

Manual técnico



Castillo de Peach

PROFESOR: CARLOS ALDAIR ROMAN BALBUENA
317222049-MENDOZA ANAYA ALDAIR ISRAEL
GRUPO: 06

1-INTRODUCCIÓN

El proyecto tiene como objetivo recrear el castillo de Peach en un entorno virtual utilizando el framework OpenGL. El propósito principal del proyecto es agrupar los elementos integrales de aprendizaje de la materia de CGEIH (Computación Gráfica y Elementos Interactivos en Humanidades y Ciencias) y demostrar el uso eficiente de habilidades prácticas y teóricas.

Objetivos del proyecto:

1. Planeación y desarrollo: Realizar la planificación y desarrollo del proyecto de acuerdo con los requisitos establecidos.
 - Seleccionar una fachada y un espacio que pueden ser reales o ficticios como referencia para la recreación 3D en OpenGL.
 - Presentar imágenes de referencia de los espacios seleccionados, mostrando los objetos que se recrearán virtualmente.
 - Los objetos deben ser lo más parecidos posible a su imagen de referencia, tanto en su apariencia como en su ambientación.
2. Manual de usuario: Crear un manual de usuario en formato PDF que proporcione instrucciones claras sobre cómo utilizar el entorno virtual del castillo de Peach. El manual debe contener capturas de pantalla que ilustren cada elemento explicado.
3. Manual técnico: Preparar un manual técnico que incluya el cronograma de actividades realizado para la conclusión del proyecto, junto con evidencias de las herramientas colaborativas utilizadas para el desarrollo del software.
 - Mostrar evidencias del uso de plataformas colaborativas de software, como Jira, Trello, Git o Github, con especificaciones de avance en los repositorios.

Los objetivos específicos del proyecto de recrear el castillo de Peach en un entorno virtual son los siguientes:

1. Capturar la apariencia del castillo de Peach: El objetivo principal es crear modelos 3D precisos y detallados que representen fielmente la arquitectura, los elementos estructurales y los detalles visuales del castillo de Peach. Esto implica recrear las formas, tamaños, colores y texturas de cada elemento del castillo de manera realista.
2. Capturar la atmósfera del castillo: El castillo de Peach tiene una atmósfera distintiva y encantadora que se refleja en su diseño, iluminación y ambientación. El objetivo es transmitir esa sensación de magia y encanto en el entorno virtual recreado. Esto puede incluir la iluminación adecuada para resaltar áreas específicas del castillo, el uso de efectos visuales y de partículas para crear una atmósfera mágica y el diseño de paisajes o elementos circundantes que complementen la experiencia.
3. Recrear los objetos presentes en el castillo: Además del castillo en sí, existen otros elementos icónicos y reconocibles dentro del mundo de Peach, como el trono, las ventanas con cortinas, las banderas, las escaleras y otros objetos decorativos. El objetivo es recrear virtualmente estos objetos y detalles presentes en el castillo de Peach con la mayor fidelidad posible, prestando atención a sus proporciones, colores y características distintivas.
4. Lograr interacción y exploración: El proyecto también puede incluir la posibilidad de interactuar con el entorno virtual del castillo de Peach. Esto implica permitir a los usuarios explorar diferentes áreas del castillo, interactuar con objetos, abrir puertas, activar mecanismos y descubrir sorpresas ocultas. El objetivo es proporcionar una experiencia inmersiva y entretenida para los usuarios.

Propuesta del proyecto planteada

Fachada a recrear:

El objetivo es recrear la fachada del castillo de Peach, tomando como referencia el juego Super Mario 3D Land. El castillo será representado como un edificio cuadrado con cuatro torres en las esquinas y una torre más grande en la cima. Estará ubicado en un rectángulo que simulará una pequeña colina. Para crear un entorno realista, se añadirán aproximadamente 12 árboles alrededor del castillo.

Objetos a recrear:

Además de la fachada del castillo y su interior, se recrearán varios objetos y personajes icónicos del juego. Estos incluyen:

- Toads: Se modelarán y texturizarán los personajes Toad, los fieles seguidores de Peach. Estos personajes estarán distribuidos en el entorno del castillo, agregando vida y movimiento al escenario.
- Goomba: También se recrearán los enemigos clásicos de Super Mario, los Goombas. Este pequeño personaje estará presente en el entorno, caminando en frente del castillo.
- Bloque con interrogación: Se modelará y texturizará el icónico bloque con signo de interrogación. Este bloque es conocido por contener monedas, power-ups y otros premios. Estará colocado estratégicamente dentro del entorno para que los usuarios puedan interactuar con él.
- Trono: Se recreará un trono de madera, oro y terciopelo donde se colocará en un lugar destacado dentro del castillo.
- Florero: Se modelará y texturizará un florero decorativo que se encontrará en una de las áreas interiores del castillo.
- Flor de fuego: Se modelará y texturizará la icónica Flor de Fuego, un power-up especial en la serie de Super Mario, solo que se decidió tomar la versión del Super smash Bros Melee. Esta flor estará disponible en el entorno exterior para que los usuarios puedan contemplar su movimiento poco apreciado en los juegos.
- Kart: Se modelará y texturizará un kart, el vehículo utilizado por Mario y sus amigos en las carreras de Mario Kart. Este objeto estará presente en el entorno, agregando un toque divertido y juguetón en una ruta a las afueras del castillo. Además de otro ubicado inmóvil en el castillo para su contemplación más detallada.

Esta propuesta busca recrear de manera fiel los elementos y la atmósfera del castillo de Peach, proporcionando una experiencia inmersiva y reconocible para los fanáticos de Super Mario.

2-PLATAFORMAS Y HERRAMIENTAS UTILIZADAS

Para el desarrollo del ambiente del castillo, se utilizaron varias plataformas de trabajo que permitieron la implementación del proyecto. A continuación, se describen las plataformas principales utilizadas:

- Microsoft Visual Studio:

Se utilizó el entorno de desarrollo integrado de Microsoft Visual Studio como la herramienta principal para desarrollar programas informáticos y aplicaciones. Este IDE ofrece una amplia gama de funcionalidades y herramientas que facilitan la programación y la manipulación de gráficos e imágenes.

- OpenGL:

Se hizo uso de OpenGL, que es una API (interfaz de programación de aplicaciones) utilizada para manipular gráficos e imágenes. Aunque OpenGL es una especificación desarrollada y mantenida por el Grupo Khronos, es necesario implementar estas especificaciones para que las funciones operen correctamente. En el proyecto, se trabajó con las bibliotecas de OpenGL escritas en C, que permiten su uso en otros lenguajes de programación.

- Maya:

Maya fue utilizado como software de modelado principal en el proyecto. Es una aplicación de modelado 3D ampliamente utilizada que permite crear y editar modelos tridimensionales con gran detalle y realismo. Con Maya, se pudo diseñar y dar forma a los elementos del castillo y los objetos/personajes que pertenecen a este universo.

- IbisPaintX:

IbisPaintX fue utilizado como software de edición de texturas en el proyecto. Es una aplicación de pintura digital que permite crear y modificar texturas con una amplia gama de herramientas y efectos. Con IbisPaintX, se logró dar vida a los modelos del mundo de Mario mediante la aplicación de texturas detalladas.

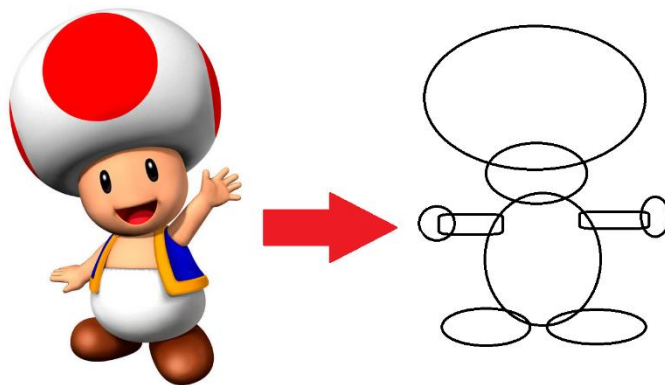
- Models Resource:

La página web Models Resource fue una fuente importante para obtener modelos y texturas utilizados en el proyecto. Models Resource es una plataforma en línea que ofrece una amplia colección de recursos, como modelos 3D y texturas, que pueden ser descargados y utilizados en proyectos de diseño y desarrollo. Gracias a Models Resource, se pudo acceder a una variedad de modelos y texturas de alta calidad para enriquecer el entorno del castillo.

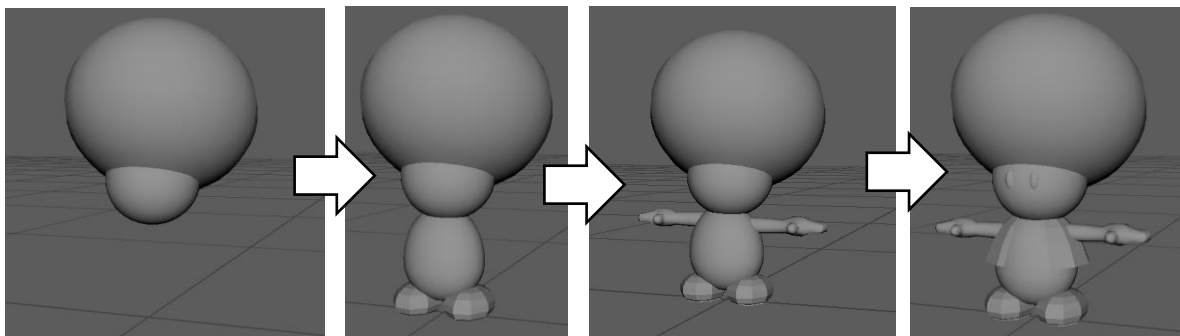
Al emplear estas plataformas y programas en el proyecto del castillo de Peach, se logró una implementación efectiva y detallada, permitiendo modelar los elementos del mundo de Mario con precisión, aplicar texturas y enriquecer el entorno con recursos disponibles en Models Resource.

3-Proceso de Modelaje 3D

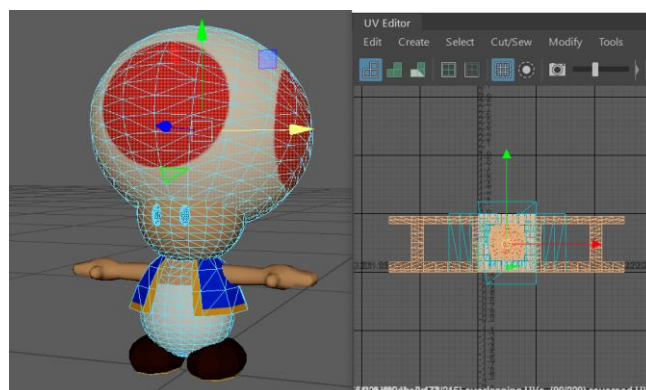
El modelado en Maya es un proceso creativo y técnico que permite dar vida a ideas y conceptos en forma de modelos 3D. Comienza con la planificación y conceptualización de los objetos que se desean crear. Esto implica definir la forma básica, las proporciones y los detalles específicos de cada componente que conformaran al objeto.



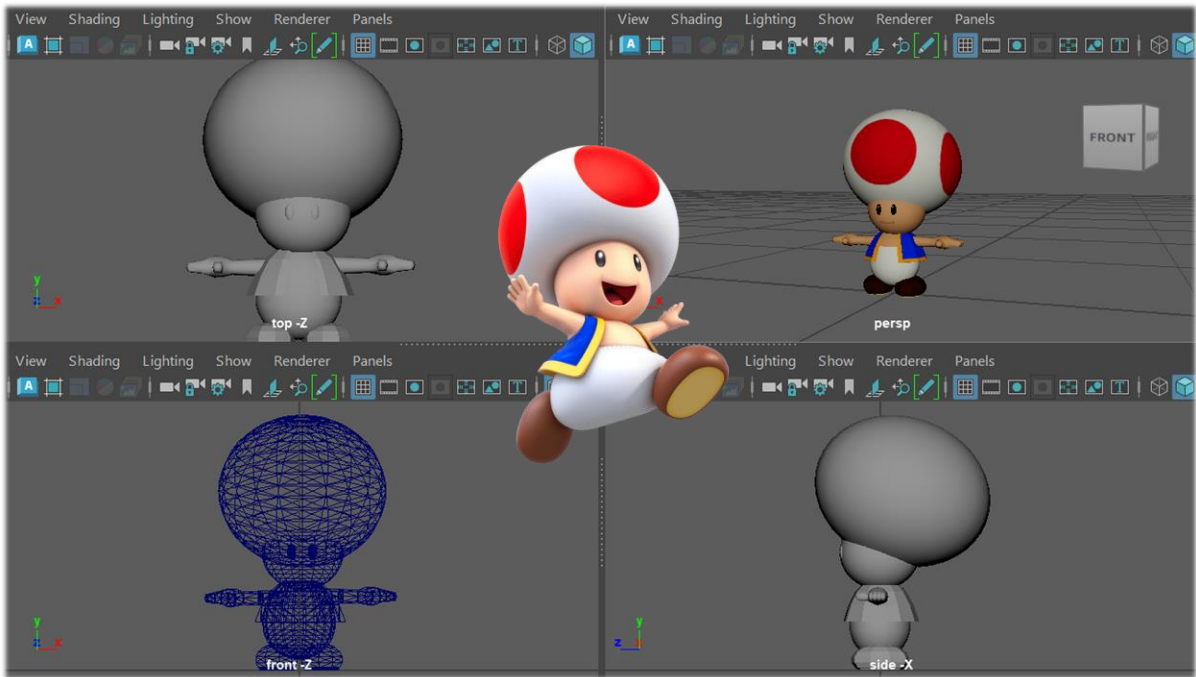
Una vez que se establecen las bases del diseño, se procede a crear la geometría utilizando las herramientas de modelado disponibles en Maya. Durante el proceso de modelado, se aplicaron técnicas de subdivisión de superficies, la extrusión, la suavización y transformación mediante caras, aristas y vértices para los detalles del objeto.



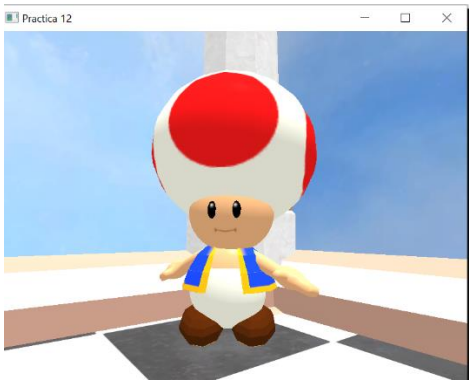
Buscamos después escoger la imagen correcta para texturizar al objeto y darle al objeto fidelidad al diseño seleccionado, así trabajamos con el mapeo de la textura para que cada cara se este pintando de la mejor manera posible.



Una vez terminado con el objeto, se analiza desde todas las perspectivas posible y con las tres vistas disponibles, para determinar si se puede optimizar o con el objeto que se tiene es asemeja de forma más fiel al objeto/personaje deseado. Hubo casos en los que decidí eliminar caras del objeto que no se apreciaban desde afuera, haciendo que sea más fácil de cargar.




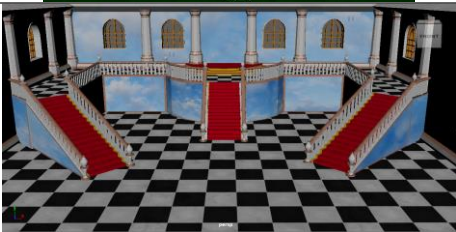


Finalmente integramos el modelo a nuestro escenario dentro de Visual Studio, comprobando que el modelo generado y/o descargado combine con el ambiente que estamos generando.





4-GALERÍA Y DETALLES DE MODELOS EN MAYA



Castillo de Peach

Imagen referencia	Modelo recreado en Maya	Numero de triángulos
		42, 655 triángulos
		

Toad

Imagen referencia	Modelo recreado en Maya	Numero de triángulos
		3, 400 triángulos

Goomba

Imagen referencia	Modelo recreado en Maya	Numero de triángulos
		1, 669 triángulos

Question Block

Imagen referencia	Modelo recreado en Maya	Numero de triángulos
		44 Triángulos



Trono

Imagen referencia	Modelo recreado en Maya	Numero de triángulos
		4,518 Triángulos

Florero

Imagen referencia	Modelo recreado en Maya	Numero de triángulos
		192 Triángulos

Flor






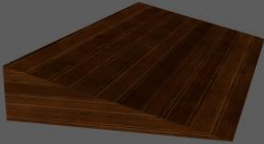


Imagen referencia	Modelo recreado en Maya	Numero de triángulos
		192 Triángulos

Kart


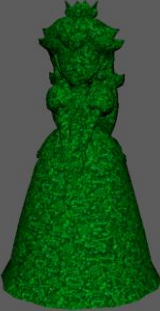







Imagen referencia	Modelo recreado en Maya	Numero de triángulos
		4, 036 Triángulos

4.1 MODELOS ADICIONALES USADOS

4.1.1 Modelados

Modelo	Modelo en Maya	Numero de triángulos por modelo
Arbusto con flores		1,848 triángulos
Pintura		98 triángulos
Marco		32 triángulos
Estrella		796 triángulos
Question Block vacío		44 triángulos
Rampa		8 triángulos
Ventanas		372 triángulos (186 cada mitad)
Vidrio translucido		16 triángulos

4.1.2 Descargados

Modelo	Modelo en Maya	Numero de triángulos por modelo	Re texturizado
Árbol		1,340 triángulos	SI
Arbusto con forma de Peach		3,419 triángulos	SI
Champiñón		466 triángulos	NO
Fly Guy		2,316 triángulos	NO
Paracaídas		410 triángulos	NO
Peach		3,755 triángulos (1,888 sin el kart)	Solo el kart cambio de color
Ventanas		3,828 triángulos	NO
Paratroopa		3,416 triángulos	NO
Puertas		3,888 triángulos	SI

El entorno virtual del castillo de Peach está utilizando una cantidad de 116,839 de triángulos en su composición.

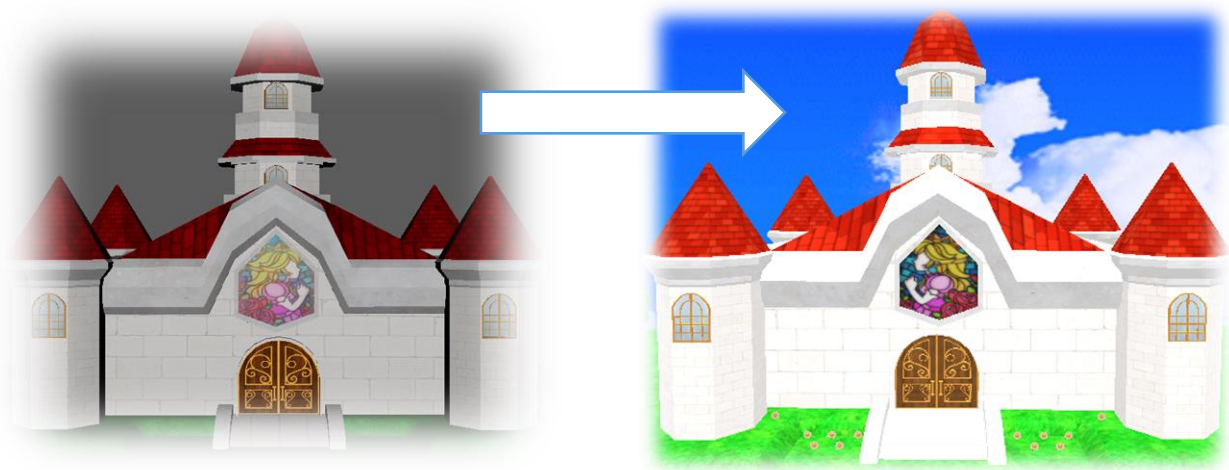
5-ILUMINACION

Mediante la configuración de la luz global, se buscó generar una iluminación que resaltara los colores y detalles del castillo, creando una atmósfera alegre y festiva. Se experimentó con diferentes intensidades y tonalidades de luz para lograr el efecto deseado, asegurándose de que la iluminación resaltara los detalles arquitectónicos y resaltara los colores vivos presentes en el castillo de Peach.

```
// Directional light
glUniform3f(glGetUniformLocation(lightningShader.Program, "dirLight.direction"), 0.0f, -0.5f, -0.1f);
glUniform3f(glGetUniformLocation(lightningShader.Program, "dirLight.ambient"), 0.9f, 0.9f, 0.9f);
glUniform3f(glGetUniformLocation(lightningShader.Program, "dirLight.diffuse"), 0.7f, 0.7f, 0.7f);
glUniform3f(glGetUniformLocation(lightningShader.Program, "dirLight.specular"), 0.5f, 0.5f, 0.5f);
```

Al ajustar los parámetros de la luz global, se logró crear una iluminación que brinda una sensación de calidez y alegría, aportando un aspecto visual atractivo y vibrante al entorno virtual del castillo.

Es importante destacar que la elección de una iluminación colorida y alegre tiene como objetivo principal enriquecer la experiencia del usuario y sumergirlo en un ambiente mágico y encantador, acorde con el estilo y la temática del castillo de Peach.



6-ANIMACIONES

Durante el proceso de animación en OpenGL, aprovecharemos las siguientes técnicas y herramientas:

- Transformaciones geométricas: Utilizaremos matrices de transformación para lograr movimientos de traslación, rotación y escalado de los objetos en el espacio. Estas transformaciones se aplicarán a los vértices de los modelos para lograr cambios en su posición y forma.
- Control del tiempo: Para controlar la velocidad y el ritmo de la animación, implementaremos mecanismos para aumentar o disminuir valores que afectaran al tiempo transcurrido entre fotogramas. Esto nos permitirá crear animaciones en tiempo real y ajustar la duración y velocidad de los movimientos según nuestras necesidades.
- Funciones matemáticas: Al incorporar funciones matemáticas en la animación en OpenGL, podremos ampliar nuestras posibilidades creativas y lograr resultados más interesantes y dinámicos.

Las capturas de las animaciones se encontrarán al final del documento en la sección **“Resultados y ejemplos visuales”**.

6.1 Animaciones sencillas

- **Puertas y ventanas:**

La animación de las puertas y ventanas del castillo de Peach agrega un elemento de interactividad cuando se presiona la tecla P o V. Las puertas realizan un movimiento de rotación en el eje Y, comenzando desde una posición cerrada de 0 grados hasta una posición abierta de 90 grados hacia el interior del castillo. Por otro lado, las ventanas también se mueven en un movimiento de rotación en el eje Y, pero con un rango más amplio, desde 0 grados hasta 120 grados y hacia afuera. Ambas animaciones se activarán con un booleano que determina cuando rotar y dicha animación no se podrá interrumpir hasta que termine de rotar hacia adentro o hacia afuera.

```
//Ventanas//////////////////////////////////////Puerta////////////////////////////////////
if (keys[GLFW_KEY_V]) if (keys[GLFW_KEY_P])
{
    AnimVentana = true;
    AnimPuerta = true;
}
if (AnimVentana) if (AnimPuerta)
{
    if (Ventana) if (Puerta)
    {
        if (RotVentana < 120) if (RotPuerta < 90)
        {
            RotVentana += 2; RotPuerta += 0.2;
        }
        else
        {
            Ventana = false; Puerta = false;
            AnimVentana = false; AnimPuerta = false;
        }
    }
    if (!Ventana) if (!Puerta)
    {
        if (RotVentana > 0) if (RotPuerta > 0)
        {
            RotVentana -= 2; RotPuerta -= 0.2;
        }
        else
        {
            Ventana = true; Puerta = true;
            AnimVentana = false; AnimPuerta = false;
        }
    }
}

model = glm::translate(model, glm::vec3(-0.428f, 12.109f, 1.559f));
model = glm::rotate(model, glm::radians(-RotVentana), glm::vec3(0.0f, 1.0f, 0.0f));
glUniform4f(glGetUniformLocation(lightningShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Ventana1.Draw(lightningShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.446f, 12.109f, 1.559f));
model = glm::rotate(model, glm::radians(RotVentana), glm::vec3(0.0f, 1.0f, 0.0f));
glUniform4f(glGetUniformLocation(lightningShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Ventana2.Draw(lightningShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(1.362f, 1.417f, 5.059f));
model = glm::rotate(model, glm::radians(-RotPuerta), glm::vec3(0.0f, 1.0f, 0.0f));
glUniform4f(glGetUniformLocation(lightningShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Puerta2.Draw(lightningShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-1.495f, 1.417f, 5.059f));
model = glm::rotate(model, glm::radians(RotPuerta), glm::vec3(0.0f, 1.0f, 0.0f));
glUniform4f(glGetUniformLocation(lightningShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Puerta1.Draw(lightningShader);
```

Para lograr una sensación de realismo y peso en la animación de la puerta, se ha ajustado la velocidad de apertura de las ventanas para que sean más rápidas que la puerta. Esto crea un contraste visual interesante y resalta la diferencia en tamaño y material entre las puertas y las ventanas. La intención es transmitir una sensación de solidez y robustez en la puerta, mientras que las ventanas brindan una apertura más ligera y ágil.

- **Toads brincando:**

La animación de los Toads saltando agrega un elemento de actividad al entorno del castillo de Peach. En esta animación, dos Toads estarán dando saltitos mientras suben y bajan levemente sus brazos en el eje Z. Para lograr este efecto, se utilizan dos variables: una variable "IDLE" y otra variable "

RotBrazo". Estas variables se incrementan y decrementan a diferentes velocidades, lo que crea un movimiento armónico y sincronizado de los Toads.

La variable "IDLE" controla el movimiento de salto, haciendo que los Toads suban y bajen en el eje Y. Al aumentar y disminuir esta variable, se logra el efecto de saltitos continuos y rítmicos. Por otro lado, la variable " RotBrazo" controla el movimiento de los brazos de los Toads en el eje Z. A medida que esta variable cambia, los brazos de los Toads se mueven hacia arriba y hacia abajo de manera suave y coordinada. El uso de diferentes velocidades para las variables "IDLE" y " RotBrazo" añade dinamismo y naturalidad a la animación, evitando movimientos repetitivos y monótonos.

```
//Toad////////////////////////////////////
if (sentidoIDLE)
{
    IDLE += 0.0006;
    if (IDLE > 0.025)
    {
        sentidoIDLE = false;
    }
}
if (!sentidoIDLE)
{
    IDLE -= 0.0006;
    if (IDLE < 0)
    {
        sentidoIDLE = true;
    }
}
if (sentidoBrazo)
{
    RotBrazo += 0.4;
    if (RotBrazo > 20)
    {
        sentidoBrazo = false;
    }
}
if (!sentidoBrazo)
{
    RotBrazo -= 0.4;
    if (RotBrazo < 0)
    {
        sentidoBrazo = true;
    }
}

//Toad
model = glm::mat4(1);
modelaux = model;
model = glm::translate(model, glm::vec3(-1.247f, 0.609f + (IDLE * 1.7), -0.619f));
model = glm::rotate(model, glm::radians(38.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniform4f(glGetUniformLocation(lightningShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
CuerpoToad.Draw(lightningShader);
modelaux = model;
model = modelaux;
model = glm::translate(model, glm::vec3(-0.044f, 0.175f, 0.0f));
model = glm::rotate(model, glm::radians(RotBrazo), glm::vec3(0.0f, 0.0f, 1.0f));
glUniform4f(glGetUniformLocation(lightningShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
BrazoDer.Draw(lightningShader);
model = modelaux;
model = glm::translate(model, glm::vec3(0.044f, 0.175f, 0.0f));
model = glm::rotate(model, glm::radians(-RotBrazo), glm::vec3(0.0f, 0.0f, 1.0f));
glUniform4f(glGetUniformLocation(lightningShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
BrazoIzq.Draw(lightningShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-1.247f, 0.576f + IDLE, -0.619f));
model = glm::rotate(model, glm::radians(38.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(IDLE * 800), glm::vec3(1.0f, 0.0f, 0.0f));
glUniform4f(glGetUniformLocation(lightningShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Zapatos.Draw(lightningShader);
```

Para evitar hacer transformaciones combinadas manualmente se opto por hacer con jerarquía al Toad, así el torso recibe el brinco y lo transmite a los brazos, mientras que los brazos hacen así el traslado en Y y la rotación en Z.

- **Goomba caminando**

La animación del Goomba caminando agrega un toque de movimiento y personalidad al entorno del castillo de Peach. En esta animación, el Goomba se desplaza de lado a lado en el eje X mientras rota su cabeza y levanta levemente sus pies en el eje Z, siguiendo su característica caminata. Para lograr este efecto, se utilizan incrementos y decrementos en variables específicas. Estas variables controlan el desplazamiento horizontal del Goomba en el eje X, la rotación de su cabeza y el movimiento de sus pies en el eje Z que son controladas con booleanos. Se añade un detalle adicional para asegurar que los pies del personaje parezcan pisar firmemente el piso y no atravesarlo. Para lograr esto, se verifica la posición vertical de los pies del Goomba durante su movimiento en el eje Z, cuando la rotación del pie hace que su posición en el eje Z supere el nivel del piso, se establece su valor en 0. Esto garantiza que el pie del Goomba esté en contacto con el suelo y no atravesase la superficie.

```
//Goomba
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(2.992f + CaminataGoomba, -0.041f, 12.026f));
model = glm::rotate(model, glm::radians(rotCabezaGoomba), glm::vec3(0.0f, 0.0f, 1.0f));
glUniform4f(glGetUniformLocation(lightningShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
GoombaCabeza.Draw(lightningShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(2.992f + CaminataGoomba, -0.041f, 12.026f));
model = glm::rotate(model, glm::radians(2 * rotPieDerGoomba), glm::vec3(0.0f, 0.0f, 1.0f));
glUniform4f(glGetUniformLocation(lightningShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
GoombaPieDer.Draw(lightningShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(2.992f + CaminataGoomba, -0.041f, 12.026f));
model = glm::rotate(model, glm::radians(2 * rotPieIzqGoomba), glm::vec3(0.0f, 0.0f, 1.0f));
glUniform4f(glGetUniformLocation(lightningShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
GoombaPieIzq.Draw(lightningShader);
```

```

if (SentidoGoomba == false)
{
    CaminataGoomba += 0.002f;
    if (CaminataGoomba > 0.647f)
    {
        SentidoGoomba = true;
    }
}

if (SentidoGoomba == true)
{
    CaminataGoomba -= 0.002f;
    if (CaminataGoomba < -0.936f)
    {
        SentidoGoomba = false;
    }
}

if (RotSentidoGoomba == false)
{
    rotCabezaGoomba += 0.3f;
    if (rotCabezaGoomba < 0)
    {
        rotPieDerGoomba = rotCabezaGoomba;
    }
    if (rotCabezaGoomba > 0)
    {
        rotPieIzqGoomba = rotCabezaGoomba;
    }
    if (rotCabezaGoomba > 10.0f)
    {
        RotSentidoGoomba = true;
    }
}

if (RotSentidoGoomba == true)
{
    rotCabezaGoomba -= 0.3f;
    if (rotCabezaGoomba < 0)
    {
        rotPieDerGoomba = rotCabezaGoomba;
    }
    if (rotCabezaGoomba > 0)
    {
        rotPieIzqGoomba = rotCabezaGoomba;
    }
    if (rotCabezaGoomba < -10.0f)
    {
        RotSentidoGoomba = false;
    }
}

```

- **Flores bailando:**

La animación de las flores de fuego de Mario agrega un toque divertido y dinámico al entorno del castillo de Peach. Estas flores realizan un movimiento de rotación de lado a lado en el eje Y, mientras que simultáneamente aumentan su posición en el eje Y cuando su rotación es 0 grados, creando un efecto de saltitos.

```

//Flor////////////////////////////////////////
if (FlorSentido)
{
    FlorGiro += 4;
    if (FlorGiro > 99)
    {
        FlorSentido = !FlorSentido;
    }
}
if (!FlorSentido)
{
    FlorGiro -= 4;
    if (FlorGiro < -99)
    {
        FlorSentido = !FlorSentido;
    }
}

//Flores
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5.350f, 0.480f + FlorBrinco, 6.050f));
model = glm::rotate(model, glm::radians(-3 * FlorGiro / 20), glm::vec3(0.0f, 0.0f, 1.0f));
glUniform4f(glGetUniformLocation(lightningShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Flor.Draw(lightningShader);

if (FlorGiro > 0)
{
    FlorBrinco = 0.025 * (100 - FlorGiro) / 100;
}
if (FlorGiro < 0)
{
    FlorBrinco = 0.025 * (100 + FlorGiro) / 100;
}

```

- **Fly Guy volando de un lado a otro**

Para la animación del FlyGuy, se crea un movimiento lateral en el eje X mientras el personaje se inclina hacia el lado al que se está desplazando. El movimiento lateral se logra modificando la posición en el eje X del FlyGuy, desplazándolo de un lado a otro, siendo controlada la dirección con un booleano. Al mismo tiempo, se aplica una inclinación hacia el lado correspondiente utilizando una rotación en el eje Y. Esta combinación de movimiento lateral e inclinación crea la sensación de que el FlyGuy se desplaza de manera dinámica y con fluidez, mientras mantiene la vista hacia un costado del castillo.

La hélice en la cabeza del FlyGuy se anima mediante una variable reciclada que controla su rotación. Esta variable se incrementa gradualmente de 0 a 360 grados, permitiendo que la hélice realice una rotación completa. Una vez que alcanza los 360 grados, se reinicia el proceso, creando un efecto de rotación continua y constante. Adicionalmente, se puede agregar un movimiento leve de vuelo arriba y abajo utilizando un auxiliar. Esta variable auxiliar se suma o resta a la posición en el eje Y del FlyGuy, generando un suave movimiento ascendente y descendente que simula su vuelo.

```

//Flyguy
if (SentidoFlyGuy)
{
    TrasladoFlyguy += 0.01;
    if (RoatFlyGuy < 20.0)
    {
        RoatFlyGuy += 0.5;
    }
    if (TrasladoFlyguy > 2)
    {
        SentidoFlyGuy = false;
    }
}
if (!SentidoFlyGuy)
{
    TrasladoFlyguy -= 0.01;
    if (RoatFlyGuy > -20.0)
    {
        RoatFlyGuy -= 0.5;
    }
    if (TrasladoFlyguy < -2)
    {
        SentidoFlyGuy = true;
    }
}

//FlyGuy
model = glm::mat4(1);
modelaux = model;
model = glm::translate(model, glm::vec3(8.162f, 2.224f + (auxilar1 / 2000), 0.0f + TrasladoFlyguy));
model = glm::rotate(model, glm::radians(-20.0f), glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::rotate(model, glm::radians(RoatFlyGuy), glm::vec3(1.0f, 0.0f, 0.0f));
glUniform4f(glGetUniformLocation(lightningShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
FlyGuy.Draw(lightningShader);
modelaux = model;
model = modelaux;
model = glm::rotate(model, glm::radians(Rotacion2*10), glm::vec3(0.0f, 1.0f, 0.0f));
glUniform4f(glGetUniformLocation(lightningShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Helices.Draw(lightningShader);

```

- **Estrella rotando**

Para la animación de la estrella, se utiliza la misma variable auxiliar mencionada anteriormente para lograr un movimiento leve de ascenso y descenso. Esta variable se suma o resta a la posición en el eje Y de la estrella, creando un efecto de flotación suave y constante. Además del movimiento vertical, se aplica una rotación lenta sobre el eje Y a través de una variable de rotación. Esta variable se actualiza gradualmente en cada cuadro de animación, permitiendo que la estrella gire de manera continua y constante en su propio eje. La combinación del movimiento ascendente y descendente junto con la rotación lenta en el eje Y crea una animación fluida y encantadora para la estrella.

```
//Estrella
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.0f, 8.884f + (auxilar1 / 2000), 0.0f));
model = glm::rotate(model, glm::radians(Rotacion2), glm::vec3(0.0f, 1.0f, 0.0f));
glUniform4f(glGetUniformLocation(lightningShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Star.Draw(lightningShader);
```

- **Toad saludando**

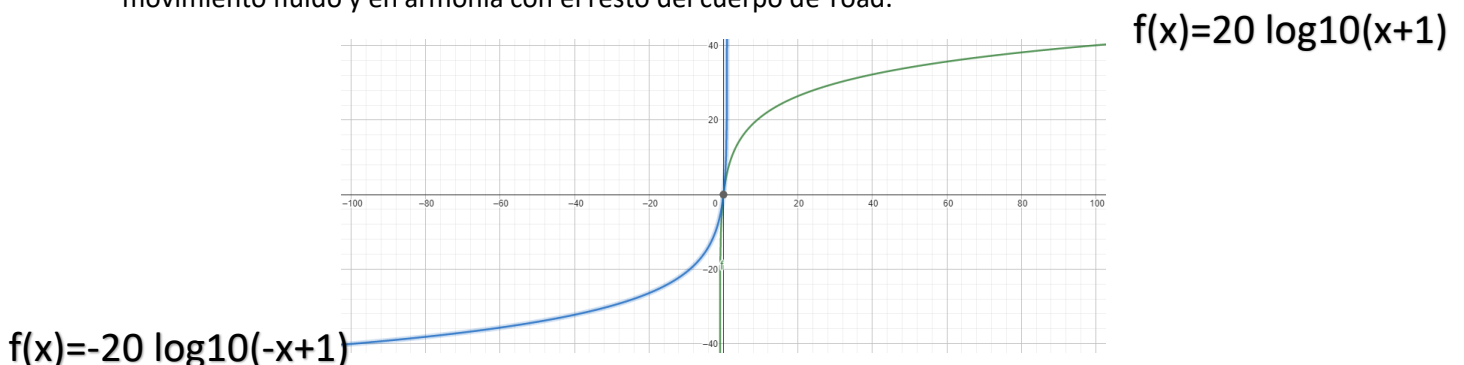
Para la animación de Toad saludando, se utiliza una rotación en el eje X que puede ser incrementada y decrementada. Tanto el cuerpo de Toad como su brazo realizan esta rotación para simular el gesto de saludo. Se reutiliza la variable de rotación brazo utilizada con los Toads anteriores para que vaya al mismo ritmo que los toads que se encuentran en el cuarto del trono.

```
model = glm::mat4(1);
modelaux = model;
model = glm::translate(model, glm::vec3(0.0f, 11.663f, 1.151f));
model = glm::rotate(model, glm::radians(RotBrazo - 10), glm::vec3(0.0f, 0.0f, 1.0f));
glUniform4f(glGetUniformLocation(lightningShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
CuerpoToad.Draw(lightningShader);
modelaux = model;
model = modelaux;
model = glm::translate(model, glm::vec3(-0.059f, 0.165f, 0.0f));
model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(4 * RotBrazo - 20.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniform4f(glGetUniformLocation(lightningShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
BrazoDer.Draw(lightningShader);
model = modelaux;
model = glm::translate(model, glm::vec3(0.044f, 0.175f, 0.0f));
model = glm::rotate(model, glm::radians(-25.0f), glm::vec3(0.0f, 0.0f, 1.0f));
glUniform4f(glGetUniformLocation(lightningShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
BrazoIzq.Draw(lightningShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.0f, 11.645f, 1.151f));
glUniform4f(glGetUniformLocation(lightningShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Zapatos.Draw(lightningShader);
```

6.2 Animaciones Complejas

- **Toad moviendo sus brazos**

La animación de Toad realizando el movimiento de cruzar los brazos alternadamente se caracteriza por un gesto en el que gira el torso mientras extiende un brazo hacia adelante y el otro brazo se mueve hacia atrás. Este movimiento se repite de forma intercalada, creando un efecto visual dinámico y energético. Para lograr esta animación, se utiliza una técnica basada en una variable auxiliar que va de -100 a 100. Esta variable es clave para determinar la posición y rotación de los brazos y el torso de Toad. La función logaritmo se emplea para controlar el movimiento de los brazos, de manera que cuando la variable es positiva, se aplica una función logaritmo positiva, y cuando la variable es negativa, se utiliza una función logaritmo negativa. Esto permite que los brazos realicen un movimiento fluido y en armonía con el resto del cuerpo de Toad.



Durante la animación, el brazo de la posición adelantada se mantiene por un breve momento extendido hacia adelante, antes de realizar el cambio de posición. Este efecto se logra aprovechando las propiedades del logaritmo, que crea una pausa en el movimiento en ciertos puntos específicos.

Un detalle interesante de esta animación es que, al alcanzar el punto máximo de rotación, independientemente de si es hacia la derecha o hacia la izquierda, se produce un pequeño aumento en el movimiento general de los brazos y el torso. Esto acentúa el gesto realizado por Toad, añadiendo un toque de expresividad a este movimiento.

```

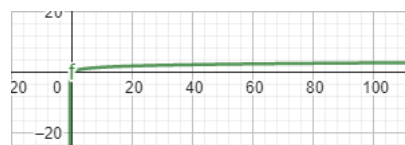
if (Sentido1)
{
    auxiliar1 += 2;
    if (auxiliar1 > 99)
    {
        Sentido1 = false;
    }
}
if (!Sentido1)
{
    auxiliar1 -= 2;
    if (auxiliar1 < -99)
    {
        Sentido1 = true;
    }
}
if (auxiliar1 > 0)
{
    Giro = 20 * log10(auxiliar1 + 1);
    Brinco = 0.025 * (Giro / 40);
}
if (auxiliar1 < 0)
{
    Giro = -20 * log10(-auxiliar1 + 1);
    Brinco = 0.025 * (-Giro / 40);
}

model = glm::mat4(1);
modelaux = model;
model = glm::translate(model, glm::vec3(5.716f, 2.029f + (Brinco * 1.7), -5.042f));
model = glm::rotate(model, glm::radians(-38.0f + (Giro / 4)), glm::vec3(0.0f, 1.0f, 0.0f));
glUniform4f(glGetUniformLocation(LightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
CuerpoToad.Draw(LightingShader);
modelaux = model;
model = modelaux;
model = glm::translate(model, glm::vec3(-0.044f, 0.175f, 0.0f));
model = glm::rotate(model, glm::radians(25.0f), glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::rotate(model, glm::radians(Giro), glm::vec3(0.0f, 1.0f, 0.0f));
glUniform4f(glGetUniformLocation(LightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
BrazoDer.Draw(LightingShader);
model = modelaux;
model = glm::translate(model, glm::vec3(0.044f, 0.175f, 0.0f));
model = glm::rotate(model, glm::radians(-25.0f), glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::rotate(model, glm::radians(Giro), glm::vec3(0.0f, 1.0f, 0.0f));
glUniform4f(glGetUniformLocation(LightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
BrazoIzq.Draw(LightingShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5.716f, 2.029f + Brinco, -5.042f));
model = glm::rotate(model, glm::radians(-38.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(Brinco * 800), glm::vec3(1.0f, 0.0f, 0.0f));
glUniform4f(glGetUniformLocation(LightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Zapatos.Draw(LightingShader);

```

- **Paratroopa volando dentro y fuera del cuadro**

El Paratroopa comienza rápidamente saliendo del cuadro y luego reduce su velocidad para mantener un movimiento constante, pero con una ligera progresión hacia adelante. Antes de completar este ciclo, realiza un giro completo de 180 grados y avanza de manera constante pero lenta hacia el cuadro de regreso. Este efecto se logra utilizando una función logarítmica para el recorrido del Paratroopa. Cuando la variable X aumenta, se verifica si ha alcanzado cierto valor para aumentar el valor de rotación y lograr así el giro de 180 grados de regreso al cuadro. Una vez que X alcanza su valor máximo, el Paratroopa disminuye gradualmente su valor de X, lo que permite repetir la secuencia una y otra vez. Además, se reutiliza la misma variable auxiliar de la misma manera que la estrella y el Fly Guy para que suba y baje levemente para lograr un bonito efecto de vuelo mientras realiza el trayecto descrito.



$$f(x)=1.5 \log_{10}(x+1)$$

```

//Paratroopa////////////////////////////////////
if (Sentido2)
{
    VueloParatroopa = 1.5 * log10(auxiliar2 + 1);
    auxiliar2 += 0.09;
    if (auxiliar2 > 79.91)
    {
        RoatParatroopa += 0.81;
    }
    if (auxiliar2 > 99.91)
    {
        Sentido2 = false;
    }
}
if (!Sentido2)
{
    if (VueloParatroopa > 0.3)
    {
        VueloParatroopa -= 0.01;
    }
    if (VueloParatroopa < 0.3 && VueloParatroopa>0)
    {
        VueloParatroopa -= 0.0005;
    }
    if (VueloParatroopa < 0.3 && VueloParatroopa < 0)
    {
        RoatParatroopa = 0;
        auxiliar2 = 0;
        Sentido2 = true;
    }
}

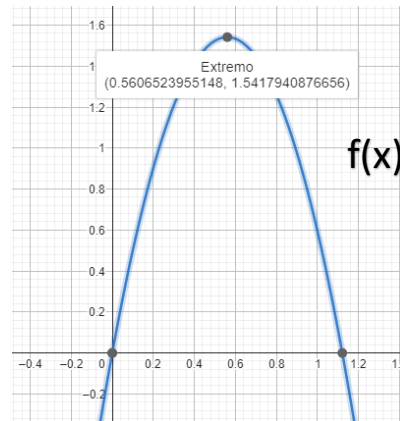
//Paratroopa
model = glm::mat4(1);
modelaux = model;
model = glm::translate(model, glm::vec3(3.422f, 1.257 + (auxiliar1 / 2000), -4.794f + VueloParatroopa));
model = glm::rotate(model, glm::radians(RoatParatroopa), glm::vec3(0.0f, 1.0f, 0.0f));
glUniform3f(glGetUniformLocation(LightingShader.Program, "dirLight.ambient"), 0.9f, 0.9f, 0.9f);
glUniform3f(glGetUniformLocation(LightingShader.Program, "dirLight.diffuse"), 0.7f, 0.7f, 0.7f);
glUniform3f(glGetUniformLocation(LightingShader.Program, "dirLight.specular"), 0.5f, 0.5f, 0.5f);
glUniform4f(glGetUniformLocation(LightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Paratroopa.Draw(LightingShader);
modelaux = model;
model = modelaux;
model = glm::translate(model, glm::vec3(-0.061f, 0.085f, -0.042f));
model = glm::rotate(model, glm::radians(RotBrazo * 4), glm::vec3(0.0f, 0.0f, 1.0f));
glUniform4f(glGetUniformLocation(LightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
AlaDerecha.Draw(LightingShader);
model = modelaux;
model = glm::translate(model, glm::vec3(0.061f, 0.085f, -0.042f));
model = glm::rotate(model, glm::radians(-RotBrazo * 4), glm::vec3(0.0f, 0.0f, 1.0f));
glUniform4f(glGetUniformLocation(LightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
AlaIzquierda.Draw(LightingShader);

```


- **Champiñón dentro del bloque**

El champiñón es disparado desde un question block y se lanza hacia arriba siguiendo las reglas de la física. A medida que asciende, va perdiendo velocidad hasta alcanzar su punto más alto y luego comienza a caer. Durante este proceso, el cubo se escalará, volviéndose más pequeño, mientras que otro cubo más pequeño se hará más grande para representar el question block vacío. Cuando el champiñón regresa a su posición original, los cubos regresan a sus posiciones originales también.

Esta animación se logra utilizando la fórmula $Y = 5.5 * X - 0.5 * 9.81 * X^2$, que representa el movimiento vertical del champiñón bajo la influencia de la gravedad. Mientras el Power-Up está en vuelo hará una rotación en Z para darle más dinamismo al giro. Esta animación se activa presionando una vez la tecla C y no puede interrumpirse. Solo se puede reiniciar cuando la secuencia completa ha terminado.



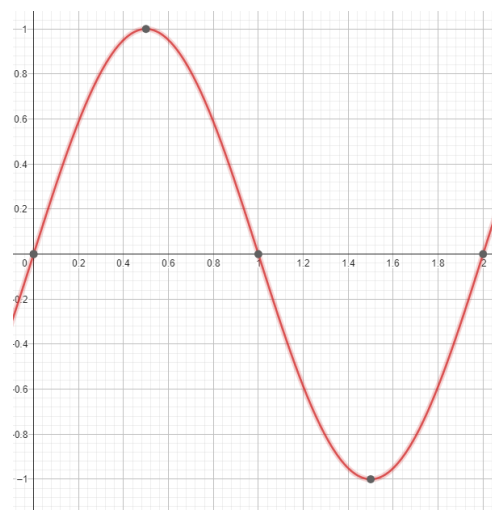
```
//Cubo-Champiñon////////////////////////////////////
if (keys[GLFW_KEY_C])
{
    AnimCubo = true;
}
if (AnimCubo)
{
    if (EstadoCubo)
    {
        if (Golpe < 0.3)
        {
            Golpe += 0.04;
        }
        else
        {
            EscalaVacía = 1.00f;
            EscalaCubo = 0.5;
            EstadoCubo = !EstadoCubo;
        }
    }
    if (!EstadoCubo)
    {
        auxiliar3 += 0.006;
        RoatChampi += 2;
        TiroVertical = 5.5 * auxiliar3 - 0.5 * 9.81 * auxiliar3 * auxiliar3;
        if (Golpe > 0.05)
        {
            Golpe -= 0.05;
        }
        if (auxiliar3 > 1.12)
        {
            EscalaCubo = 1.00f;
            EscalaVacía = 0.5f;
            auxiliar3 = 0.0f;
            RoatChampi = 0.0f;
            TiroVertical = 0.0f;
            EstadoCubo = !EstadoCubo;
            AnimCubo = false;
        }
    }
}
```

```
//Cubo
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-15.425f, 1.404f + Golpe, 4.719f));
model = glm::scale(model, glm::vec3(EscalaCubo, EscalaCubo, EscalaCubo));
glUniform4f(glGetUniformLocation(lightningShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Cubo.Draw(lightningShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-15.425f, 1.404f + Golpe, 4.719f));
model = glm::scale(model, glm::vec3(EscalaVacía, EscalaVacía, EscalaVacía));
glUniform4f(glGetUniformLocation(lightningShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Vacío.Draw(lightningShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-15.425f, 1.404f + TiroVertical + Golpe, 4.719f));
model = glm::rotate(model, glm::radians(RoatChampi), glm::vec3(0.0f, 0.0f, 1.0f));
glUniform4f(glGetUniformLocation(lightningShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Champi.Draw(lightningShader);
```

- **Mario Tanooki volando**

Mario asciende y desciende mientras mueve su colita arriba y abajo, lo cual se logra mediante una función $\text{Sen}(x)$ que determina el aumento y disminución de su altura. Esta función crea un efecto suave y fluido en el movimiento vertical de Mario. Para evitar el llenado de memoria la variable X se reinicia en 0 para que continúe el ciclo sin interrumpir la animación.

Además del movimiento de la cola, se agrega una variable auxiliar para hacer que Mario gire ligeramente de lado a lado, lo cual añade un efecto más natural al vuelo del personaje. Este sutil movimiento lateral complementa el ascenso y descenso de Mario, creando una sensación de vuelo más dinámica y realista. El traje de Tanooki le da a Mario la capacidad de volar y esta animación resalta esa característica.



$$f(x) = \sin(x \cdot 3.1416)$$

```
//Mario Tanooki
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.0f, 11.201f + VueloAltura/2, -3.490f));
model = glm::rotate(model, glm::radians(180.0f + 10 * (auxilar1 / 100)), glm::vec3(0.0f, 1.0f, 0.0f));
glUniform4f(glGetUniformLocation(lightningShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Mario.Draw(lightningShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.0f, 11.201f + VueloAltura/2, -3.490f));
model = glm::rotate(model, glm::radians(180.0f + 10 * (auxilar1 / 100)), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(20.0f * (Viento / 3)), glm::vec3(1.0f, 0.0f, 0.0f));
glUniform4f(glGetUniformLocation(lightningShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Colita.Draw(lightningShader);
....

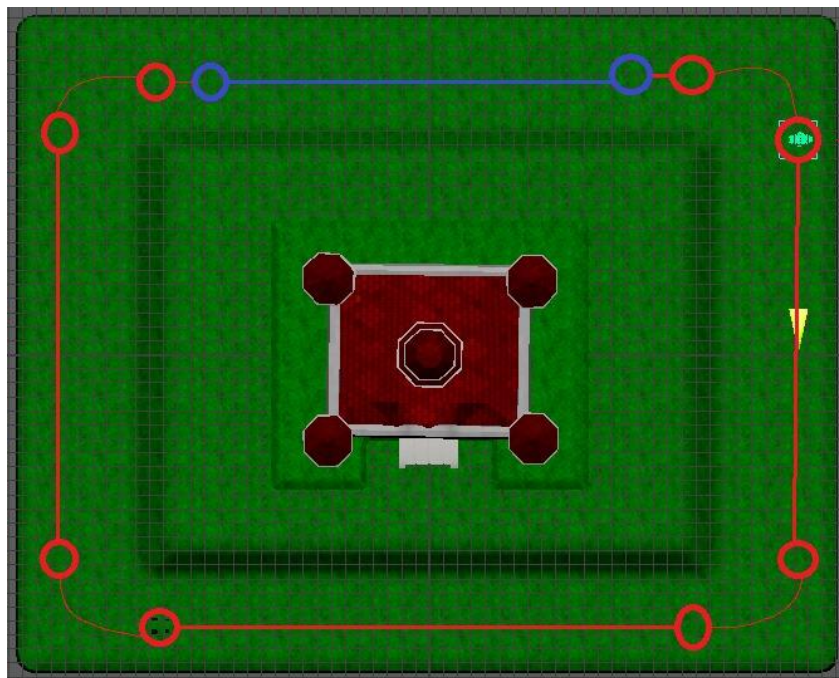
//Mario Tanooki
Tanooki += 0.01;
if (Tanooki > 2)
{
    Tanooki = 0.0f;
}
VueloAltura = 1 * sin(Tanooki * 3.1416);
```

- **Princesa Peach conduciendo un kart**

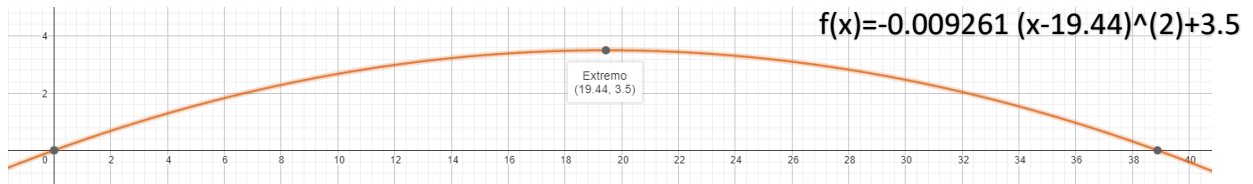
La animación de Peach en el kart fue la más compleja de este proyecto. Ella recorre todo el entorno del castillo en un circuito bien establecido. A medida que avanza, se encuentra con una rampa en la parte trasera del castillo, que la impulsa a volar por los aires.

Cuando Peach se eleva en el aire, despliega un paracaídas para mantener un vuelo constante, similar a lo que se ve en Mario Kart 8. El paracaídas le permite mantener una trayectoria suave y estable en el aire mientras disfruta del vuelo. Una vez que regresa al suelo, guarda su paracaídas y continúa su recorrido, repitiendo el circuito una y otra vez.

Esta compleja animación se logra a través de una serie de rutas predefinidas. Se marcan ocho recorridos en total. Comienza con el recorrido 1, que consiste en una línea recta. Una vez completada esta ruta, se activa el recorrido 2, que implica un giro mientras se disminuye el aumento en la distancia X y se aumenta el incremento en Z mientras aumenta un valor de rotación para que el coche sea más realista en cuanto a su recorrido. Luego, se activa el recorrido 3, y así sucesivamente, hasta completar el cuadrado y comenzar de nuevo.



Sin embargo, en el recorrido 5, la animación se vuelve aún más interesante. Peach no solo avanza en línea recta, sino que también sube en el eje Y. Se utiliza una fórmula de parábola en función de la variable X para crear el efecto de vuelo ascendente. Durante esta parte del recorrido, Peach realiza una acrobacia girando 380 grados, añadiendo un toque de emoción y estilo a su vuelo.

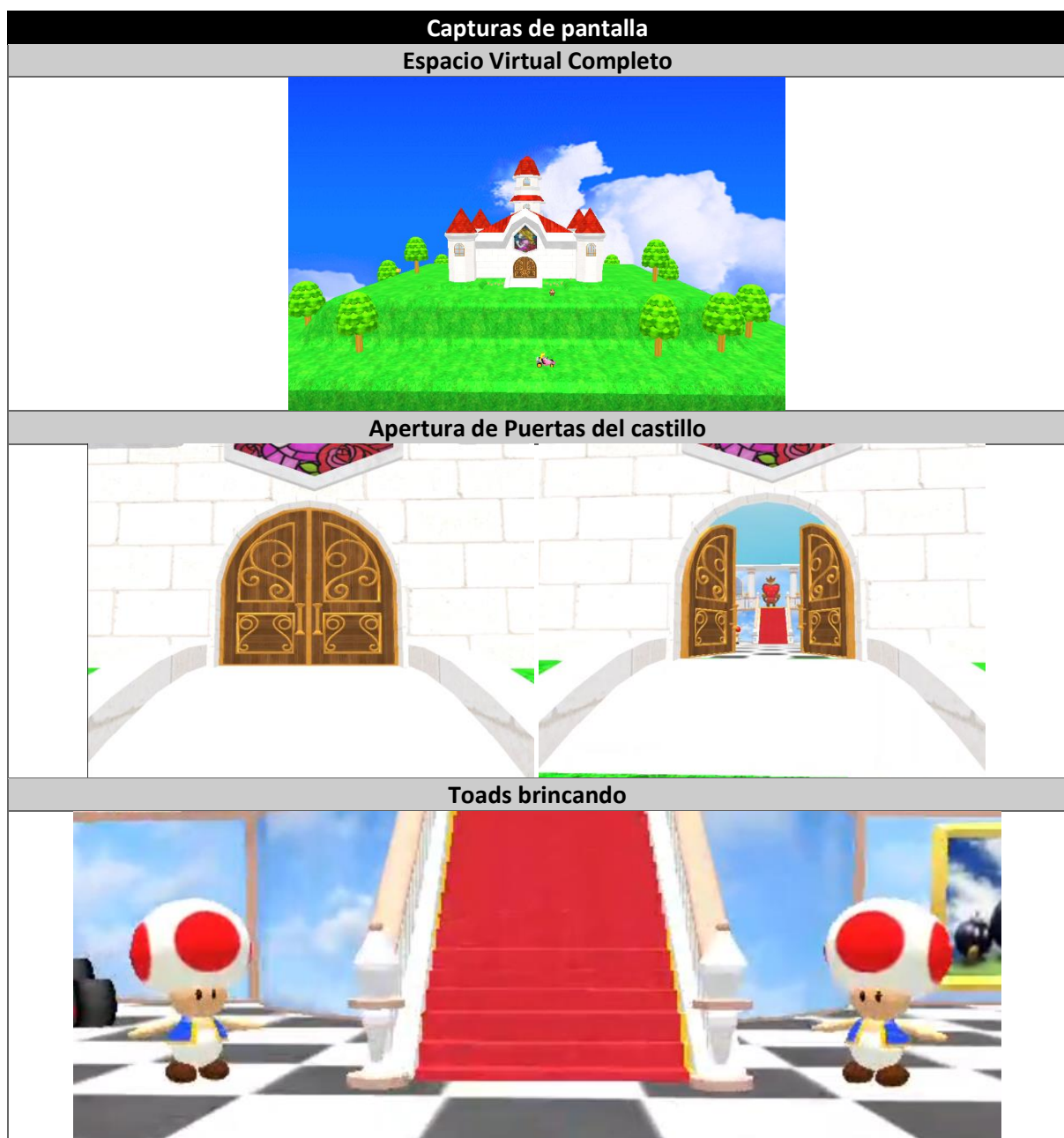


Cuando Peach termina el circuito todos los valores vuelven a su valor que tenía cuando se inicializó el programa para asegurar que repita el recorrido de la misma exacta manera.

Durante el vuelo, el paracaídas se escala gradualmente para que sea visible. Además, el paracaídas rota rápidamente de forma ligera de un lado a otro para simular el viento. Una vez que el valor de Y vuelve a ser 0, el paracaídas se guarda y el kart retoma las rotaciones iniciales del recorrido 5 para continuar con el circuito.

7-Resultados con ejemplos visuales

En esta sección, presentaremos los resultados obtenidos y ejemplos visuales de las animaciones implementadas en nuestro programa utilizando OpenGL. Aquí podrás apreciar el resultado final de este proyecto y visualizar de manera concreta las animaciones descritas anteriormente.



Pintura



ParaTroopa saliendo de la pintura



Kart inmóvil



Trono, flores y jarrones



Ventana translúcida



Toad moviendo las manos



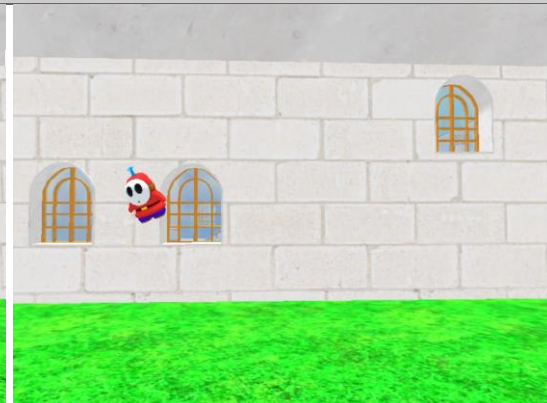
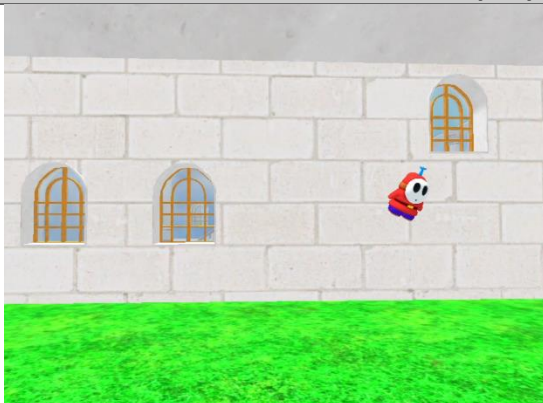
Goomba caminando



Flores bailando



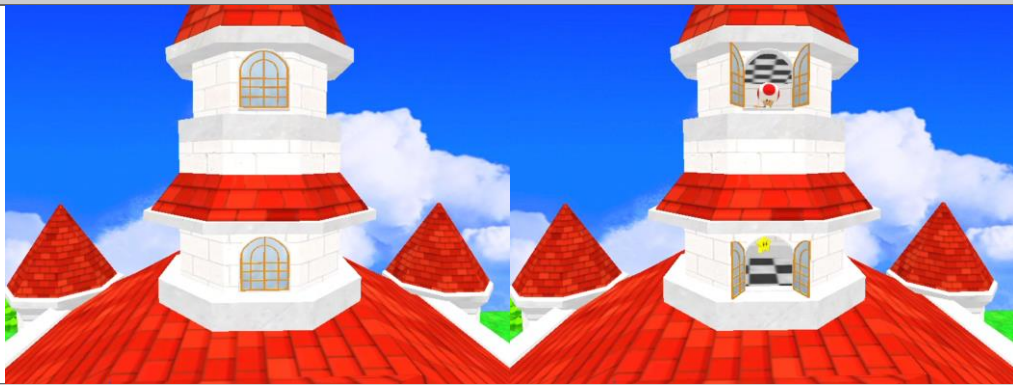
FlyGuy volando



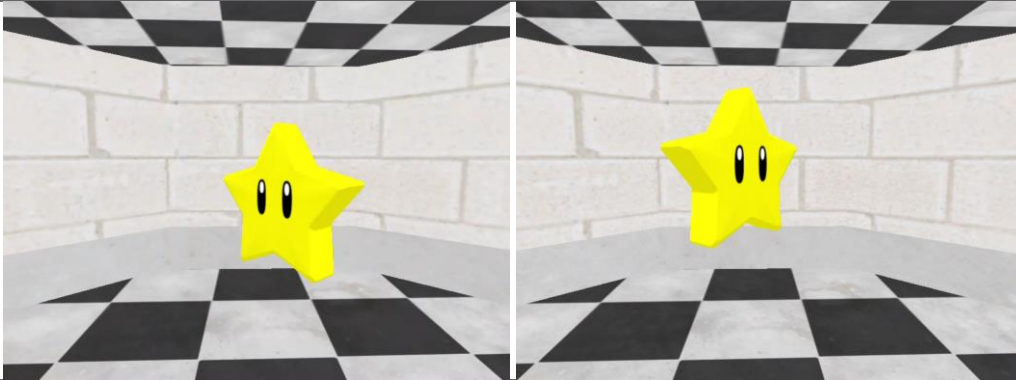
Champiñón tiro vertical



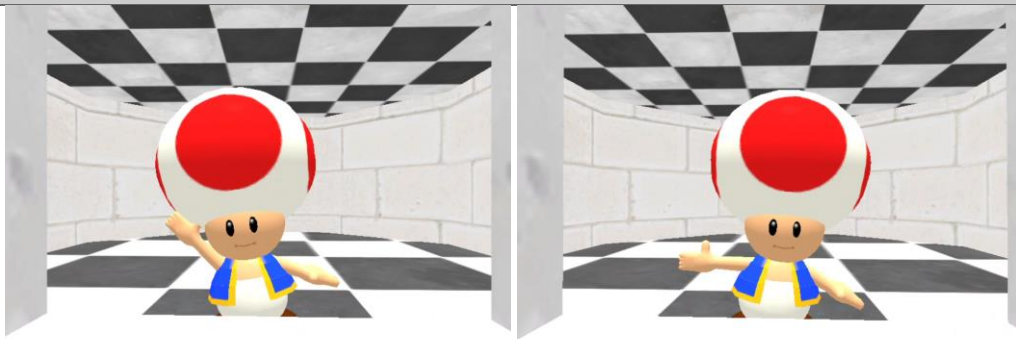
Ventanas



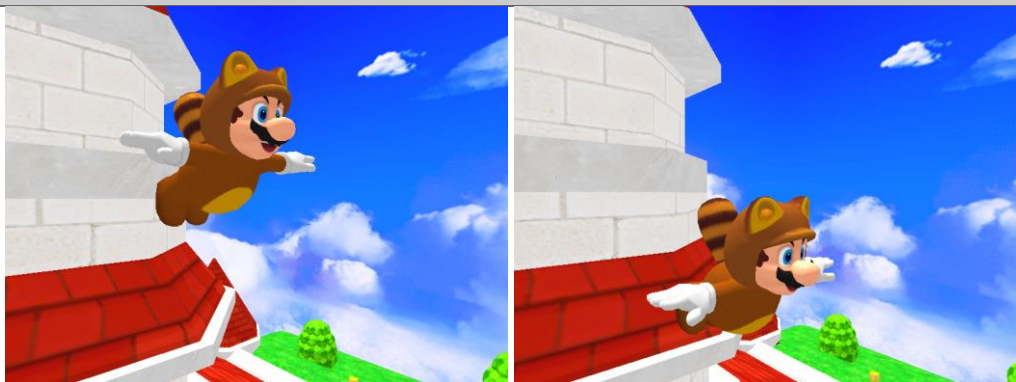
Estrella rotando



Toad Saludando



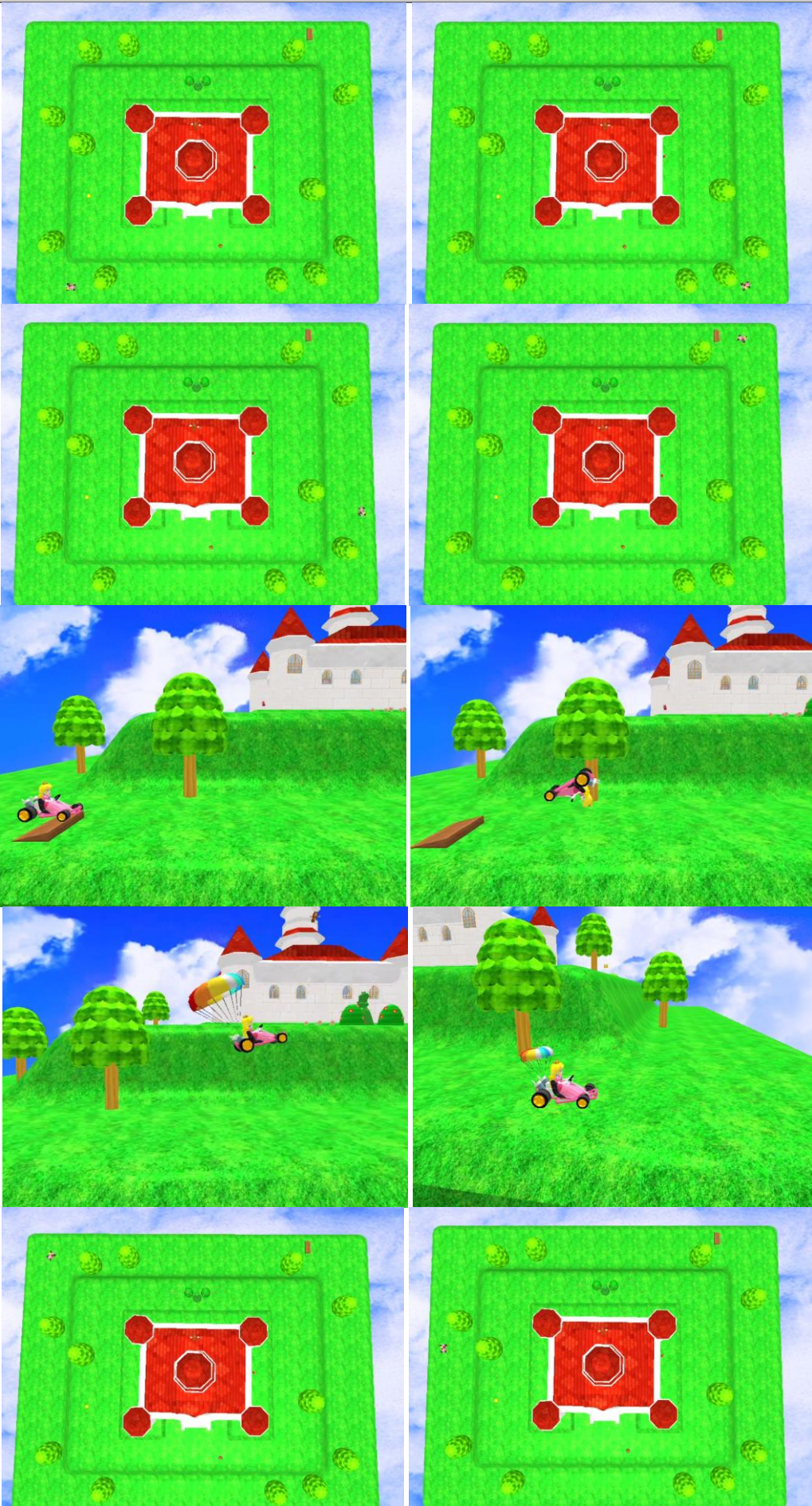
Mario Tanooki



Arbustos



Recorrido de Peach



8-CONCLUSIONES

En este manual técnico, hemos explorado los fundamentos de OpenGL y cómo aplicarlos para crear un impresionante castillo inspirado en el mundo de Mario. Mediante el uso de las técnicas de modelado 3D, iluminación y animación que hemos cubierto, así como hemos logrado recrear de manera fiel el encanto y la magia de los juegos de Mario.

Durante el proceso de modelado, hemos utilizado herramientas como Maya para crear detallados modelos en 3D de los diferentes elementos del castillo, incluyendo torres, paredes y el emblemático vitral de Peach. Además, hemos modelado objetos y personajes representativos de este mundo para darle un toque aún más vivo.

Con respecto a la iluminación, hemos aplicado técnicas como la iluminación ambiental, difusa y especular para resaltar los detalles arquitectónicos del castillo y crear una atmósfera realista. Esto ha permitido que la luz se refleje de manera adecuada en las superficies y genere sombras que aportan profundidad y realismo a la escena.

La animación ha sido un aspecto clave para hacer que el castillo cobre vida. Mediante la implementación de animaciones fluidas, hemos logrado que los personajes se muevan, salten e interactúen con su entorno.

A lo largo del manual, hemos proporcionado ejemplos visuales y explicaciones detalladas de los pasos necesarios para lograr este resultado final. Por mi parte esto ha sido una experiencia que disfrute desde principio a fin, desarrollando por fin mi pasión por algo, emocionarme por cada vez que terminaba un modelo, animarlo, combinar todo lo visto en clase y aplicarlo para crear algo que no me termino de creer que eh hecho, que disfruto de ver y compartir.

9-Enlace de GitHub

Para acceder a toda esta información o recuperar una copia del proyecto acceder al siguiente repositorio. Repositorio de Aldair20:

https://github.com/Aldair20/317222049_PROYECTOFINAL2023-2_GPO06/tree/main/ProyectoFinal/Models