: Gris oscuro (#222222) para reflejos sutiles
- **Sombras**: Habilitadas tanto para proyectar como para recibir sombras

4.1.3 Órbitas

- **Geometría**: `RingGeometry` (anillo 2D rotado)
- **Material**: `MeshBasicMaterial` semitransparente
- **Color**: Gris oscuro (#444444)
- **Opacidad**: 0.3 para efecto sutil
- **Ancho**: 0.2 unidades (configurable)

4.1.4 Anillos de Saturno

- **Geometría**: `RingGeometry` con radio interno 2.4 y externo 3.2
- **Material**: `MeshPhongMaterial` semitransparente
- **Color**: Dorado (#c9a961)
- **Opacidad**: 0.7
- **Rotación**: Continua alrededor del eje Z

4.1.5 Estrellas de Fondo

- **Geometría**: `BufferGeometry` con 10,000 vértices
- **Material**: `PointsMaterial`
- **Color**: Blanco (#ffffff)
- **Tamaño**: 0.5 unidades por punto
- **Distribución**: Aleatoria en un espacio cúbico de 2000×2000×2000 unidades

4.2 Iluminación

4.2.1 Luz Ambiental

- **Tipo**: `AmbientLight`
- **Color**: Blanco (#404040)
- **Intensidad**: 0.5
- **Propósito**: Proporcionar iluminación base para evitar sombras completamente negras

4.2.2 Luz del Sol (Puntual)

- **Tipo**: `PointLight`
- **Color**: Blanco (#ffffff)
- **Intensidad**: 2.0
- **Distancia**: 2000 unidades (rango máximo)
- **Posición**: Centro de la escena (0, 0, 0), coincidiendo con el Sol
- **Sombras**: Habilitadas con `castShadow = true`

- **Tipo de sombras**: `PCFSoftShadowMap` para sombras suaves y realistas

4.3 Escala y Proporciones

4.3.1 Tamaños Relativos

Los tamaños de los planetas están ajustados para mantener proporciones visibles:

- Sol: 5 unidades
- Mercurio: 0.8 unidades
- Venus: 0.95 unidades
- Tierra: 1.0 unidad (referencia)
- Marte: 0.75 unidades
- Júpiter: 2.5 unidades
- Saturno: 2.2 unidades (sin incluir anillos)
- Urano: 1.8 unidades
- Neptuno: 1.7 unidades

Nota: Las proporciones no son exactas a escala real debido a limitaciones de visualización (el Sol real sería aproximadamente 109 veces el diámetro de la Tierra).

4.3.2 Distancias Orbitales

Las distancias están comprimidas para mantener visibilidad:

- Mercurio: 15 unidades
- Venus: 22 unidades
- Tierra: 30 unidades
- Marte: 45 unidades
- Júpiter: 65 unidades
- Saturno: 85 unidades
- Urano: 105 unidades
- Neptuno: 125 unidades

Nota: En realidad, estas distancias serían astronómicamente mayores (por ejemplo, Neptuno está a aproximadamente 30 veces la distancia Tierra-Sol).

4.4 Animaciones Visuales

4.4.1 Movimiento Orbital

- **Tipo**: Circular (simplificado, no elíptico como en la realidad)
- **Velocidad angular**: Variable por planeta, proporcional a su distancia (planetas más cercanos giran más rápido)
- **Dirección**: Todas las órbitas en el mismo plano (plano XZ)
- **Visualización**: A través de anillos de órbita y movimiento continuo de los planetas

4.4.2 Rotación Axial

- **Velocidad**: Constante para todos los planetas (0.01 radianes por frame × animationSpeed)
- **Eje**: Rotación alrededor del eje Y (vertical)
- **Efecto**: Los planetas giran sobre sí mismos mientras orbitan

4.4.3 Efectos Continuos

- **Actualización**: 60 veces por segundo (60 FPS) mediante `requestAnimationFrame`
- **Sincronización**: Todas las animaciones están sincronizadas con el ciclo de renderizado del navegador

4.5 Interfaz de Usuario

4.5.1 Panel de Controles

- **Posición**: Superior derecha
- **Diseño**: Fondo semitransparente oscuro con borde azul
- **Contenido**: Sliders, checkboxes e instrucciones
- **Estilo**: Moderno con efectos de blur y sombras

4.5.2 Panel de Información

- **Diseño**: Modal centrado con fondo oscuro semitransparente
- **Animación**: Transición de opacidad al mostrar/ocultar

- **Tipografía**: Jerarquía clara con títulos destacados

4.5.3 Título

- **Posición**: Superior izquierda
- **Estilo**: Coherente con el resto de la interfaz

4.5.4 Etiquetas de Planetas

- **Tipo**: HTML overlay posicionado dinámicamente
- **Estilo**: Fondo semitransparente, borde azul, sombra de texto
- **Posición**: Por encima de cada planeta, siguiendo su movimiento

5. REFLEXIÓN CRÍTICA

Este proyecto me ayudó bastante a entender cómo funciona la computación gráfica cuando la aplicas en la práctica. Al principio estaba perdido porque nunca había trabajado con Three.js, tuve que ver varios tutoriales en YouTube y leer bastante documentación para poder empezar. Después de mucho ensayo y error logré hacer que funcione aunque al principio no salía nada.

Lo más difícil fue encontrar el equilibrio entre que se viera bien y que no se trabara todo. Al principio intenté poner texturas reales de los planetas que descargué de internet, pero mi laptop empezaba a laggear mucho y se veía horrible. Tuve que simplificar todo y usar solo colores sólidos, que aunque no es tan realista al menos funciona bien y se ve decente. También me di cuenta que es super importante optimizar el código, sobre todo cuando tienes animaciones corriendo constantemente como en este caso.

Hacer que funcionara el click en los planetas (lo del raycasting) fue lo que más me costó. No entendía bien cómo funcionaba, tuve que buscar en foros y ver ejemplos hasta que finalmente logré hacerlo. Cuando por fin pude clickear un planeta y que apareciera la información me sentí super bien, fue como un logro personal jaja.

Aprendí bastante sobre iluminación también. No sabía que había diferentes tipos de luz, pensaba que era solo poner una y ya. La luz ambiental ayuda a que no se vea todo muy oscuro, y la puntual simula el sol. El material Phong se ve mucho mejor que el Basic aunque consume más recursos, tuve que balancear eso.

Las transformaciones 3D eran cosas que había visto en las clases teóricas pero nunca las había aplicado. Ver cómo funcionan el coseno y seno para hacer las órbitas fue interesante, aunque confieso que al principio me costó entender por qué funcionaba así. También aprendí sobre requestAnimationFrame que es la forma correcta de hacer animaciones, antes pensaba que con setInterval era suficiente pero no es lo mismo.

En general, este proyecto me enseñó que hacer gráficos 3D no es solo hacer que se vea bonito, tienes que pensar en muchas cosas al mismo tiempo: que funcione bien, que se vea decente, que no consuma muchos recursos. Fue un reto pero siento que aprendí mucho haciendo esto, aunque al final algunas cosas quedaron más simples de lo que quería pero funciona bien.

Anexo A. Bitácora personal de desarrollo

Pequeño registro de cómo fui resolviendo cosas mientras armaba el proyecto.

- **Día 1**: Elegí el tema (sistema solar). Abrí un lienzo básico y no se veía nada; resultó que me faltaba crear la cámara y llamar a `animate()`.
- **Día 2**: Agregué el Sol y los primeros planetas. Júpiter se veía enano; ajusté tamaños y mejoró. Probé texturas, pero el rendimiento bajó, así que volví a colores sólidos.
- **Día 3**: Raycasting. No funcionaba porque estaba escuchando clics en el elemento equivocado. Lo moví al canvas y ya pude detectar intersecciones.
- **Día 4**: Etiquetas con HTML overlay. CSS3D me dio problemas, así que lo resolví proyectando posiciones 3D a 2D y posicionando `
  `s.
- **Día 5**: Panel de controles y limpieza. Ajusté velocidad, órbitas, nombres y detalles visuales. Exporté la documentación en PDF (tuve que desactivar “encabezados y pies de página” en la impresión).

Anexo B. Lecciones rápidas

- Empieza simple y agrega detalles solo si el rendimiento lo permite.
- El raycasting es más fácil de lo que parece cuando entiendes las coordenadas normalizadas.
- Los materiales Phong se ven mejor, pero hay que usarlos con cabeza.
- `requestAnimationFrame` se siente más fluido que cualquier temporizador.
- La depuración con mensajes cortos en consola ahorra tiempo (pero bórralos al final).

Referencias

Three.js. (2020). *Three.js - JavaScript 3D library*. https://threejs.org/

Mozilla Developer Network. (2024). *WebGL*. MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API

Shirley, P., & Marshner, S. (2019). *Fundamentals of Computer Graphics* (5th ed.). CRC Press.