UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA

INGENIERÍA EN COMPUTACIÓN

COMPUTACIÓN GRÁFICA e INTERACCIÓN HUMANO COMPUTADORA

Manual Técnico

Alumno	No. de cuenta
Barrios López Francisco	317082555
Mendoza Anaya Aldair Israel	317222049
Páez López Didier Marcelo	317224108

GRUPO DE TEORÍA: 04

SEMESTRE 2023-2

CALIFICACIÓN: _____

Contenido

Objetivos del proyecto

1. Desarrollar un ambiente virtual interactivo con elementos propios de la vida cotidiana de varios personajes de ficción.
2. Implementar la geometría y texturización de por lo menos 10 elementos de la vida cotidiana, incluyendo vehículos, alimentos y bebidas, flora y fauna, arquitectura y elementos industriales, y habitantes.
3. Incluir animación básica, animación compleja y animación por keyframes en los elementos del escenario, y utilizar una metodología de desarrollo que incluya un sistema de almacenamiento, documento de propuesta y croquis/boceto con el diseño del escenario a crear.
4. Permitir al usuario recorrer el escenario con una cámara de tercera persona y una cámara isométrica, además de contar con iluminación puntual que refleje el ciclo de día y noche y luces spotlight que se puedan prender y apagar con el teclado.

5. El proyecto debe estar optimizado para que corra en distintas configuraciones de hardware sin problemas. Se debe tener en cuenta la carga gráfica, la cantidad de elementos en el escenario y el número de luces activas en todo momento.

Presentación

El proyecto "Diorama de la vida cotidiana" es un ambiente virtual interactivo que permite al usuario explorar una escena con elementos característicos de la vida cotidiana de varios personajes de ficción. Este proyecto será entregado en equipos de dos o tres personas y consta de varios elementos que se deben incluir en el escenario.

Para poder utilizar este proyecto, es importante seguir los siguientes pasos:

6. Ejecutar el archivo de tipo exe que se entrega con el proyecto.
7. Recorrer el escenario utilizando la cámara de 3era persona ligada a un plano paralelo al plano XZ que representa el piso. También se puede cambiar a una cámara isométrica que permitirá mostrar el escenario de forma total y permite desplazarse para acercar o alejar la cámara.
8. Interactuar con los elementos del escenario usando el teclado. Se pueden iniciar animaciones básicas en los elementos del escenario, como también interactuar con las luces de tipo Spotlight que simulan ser luces que podemos manipular en la vida cotidiana.
9. Observar el cambio en las texturas del skybox, que refleja el ciclo de día y noche del ambiente virtual.

En cuanto a la metodología de desarrollo, se debe utilizó un repositorio de GithHuben el cual se almacenaron versiones del proyecto para demostrar cómo fue avanzando en su desarrollo. También se debe entregó una propuesta de proyecto en la plataforma de Google Classroom y un croquis/boceto con el diseño del escenario.

Introducción

En el ámbito de la computación gráfica, la capacidad de modelar entornos tridimensionales y objetos virtuales ha sido fundamental para crear mundos digitales inmersivos y realistas. En el marco de este proyecto final, hemos desarrollado un entorno tridimensional que incluye una choza, un cuarto y siete objetos que se encuentran dentro de dicho cuarto. El objetivo principal de este proyecto ha sido aplicar los conocimientos adquiridos en la asignatura de computación gráfica y demostrar nuestra habilidad para crear y manipular elementos virtuales en un espacio tridimensional.

En nuestro proyecto hemos recreado un diorama con atractivo principal de “El castillo de Peach”. El Castillo de Peach es un lugar emblemático en el universo de los videojuegos de Mario Bros. Aparece principalmente en la serie de juegos de plataformas de Super Mario. El castillo es la residencia de la Princesa Peach, uno de los personajes principales de la serie.

El proyecto no solo se limitó a representar este castillo y varios objetos en el diorama, sino que el equipo de trabajo se esforzó en darle un sentido a cada una de las piezas que tiene

nuestro ambiente recreado, el cual se podrá ver en una sección del manual de usuario, así como cada animación que presentamos.
A lo largo de esta documentación, presentaremos los pasos y procesos que hemos seguido para lograr nuestros objetivos, desde la creación y carga de modelos tridimensionales hasta la implementación de técnicas de renderizado y manipulación de objetos en tiempo real.

Metodología de desarrollo.

El propósito de implementar una metodología en un proyecto de desarrollo de software es establecer un enfoque estructurado y organizado para guiar el proceso de creación del software.
El objetivo principal de este proyecto ha sido aplicar los principios y técnicas de la programación gráfica para crear un entorno virtual convincente y visualmente atractivo. La representación precisa de nuestro diorama, así como la disposición y la interacción de los objetos en el entorno, han sido aspectos clave en nuestro enfoque.
Hemos creado y ubicado varios objetos dentro del entorno, cada uno con sus características y comportamientos únicos. Todo esto se ha llevado a cabo utilizando las capacidades versátiles y flexibles de OpenGL.
Como modelo de metodología de software se utilizó Scrum, esto gracias a la flexibilidad, adaptabilidad, gestión efectiva del tiempo y prioridades, y sobre todo por la mejora continua que tendríamos gracias a utilizar esta metodología ágil.
Para entender mejor la metodología de desarrollo utilizada en este proyecto se puede consultar el documento del “Plan del proyecto” el cual será adjuntado con este mismo documento, y ahí se puede observar detalladamente el avance del proyecto.

Control de calidad

Para el proyecto el control de calidad se basa en dos aspectos principales, en que el objeto haya sido modelado con respecto a los objetos propuestos y que esté bien optimizado para su posterior carga con ayuda de OpenGL. Conforme se fue avanzando en las prácticas del semestre 2023-2 así como las clases de teoría con las explicaciones técnicas el control de calidad aumentaba en cuanto a sus requerimientos, el primero criterio que se agregó para considerar el modelo adecuado para el proyecto fue que se estuviera bien texturizado para que al incluirlo al producto final fuera visualmente atractivo para el espectador. El siguiente criterio agregado fue el uso correcto de la ambientación, es decir estuviera iluminado con sentido y adecuadamente. Por último, se consideró el criterio de animar con sentido, en caso de que el objeto tuviera una animación dentro del escenario primero se debe proponer y justificar el contexto del porqué aparecerá la animación en el producto final.

Cumplimiento de plazos y presupuesto

Para poder terminar el proyecto adecuadamente a los requisitos planteados desde el inicio, se siguió un cronograma, el cual tenía el cumplimiento de tareas, tomando en cuenta posibles

retrasos o desviaciones que se podrían presentar durante el proyecto y así tomar las medidas necesarias para administrar esas situaciones y evitar un alto impacto en el proyecto.

Mejora continua

Una característica importante del modelo seleccionado se basa en la mejor continua, al ser un proyecto que vamos a ir agregando objetos al diorama que propusimos, y no tendremos que descartar los que ya hemos realizado, la metodología seleccionada es perfecta para este proyecto y cada vez que se realizaba un modelo o se descargaba de alguna plataforma que proporcionan modelos en 3D útiles para poder incluirlos dentro del cuarto seleccionado.
Cada vez que se realizaba un modelo se obtenía de alguna plataforma, este se iba escalando, trasladando o rotando dentro del escenario para darle sentido y un resultado estéticamente agradable para el usuario final.
El proyecto fue realizado en equipos, y la comunicación fue un aspecto sumamente importante para su correcta conclusión. Durante las clases de teoría, así como a través de un grupo en la red social "WhatsApp", pudimos mantener una comunicación fluida y directa entre los participantes. Además, utilizamos la red social "Discord" para mantener una comunicación activa con el profesor, quien actuaba como cliente, lo que nos permitió realizar consultas, así como compartir comentarios y dudas.

Alcance del proyecto

El alcance del proyecto es diseñar un diorama que represente una escena con elementos característicos de la vida cotidiana de varios personajes de ficción, con previa aprobación. El escenario debe incluir elementos geométricos correctamente texturizados, con al menos 10 elementos propios de la vida cotidiana, como vehículos, alimentos, flora, fauna, arquitectura, elementos industriales, habitantes, y 1 o 2 personajes adicionales según el tamaño del equipo.
En cuanto a la iluminación, se deben incluir elementos emisores de luz como luminarias o lámparas en el escenario, con luces puntuales que se enciendan y apaguen simultáneamente siguiendo un ciclo día y noche determinado por los alumnos. El skybox también debe reflejar el cambio de texturas en el ciclo día y noche. Además, se deben agregar luces tipo Spotlight que se puedan encender y apagar mediante el teclado, simulando luces manipulables en la vida cotidiana.
Los alumnos deberán utilizar sus habilidades en modelado y diseño 3D para recrear los objetos de manera precisa, tomando como referencia las imágenes proporcionadas. Además, se espera que preste atención a los detalles y características específicas de cada objeto, así como a la atmósfera y estilo general del entorno.
El resultado final del proyecto será una representación tridimensional del espacio seleccionado, donde los objetos recreados se asemejen lo más posible a las imágenes de referencia y se refleje adecuadamente la ambientación deseada.
Es importante destacar que el proyecto se enfoca principalmente en la recreación visual y estética de los espacios y objetos seleccionados, utilizando OpenGL como herramienta para lograrlo.

Limitantes del proyecto

Disponibilidad de imágenes de referencia: Puede haber limitaciones en cuanto a la disponibilidad de imágenes de referencia de alta calidad y detalladas para nuestro diorama, el espacio y los objetos seleccionados. Esto podría dificultar la precisión y la fidelidad en la recreación 3D.
Complejidad del diorama y el espacio seleccionados: Si el diorama y el espacio elegidos son muy complejos en términos de arquitectura, detalles estructurales o características específicas, podría requerir un mayor nivel de habilidad y esfuerzo para recrearlos fielmente en el entorno 3D.
Recursos y tiempo disponibles: El proyecto puede tener limitaciones en cuanto a los recursos disponibles, como el tiempo asignado para completarlo, la capacidad de procesamiento de la computadora o la disponibilidad de software y herramientas necesarias para la creación en OpenGL.
Nivel de detalle de los objetos: Si los objetos de referencia son muy detallados o tienen características intrincadas, puede ser desafiante recrearlos virtualmente en OpenGL con un alto nivel de fidelidad y realismo.
Conocimientos y habilidades del equipo: El nivel de habilidad y conocimiento de los alumnos en OpenGL y modelado 3D puede afectar la calidad y la complejidad de la recreación. Limitaciones en términos de conocimientos técnicos podrían influir en las técnicas de renderizado y en la implementación de efectos visuales avanzados.
Limitaciones legales: Si se utilizan imágenes de referencia protegidas por derechos de autor, se deben cumplir las leyes y regulaciones correspondientes. Esto podría requerir obtener permisos o utilizar imágenes de dominio público o con licencias adecuadas para evitar problemas legales.
Una vez señaladas las principales limitantes del proyecto, procedimos a realizar un análisis de estos y tratar de encontrar un equilibrio para poder recrear el ambiente correctamente.
Para evitar los problemas de complejidad en el diorama que va de la mano con el nivel de detalle de los objetos, se optó por realizar el modelado de manera personal por el equipo, sin embargo; esto nos llevó a otra limitante, en la cual se destacan los conocimientos y habilidades de los alumnos para poder modelar los espacios en 3D seleccionados, como algunos modelos presentaban un reto mayor al conocimiento adquirido hasta el momento de la realización se optó por recurrir a plataformas que proporcionan modelos de manera libre, siempre respetando las limitaciones legales, al tratarse de un proyecto escolar y sin fin de lucro, podemos utilizar estos modelos y adaptarlos a nuestro escenario, siempre teniendo en cuenta los aspectos mencionados en el control de calidad.

Plataformas de trabajo

Requerimientos

Requerimientos	Descripción
Visual Studio 2022	Visual Studio 2022 es el programa en el cual podemos ingresar la solución diseñada para el proyecto y poder interactuar con este.

OpenGL	OpenGL es una librería necesaria para poder ejecutar el programa de manera adecuada, se debe utilizar la versión 3.3 en adelante.
Software de modelado	En el proyecto se emplearon las herramientas de diseño 3D Blender, Maya y 3ds MAX, las cuales pueden resultar útiles si se desea visualizar un modelo del proyecto de manera individual.
Software para la creación de texturas para modelos	Para desarrollar el proyecto se utilizó el software GIMP para poder crear, editar y exportar texturas que serán útiles para nuestro proyecto.

Software de modelado

Para realizar los modelos correspondientes al proyecto se utilizó el software de modelado Maya para poder realizar, manipular y exportar los modelos 3D seleccionados.
Maya es un software de modelado 3D y animación ampliamente utilizada en la industria del entretenimiento, la producción de películas, los videojuegos y la creación de efectos visuales. Desarrollado por Autodesk, Maya ofrece una amplia gama de herramientas y funcionalidades que permitieron crear modelos 3D detallados.
Es importante mencionar que para poder utilizar este software correctamente debemos contar con un ordenador que cuente con las especificaciones de hardware mínimas proporcionadas por el desarrollador, las cuales son:
CPU: Procesador Intel® o AMD® de 64 bits y varios núcleos, con el conjunto de instrucciones SSE4.2 Los modelos de Apple Mac con el chip de la serie M son compatibles en modo Rosetta 2
Hardware de gráficos: Consulte las siguientes páginas para obtener una lista detallada de los sistemas y las tarjetas gráficas recomendados:
Hardware certificado para Maya
RAM: 8 GB de RAM (se recomiendan 16 GB o más)
Espacio en disco: 4 GB de espacio libre en disco para la instalación
Dispositivo señalador: Ratón de tres botones

Software para la creación de texturas para modelos

En este proyecto se utilizó GIMP para la edición y creación de texturas que posteriormente serían asignadas a los modelos realizados.
GIMP (GNU Image Manipulation Program) es un software de edición de imágenes de código abierto y gratuito. Con GIMP, los usuarios pueden realizar una amplia gama de tareas de manipulación y edición de imágenes, desde simples ajustes de color y retoques hasta la creación de ilustraciones complejas y composiciones avanzadas.
Para poder utilizar el software GIMP de igual manera contamos con algunos requisitos mínimos para poder ser ejecutado en un ordenador correctamente, estos son:

Sistema operativo: GIMP es compatible con varios sistemas operativos, incluyendo Windows, macOS y Linux. Se recomienda utilizar la versión más actualizada del sistema operativo para garantizar un mejor rendimiento y compatibilidad.
Procesador: Se recomienda un procesador con al menos 1 GHz de velocidad para un funcionamiento fluido de GIMP.
Memoria RAM: GIMP requiere al menos 2 GB de memoria RAM para un rendimiento óptimo. Sin embargo, se recomienda disponer de 4 GB o más para trabajar con imágenes de mayor resolución y realizar operaciones más complejas.
Espacio en disco: GIMP requiere al menos 350 MB de espacio libre en el disco duro para su instalación. Además, se debe considerar espacio adicional para almacenar las imágenes y archivos creados o editados con el software.
Resolución de pantalla: GIMP funciona adecuadamente en una resolución de pantalla de al menos 1024x768 píxeles. Sin embargo, una resolución más alta permitirá una visualización más cómoda y detallada de las imágenes y herramientas.

Librería para ejecutar nuestro entorno

Como hemos mencionado a lo largo del manual técnico para poder hacer la carga correcta de nuestro entorno utilizaremos OpenGL.
OpenGL es una potente API de gráficos que permite a los desarrolladores crear aplicaciones y juegos interactivos con gráficos 2D y 3D de alta calidad. Su portabilidad, eficiencia y capacidad de rendimiento hacen de OpenGL una opción popular en la industria de gráficos y juegos.

Entorno de desarrollo

Para desarrollar el proyecto hicimos uso de un IDE (Entorno de Desarrollo Integrado), este fue Visual Studio 2022.
Visual Studio 2022 es la última versión de la suite de desarrollo de software de Microsoft. Es un entorno de desarrollo integrado (IDE) ampliamente utilizado que proporciona herramientas y características avanzadas para la creación de aplicaciones para diversos sistemas operativos, plataformas y dispositivos.
Visual Studio 2022 es un IDE poderoso y versátil que brinda a los desarrolladores las herramientas necesarias para crear aplicaciones de calidad en diferentes plataformas y lenguajes de programación. Su conjunto de características, integración con servicios en la nube y enfoque en la productividad hacen que sea una opción popular entre los desarrolladores de software.
Requisitos mínimos del entorno de trabajo.

- Procesador ARM64 o x64; se recomienda uno de cuatro núcleos o superior. No se admiten procesadores ARM 32.
- 4 GB de memoria, como mínimo. Hay muchos factores que afectan a los recursos usados, se recomiendan 16 GB de RAM para soluciones profesionales típicas.
- Windows 365: 2 vCPU y 8 GB de RAM, como mínimo. Se recomiendan 4 vCPU y 16 GB de RAM.
- Espacio en disco duro: mínimo de 850 MB y hasta 210 GB de espacio disponible, en función de las características instaladas; las instalaciones típicas requieren entre 20 y

50 GB de espacio libre. Se recomienda instalar Windows y Visual Studio en una unidad de estado sólido (SSD) para mejorar el rendimiento.

- Tarjeta de vídeo que admita una resolución de pantalla mínima de WXGA (1366 x 768); Visual Studio funcionará mejor con una resolución de 1920 x 1080 o superior.
    - La resolución mínima supone que el zoom, la configuración de PPP y el escalado de texto se establecen en un 100 %. Si no se establece en un 100 %, se debe ajustar la resolución mínima en consecuencia. Por ejemplo, si establece la configuración de pantalla de Windows "Escala y distribución" en la Surface Book, que tiene una pantalla física de 3000 x 2000, en 200 %, Visual Studio vería una resolución de pantalla lógica de 1500 x 1000, cumpliendo el requisito mínimo de 1366 x 768.

Una vez instalados los programas necesarios para poder desarrollar nuestro proyecto, podemos comenzar a desarrollarlo.

Para poder cargar correctamente los modelos con ayuda de OpenGL primero haremos uso de algunas librerías externas que serán vinculadas a través de la ventana de trabajo de Visual Studio 2022.

GLEW (The OpenGL Extension Wrangler Library):

GLEW es una biblioteca externa utilizada en el desarrollo de aplicaciones OpenGL. Su función principal es facilitar la gestión de extensiones de OpenGL en diferentes plataformas. GLEW proporciona una interfaz unificada para cargar y utilizar extensiones de OpenGL de forma dinámica, lo que permite a los desarrolladores acceder a características y funciones avanzadas de OpenGL que no están disponibles en la implementación base. GLEW simplifica la administración de extensiones de OpenGL y permite a los programadores aprovechar al máximo las capacidades de la API.

GLFW (Graphics Library Framework):

GLFW es una biblioteca de código abierto que proporciona una interfaz de programación de aplicaciones (API) para crear y gestionar ventanas, contextos de OpenGL y manejo de eventos en aplicaciones gráficas. Su función principal es abstraer la creación y gestión de ventanas y contextos gráficos en diferentes plataformas, lo que facilita el desarrollo de aplicaciones OpenGL multiplataforma. GLFW también ofrece características adicionales, como la detección de entradas de teclado y mouse, manejo de eventos, soporte de múltiples monitores y administración de temporizadores. GLFW simplifica la creación de ventanas y el manejo de eventos en aplicaciones gráficas.

glm (OpenGL Mathematics):

glm es una biblioteca matemática de código abierto diseñada específicamente para su uso con OpenGL. Proporciona una amplia gama de funciones y clases matemáticas que son útiles en el desarrollo de gráficos 3D y aplicaciones de renderizado. glm ofrece funcionalidades para realizar cálculos de vectores, matrices, transformaciones, proyecciones, intersecciones y otras operaciones matemáticas comunes en gráficos 3D. Esta biblioteca se inspira en la sintaxis de GLSL (OpenGL Shading Language) y proporciona una interfaz intuitiva y fácil de usar para realizar cálculos matemáticos en el contexto de OpenGL.

SOIL2 (Simple OpenGL Image Library):

SOIL2 es una biblioteca externa utilizada para cargar, manipular y guardar imágenes en aplicaciones OpenGL. Proporciona funciones y utilidades para cargar texturas en formatos comunes de imágenes, como JPEG, PNG, BMP y TGA. SOIL2 simplifica el proceso de carga de texturas en OpenGL, permitiendo a los desarrolladores cargar imágenes de manera eficiente y aplicarlas a objetos 3D en sus aplicaciones gráficas.

assimp (Open Asset Import Library):

assimp es una biblioteca externa que proporciona funciones y herramientas para importar y procesar modelos 3D en diferentes formatos en aplicaciones gráficas. Su función principal es facilitar la importación de modelos 3D desde una variedad de formatos comunes, como OBJ, FBX, Collada, DirectX, entre otros. assimp se encarga de cargar la geometría, las texturas, los materiales y otros datos relacionados con los modelos 3D, lo que permite a los desarrolladores utilizar modelos externos de manera eficiente y sencilla en sus aplicaciones. Además, assimp ofrece funcionalidades adicionales como la optimización de mallas, la manipulación de nodos de la escena y la gestión de animaciones. Simplifica el proceso de importación y procesamiento de modelos 3D en aplicaciones gráficas.

irrKlang

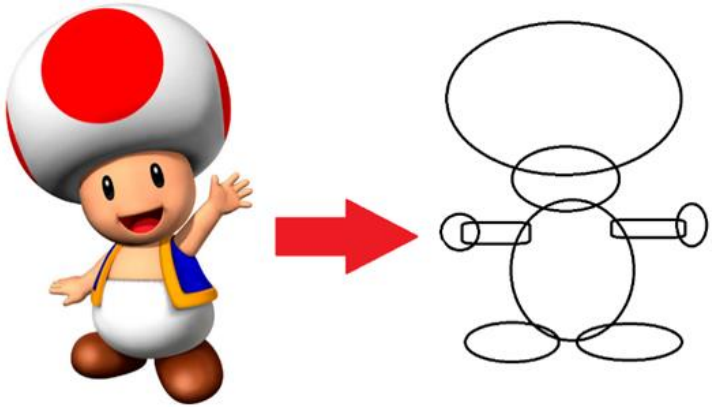
IrrKlang es una biblioteca de audio multiplataforma de alto rendimiento diseñada para proporcionar una solución fácil de usar para la reproducción de sonido en aplicaciones y videojuegos. Con su amplia compatibilidad, permite a los desarrolladores implementar de manera eficiente efectos de sonido, música de fondo y cualquier otro tipo de audio en sus proyectos.

Además, ofrece funciones avanzadas como el soporte de posicionamiento 3D de sonido, lo que permite una experiencia inmersiva al simular la ubicación espacial de los efectos de sonido. Gracias a su arquitectura eficiente, IrrKlang ofrece un rendimiento óptimo incluso en sistemas con recursos limitados, lo que lo convierte en una opción ideal para aplicaciones y videojuegos en plataformas como Windows, macOS, Linux y dispositivos móviles.

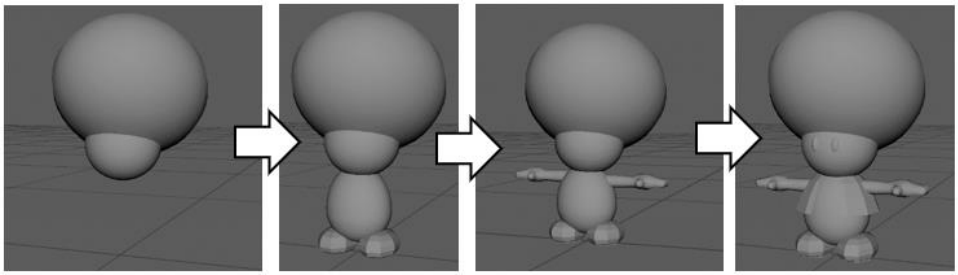
PROCESO CREATIVO

Para comenzar con el proyecto se comenzó con el proceso creativo de animación en tiempo real, el cual consta en primer lugar del modelado, seguido del texturizado, se planteó la animación en caso de tenerla, y finalmente la renderización del modelo en el escenario planteado.

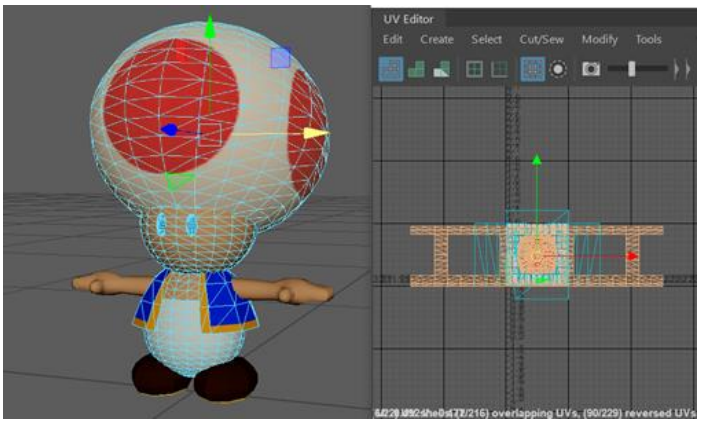
El modelado en Maya es un proceso creativo y técnico que permite dar vida a ideas y conceptos en forma de modelos 3D. Comienza con la planificación y conceptualización de los objetos que se desean crear. Esto implica definir la forma básica, las proporciones y los detalles específicos de cada componente 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 esté 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 de que se tiene es asemeja de forma más fiel al objeto/personaje deseado. Hubo casos en los que se decidió 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.

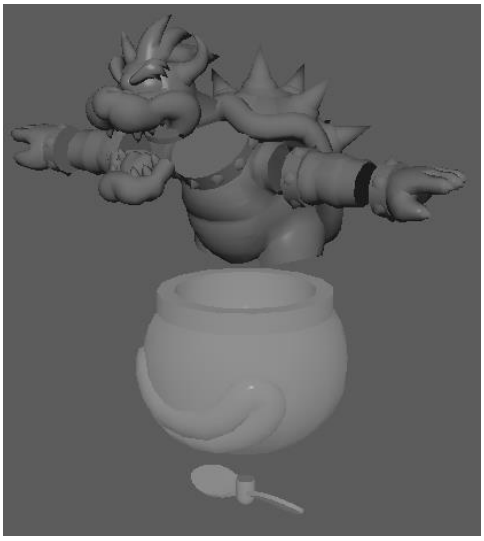
Primitivas:

Dentro del proyecto también se incluyeron objetos dibujados a partir de primitivas declaradas en código. Más específico, el letrero de las hamburguesas y el cartel del comienzo de la pista.

Dentro de un mismo arreglo de vértices e índices, se dibujan los triángulos necesarios para crear dichas figuras. Usando el concepto de texturización visto en clase, se usa una imagen sola para texturizar estas figaras, por lo que el mapeo fue aún más minucioso, para eso se recreó un cubo para el cubo de nota musical, dos cubos y dos planos para hacer la meta y dos cubos para el cartel de hamburguesa. Para lograr obtener las ubicaciones de los vértices, se crearon los modelos en maya, exportándolos sin texturización, pues solo nos interesa el archivo .obj, pues ahí vienen las coordenadas de cada vértice, haciendo posible el mapeo previamente explicado.

AVATAR

Para nuestro proyecto ambientado en el mundo de Mario Bros, hemos elegido a Bowser como el avatar principal, utilizando el modelo que se ve en New Super Mario Bros Wii. Para construir el avatar en OpenGL, hemos adoptado una estructura jerárquica, dividiendo el modelo de la siguiente manera:



- El Koopa Clown actúa como el elemento principal, siendo el padre.
- Las hélices y el torso de Bowser son sus hijos.
- El torso de Bowser es el padre de los brazos y la cabeza.

- La cabeza es el padre de la mandíbula.
- Los brazos son los padres de los antebrazos.
- Los antebrazos son los padres de las manos.

Para lograr la animación, hemos utilizado un incrementable que reinicia sus valores, lo que permite que las hélices giren constantemente, justificando así su capacidad de moverse incluso a alturas elevadas. Para el cuerpo de Bowser, hemos empleado funciones seno que le dan una rotación natural a cada parte de su cuerpo.
Además, hemos añadido una animación adicional para recrear su rugido. Para lograr esto, ampliamos el valor por el cual se multiplica la función seno, lo que hace que cada parte de su cuerpo rote aún más hacia atrás, creando la ilusión de que lanza un rugido. Mientras alcanza esta posición ideal, se inicia un contador que marca una bandera. Cuando la bandera está activa, reiniciamos los valores en un intervalo para que el modelo tiemble, simulando el temblor del rugido. Al mismo tiempo, una función seno adicional hace que su cabeza rote de lado a lado, aportando dinamismo al rugido. Durante este proceso, reproducimos un audio que contiene el rugido de Bowser, lo que da vida a la escena.
Una vez que el contador alcanza cierto valor, la bandera se desactiva y el avatar continúa su recorrido en la función seno para volver a su posición inicial. La animación regresa a un estado "IDLE" con una menor amplitud en la función seno. Esto completa el ciclo del rugido.

RECORRIDO

Cámara 3D

El avatar está ligado a una cámara 3D la cual toma base de la cámara libre que se proporcionó y se usó a lo largo del semestre.

Camera.h

En este archivo se maneja tanto la cámara 3D tanto la cámara isométrica. Comenzando con la cámara 3D, se modifica la función void ProcessKeyboard(Camera_Movement direction, GLfloat deltaTime) para restringir la posición de la cámara libre:

```
GLfloat velocity = this->movementSpeed * deltaTime;
if (!isometric) {
    if (direction == FORWARD) this->position += this->front * velocity;
    if (direction == BACKWARD) this->position -= this->front * velocity;
    if (direction == LEFT) this->position -= this->right * velocity;
    if (direction == RIGHT) this->position += this->right * velocity;
    if (this->position.y < 2.0f) this->position.y = 2.0f;
    if (this->position.y > 40.0f) this->position.y = 40.0f;
    if (this->position.x < -20.0f) this->position.x = -20.0f;
    if (this->position.x > 20.0f) this->position.x = 20.0f;
    if (this->position.z < -15.0f) this->position.z = -15.0f;
    if (this->position.z > 30.0f) this->position.z = 30.0f;
}
```

Cuando la cámara isométrica no está activa, se modifica la posición de la cámara. El movimiento de la misma se restringe mediante condicionales para que el personaje no salga del límite del mundo y tampoco atraviese el piso.
Dentro de este archivo, se genera una función que regresa la posición del mouse en horizontal, cuyo valor va de 0 a 360°

```
GLfloat getYaw() { return this->yaw; }
```

MainPrueba.cpp

Con el fin de ligar la cámara al personaje, este va a recibir dentro de la traslación la posición de la cámara. Al momento de girar la cámara, el avatar debe de girar junto con ella. Para esto, se sabe que, al mover la cámara, se genera un círculo por el cual el avatar debe ajustarse a la cámara. Para esto, se generan dos variables:

```
BowserCameraX = 1.75f * glm::cos(glm::radians(camera.getYaw()));
BowserCameraZ = 1.5f * glm::sin(glm::radians(camera.getYaw()));
```

Cómo el avatar está dibujado por jerarquía a partir de la copa, esta es la que va a tener posicionada la cámara.

```
model = glm::translate(model, glm::vec3(camera.GetPosition().x + BowserCameraX,
    camera.GetPosition().y + (auxilar1 / 2000) - 1.0f, camera.GetPosition().z + BowserCameraZ));
model = glm::rotate(model, glm::radians(- camera.getYaw() + 90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
modelaux = model;
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
KoopaClown.Draw(lightingShader);
```

Con esto, el personaje gira cuando la cámara gira, y siempre se ve al personaje mirando hacia el frente.

Cámara isométrica

Camera.h

Para la cámara isométrica, se comenzó definiendo las siguientes variables:

```
// variables camara isometrica
bool isometric = false;
GLfloat iso_right, iso_up;
glm::vec3 iso_position;
GLfloat iso_zoom = 2.0f;
```

A su vez, se declaran las siguientes funciones:

```
bool getIsometric() { return this->isometric; }
GLfloat getIsoZoom() { return this->iso_zoom; }
void setZoom(GLfloat zoom) { this->iso_zoom = zoom; }

void setIsometric(GLfloat iso) { this->isometric = iso; }
```

La primera función define si la cámara isométrica está activa o no. La segunda obtiene el valor del zoom el cual va a modificar el área de visión del diorama. La tercera función modifica el zoom, esto debido a que el control del teclado se realiza externamente.

Dentro de la función ProcessKeyboard(Camera_Movement direction, GLfloat deltaTime), se realiza el movimiento de la cámara hacia los lados con WASD. Se establecen límites por cada lado para evitar que la cámara se salga del campo de visión del diorama.

```
else {
    if (direction == FORWARD) {
        iso_up += 0.5f;
        if (iso_up >= 20.0f) iso_up = 20.0f;
    }
    if (direction == BACKWARD) {
        iso_up -= 0.5f;
        if (iso_up <= -25.0f) iso_up = -25.0f;
    }
    if (direction == LEFT) {
        iso_right -= 0.5f;
        if (iso_right <= -20.0f) iso_right = -20.0f;
    }
    if (direction == RIGHT) {
        iso_right += 0.5f;
        if (iso_right >= 20.0f) iso_right = 20.0f;
    }
    iso_position = glm::vec3(iso_right, iso_up, iso_right);
}
```

Para que la cámara 3D guarde el estado de la misma al momento de cambiar de cámara, el movimiento del mouse se aprovecha únicamente cuando la cámara isométrica está desactivada:

```
if (!isometric) {
    this->yaw += xOffset;
    this->pitch += yOffset;
}
```

Dentro de la función glm::mat4 GetViewMatrix(), se obtiene la vista de la cámara, dependiendo de cuál es la que está activa:

```
glm::mat4 GetViewMatrix()
{
    if (isometric == false) return glm::lookAt(this->position, this->position + this->front, this->up);
    else return glm::lookAt(iso_position, iso_position + glm::vec3(1.0f, 0.0f, -1.0f), glm::vec3(0.0f, 1.0f, 0.0f));
}
```

Por último, para que los modelos estén dibujados en la cámara isométrica, se modifica cada matriz model, para realizar dos rotaciones.

```
glm::mat4 ConfIsometric(glm::mat4 model) {
    model = glm::rotate(model, glm::radians(45.0f), glm::vec3(1.0f, 0.0f, 0.0f));
    model = glm::rotate(model, glm::radians(35.2644f), glm::vec3(0.0f, 0.0f, 1.0f));
    return model;
}
```

MainPrueba.cpp

La cámara isométrica utiliza la proyección ortogonal. Dentro del ciclo while, se define que proyección se va a utilizar:

```
if (!camera.getIsometric()) {
    projection = glm::perspective(camera.GetZoom(), (GLfloat)SCREEN_WIDTH / (GLfloat)SCREEN_HEIGHT, 0.1f, 1000.0f);
else {
    projection = glm::ortho(-camera.getIsoZoom(), camera.getIsoZoom(), -camera.getIsoZoom(), camera.getIsoZoom(), -30.0f, 40.f);
```

Dentro del mismo archivo, se usa I para activar la cámara isométrica, y se usa U para desactivarla. Al mismo tiempo, cuando la flecha de arriba se activa, el zoom se incrementa hasta 2.0 (el máximo zoom), y cuando la flecha de abajo se presiona, el zoom se decrementa hasta 30.0f (todo el escenario visible).

```
if (keys[GLFW_KEY_I]) camera.setIsometric(true);
if (keys[GLFW_KEY_U]) camera.setIsometric(false);
if (keys[GLFW_KEY_DOWN]) {
    camera.setZoom(camera.getIsoZoom() + 0.5f);
    if (camera.getIsoZoom() >= 30.0f) camera.setZoom(30.0f);
}
if (keys[GLFW_KEY_UP]) {
    camera.setZoom(camera.getIsoZoom() - 0.5f);
    if (camera.getIsoZoom() <= 2.0f) camera.setZoom(2.0f);
}
```

Como se mencionó, cada modelo usa la función ConfIsometric(model) para que se dibuje correctamente cada modelo en la vista isométrica.

```
model = glm::mat4(1);
if (camera.getIsometric()) model = camera.ConfIsometric(model);
```

ILUMINACION

Para este apartado, se considera la existencia de dos ambientes distintos: el día y la noche. Inspirándose en Super Mario Maker 2, se tomaron como referencia las características de iluminación de ambas versiones para crear la iluminación en el ambiente del castillo.

Durante el día, se optó por simular la presencia de un rayo de luz solar mediante una luz direccional con valores de dirección de (0, -1, -1). Esta luz proporcionará una iluminación principal que proviene de una dirección específica. Por otro lado, la iluminación ambiental se estableció en un valor de (0.9, 0.9, 0.9) para lograr un ambiente colorido y alegre durante el día. En esta etapa, ningún objeto brillará, y todos estarán iluminados de manera colorida para que concuerde con la iluminación diurna. Además, el skybox utilizará imágenes de un cielo con nubes para reflejar la apariencia del ambiente de día.

En contraste, durante la noche, se decidió reducir ligeramente la intensidad de los colores y enfatizarlos en tonos más azules. Para lograr esto, la iluminación ambiental se ajustó a (0.6, 0.6, 0.8) durante la noche. La luz direccional se estableció en (0.0, 0.0, 0.0) porque durante la noche no hay rayos de luz solar directos. En su lugar, la iluminación dependerá principalmente de las luces puntuales que se verán únicamente en este período.

Se emplearán tres Point light para resaltar elementos específicos del entorno nocturno:

- La primera Point light se ubicará en la parte superior del castillo y estará presente independientemente del estado del día o la noche. Esta luz proporcionará una iluminación blanca que simula la luz de una estrella, pero no será tan intensa como la luz solar durante el día. Para lograr este efecto, se utilizarán valores de (1.0, 1.0, 1.0) para la difusa y la especular, mientras que los componentes lineales y cuadráticos se ajustarán a valores cercanos a cero para permitir un mayor alcance de la luz.
- La segunda Point light se colocará encima de la parrilla de Purohuesos para simular la presencia de un fuego encendido. Esta luz tendrá un radio de alcance más limitado, por lo que sus componentes lineales y cuadráticos se establecerán en 5 y 3, respectivamente. El color emitido por esta luz será rojo, utilizando los valores (1.0, 0.0, 0.0), para simular el color del fuego de la parrilla.
- Para la última Point light, se coloca en la espada maestra, la cual usará una luz de color cian con los valores (0.0, 1.0, 1.0), así mismo, la hoja emitirá una luz cian en la parte de la hoja. Cuando se active la animación correspondiente, se encenderá una

SpotLight para iluminar tanto la espada como la piedra, tal como se muestra en los juegos de The Legend of Zelda y se desactivara cuando la animación haya concluido.

Para lograr estos cambios en la iluminación y skybox, se utilizará un contador que, al alcanzar su valor máximo, reiniciará el contador y cambiará el booleano que indica si es de día o de noche.

Se usa dos vectores de Skybox, uno para el día y otro para la noche.

```
//Texturas para SkyBox
vector<const GLchar*> facesDias, facesNoche;

facesDias.push_back("SkyBox/right.tga");
facesDias.push_back("SkyBox/left.tga");
facesDias.push_back("SkyBox/top.tga");
facesDias.push_back("SkyBox/bottom.tga");
facesDias.push_back("SkyBox/back.tga");
facesDias.push_back("SkyBox/front.tga");

facesNoche.push_back("SkyBox(Noche)/right.tga");
facesNoche.push_back("SkyBox(Noche)/left.tga");
facesNoche.push_back("SkyBox(Noche)/top.tga");
facesNoche.push_back("SkyBox(Noche)/bottom.tga");
facesNoche.push_back("SkyBox(Noche)/back.tga");
facesNoche.push_back("SkyBox(Noche)/front.tga");

GLuint cubemapTexture = TextureLoading::LoadCubemap(facesDias);
```

Esto también activará los valores de iluminación adecuados para cada estado, encenderá las Point light cuando se cambie al estado nocturno y las apagará cuando se cambie al estado diurno. Además, cuando el booleano esté en "true" (estado diurno), el skybox se iluminará con las imágenes de nubes en un cielo azul y claro, mientras que cuando esté en "false" (estado nocturno), se utilizarán imágenes del cielo estrellado.

```
if (EstadoDia) GLuint cubemapTexture = TextureLoading::LoadCubemap(facesDias);
else GLuint cubemapTexture = TextureLoading::LoadCubemap(facesNoche);
```

Para cambiar el estado entre día y noche, se usa una estructura de código como la siguiente:

```
LaHora += deltaTime;
if (EstadoDia)
{
    if (LaHora > 59.72f)
    {
        LaHora = 0.0f;
        EstadoDia = false;
        PointlightTrue = 1.0f;
        ambiental1 = 0.6f;
        ambiental2 = 0.8f;
        directionalAmbiental = 0.0f;
        PosteLuzB = 0.0f;
        PosteLuzRG = 1.0f;
        SoundEngineNigth->play2D(bocina2, false);
    }
}
```

Encendido/apagado de Point Lights

De acuerdo con la estructura de código anterior, se modifican ciertas variables las cuales corresponden al cocinado de luces para los postes, esto con el fin de evitar dibujar un gran número de Point Lights pero simulando que las luces están ahí. Los postes se conforman del poste como tal, y de las estrellas que son las que simularán ser focos.

```
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), 0.0f, 0.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), PosteLuzRG, PosteLuzRG, PosteLuzB);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.7f, 0.7f, 0.7f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.9f, 0.9f, 0.9f);
model = glm::mat4(1);
if (camera.getIsometric()) model = camera.ConfIsometric(model);
model = glm::translate(model, glm::vec3(8.854f, 2.559f, 27.501f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Estrellitas.Draw(lightingShader);
```

Además, se cuenta con luces que iluminan la estrella, la parrilla, y la espada. La definición de cada Point Light toma como argumentos para las componentes difusa y especular las variables modificadas en el ciclo día/noche, con lo cual se logra el efecto de encendido/apagado.

```
// Point light 1
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].position"), pointLightPositions[0].x, pointLightPositions[0].y, pointLightPositions[0].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].ambient"), 0.05f, 0.05f, 0.05f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].diffuse"), 0.0f * PointlightTrue, 1.0f * PointlightTrue, 1.0f * PointlightTrue);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].specular"), 0.0f * PointlightTrue, 1.0f * PointlightTrue, 1.0f * PointlightTrue);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].linear"), 0.054f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].quadratic"), 0.0192f);
```

Encendido/apagado de SpotLights

Para encender y apagar las SpotLights, se utilizan las teclas 2 y 3:

```
if (keys[GLFW_KEY_2]) {
    Reflectores = 1.0f;
}
if (keys[GLFW_KEY_3]) {
    Reflectores = 0.0f;
}
```

Para los reflectores, se usó la técnica de “Iluminación cocinada” en el espacio del reflector que se ilumina cuando la SpotLight se enciende. Para esto, el modelo fue separado en dos partes, uno que consta de la parte metálica del reflector y otra que corresponde al foco del reflector. Mediante variables, se modifica las componentes de la dirección, la ambiental, difusa y especular, y con esto se dibuja el foco de cada reflector.

```
if (Reflectores>0) {
    glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), 0.0f, 0.0f, 0.0f);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), 2.0f, 2.0f, 2.0f);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.7f, 0.7f, 0.7f);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.9f, 0.9f, 0.9f);
}
if (Reflectores < 1.0) {
    glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), 0.0f, -directionalAmbiental, -directionalAmbient
    glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), ambiental1, ambiental1, ambiental2);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.2f, 0.2f, 0.2f);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.1f, 0.1f, 0.1f);
}
model = glm::mat4(1);
if (camera.getIsometric()) model = camera.ConfIsometric(model);
model = glm::translate(model, glm::vec3(8.213f, 0.595f, 21.701f));
model = glm::scale(model, glm::vec3(-1.0f, 1.0f, 1.0f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Foco.Draw(lightingShader);
```

La definición de cada SpotLigth toma como base las variables de los reflectores (debido a que la luz de la espada depende de otras variables explicadas más adelante). Con esto, cuando se presione 2, la SpotLight se dibujará, y con 3 no.

```
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[1].position"), SpotLightPositions[1].x, SpotLightPositions[1].y, SpotLightPositions[1].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[1].direction"), SpotLightDirections[1].x, SpotLightDirections[1].y, SpotLightDirections[1].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[1].ambient"), Reflectores, Reflectores, Reflectores);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[1].diffuse"), Reflectores * 0.1f, Reflectores * 0.1f, Reflectores * 0.1f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[1].specular"), 1.0f, 0.0f, 0.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[1].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[1].linear"), 0.1f);//0.09
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[1].quadratic"), 0.01f);//0.032
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[1].cutOff"), glm::cos(glm::radians(12.5f)));
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[1].outerCutOff"), glm::cos(glm::radians(15.0f)));
```

Iluminación en cámara isométrica

Debido a que la cámara isométrica se basa en rotaciones de los modelos en dos ejes, las luces también deben ajustarse para que se muestren correctamente. Esto se logra pasando la

posición de las Point Lights, y la posición y dirección de las Spot Ligths a la función ConfIsometric(model).
Se comienza definiendo la matriz de posiciones de cada Point Light:

```
// Positions of the point lights
glm::vec3 pointLightPositions[] = {
    glm::vec3(6.766,0.670,24.985),
    glm::vec3(0.0f, 9.751, 6.016),
    glm::vec3(0,1.0,19.892)
};

// Positions of the Spot lights
glm::vec3 SpotLightPositions[] = {
    glm::vec3(6.766f, 5.0f, 24.985f),
    glm::vec3(8.099f, 1.002f, 21.799f),
    glm::vec3(-8.099f, 1.002f, 21.799f)
};

// Directions of the Spot lights
glm::vec3 SpotLightDirections[] = {
    glm::vec3(6.766f, 5.0f, 24.985f),
    glm::vec3(-8.099f, 2.909f, 3.455f),
    glm::vec3(8.099f, 2.909f, 3.455f)
};
```

Las luces en su definición toman los valores de los arreglos anteriores. Cuando la cámara isométrica está desactivada, las posiciones se conservan:

```
if (!camera.getIsometric()) {
    projection = glm::perspective(camera.GetZoom(), (GLfloat)SCREEN_WIDTH / (GLfloat)SCREEN_HEIGHT, 0.1f, 1000.0f);
    pointLightPositions[0] = glm::vec3(6.766, 0.670, 24.985);
    pointLightPositions[1] = glm::vec3(0.0f, 9.751f, 6.016f);
    pointLightPositions[2] = glm::vec3(0, 1.0, 19.892);

    SpotLightPositions[0] = glm::vec3(6.766f, 5.0f, 24.985f);
    SpotLightPositions[1] = glm::vec3(8.099f, 1.002f, 21.799f);
    SpotLightPositions[2] = glm::vec3(-8.099f, 1.002f, 21.799f);

    SpotLightDirections[0] = glm::vec3(0.0f, -1.0f, 0.0f);
    SpotLightDirections[1] = glm::vec3(-8.099f, 2.909f, 3.455f);
    SpotLightDirections[2] = glm::vec3(8.099f, 2.909f, 3.455f);
}
```

Cuando la cámara isométrica está activa, se modifican las posiciones:

```
else {
    projection = glm::ortho(-camera.getIsoZoom(), camera.getIsoZoom(), -camera.getIsoZoom
    glm::mat4 model(1);
    model = camera.ConfIsometric(model);
    pointLightPositions[0] = glm::vec3(model * glm::vec4(6.766, 0.670, 24.985, 1.0f));
    pointLightPositions[1] = glm::vec3(model * glm::vec4(0.0f, 9.751f, 6.016f, 1.0f));
    pointLightPositions[2] = glm::vec3(model * glm::vec4(0, 1.0, 19.892, 1.0f));

    SpotLightPositions[0] = glm::vec3(model * glm::vec4(6.766, 5.0f, 24.985, 1.0f));
    SpotLightPositions[1] = glm::vec3(model * glm::vec4(8.099f, 1.002f, 21.799f, 1.0f));
    SpotLightPositions[2] = glm::vec3(model * glm::vec4(-8.099f, 1.002f, 21.799f, 1.0f));

    SpotLightDirections[0] = glm::vec3(model * glm::vec4(0.0f, -1.0f, 0.0f, 1.0f));
    SpotLightDirections[1] = glm::vec3(model * glm::vec4(-8.099f, 2.909f, 3.455f, 1.0f));
    SpotLightDirections[2] = glm::vec3(model * glm::vec4(8.099f, 2.909f, 3.455f, 1.0f));
}
```

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 valore 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"**.

Animaciones sencillas

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.

Para evitar hacer transformaciones combinadas manualmente se optó 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. Primero el Goomba rotara su cabeza de lado a lado sin realizar ningún otro movimiento; cuando el usuario presione la tecla correspondiente, 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 y un booleano que determina cuando activar dicha animación. 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 atraviese la superficie.

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 Z, mientras que simultáneamente aumentan su posición en el eje Y cuando su rotación es 0 grados, creando un efecto de saltitos. Para esto se hacen aumentos y decrementos que controlan tanto la rotación como el brinco de la flor, mientras su dirección está siendo controlado por un booleano.

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.

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.

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 todas que se encuentran en el cuarto del trono.

Planta Piraña mordiendo

La animación de la planta piraña comienza con su posición inicial estática dentro de la tubería. Utilizando una variable de incremento y decremento en el eje Y, la planta piraña realiza un suave movimiento dentro de la tubería. Mientras tanto, al presionar el botón correspondiente, la planta piraña se eleva y desciende en el eje Y, mientras su tallo cambia de tamaño para simular un estiramiento. Durante este desplazamiento, la planta piraña realiza

su icónica animación de abrir y cerrar la boca. La implementación de esta animación se basa en el uso de variables para controlar diferentes aspectos. Una variable de posición en Y se incrementa y decrementa para que la planta piraña se mueva a lo largo de su recorrido. Al mismo tiempo, una variable de escala se ajusta para que el tallo parezca estar estirándose a medida que la planta se mueve. Además, se utiliza una variable de conteo que varía de 0 a 1 y se reinicia rápidamente. Cada vez que ocurre este reinicio, la rotación de la planta piraña cambia de 0 a 45 grados y viceversa. Cuando termina su recorrido y vuelve a la tubería se quedará estática como en el principio de la descripción para que el usuario pueda volver a activar su animación.

Bob-omb marchando

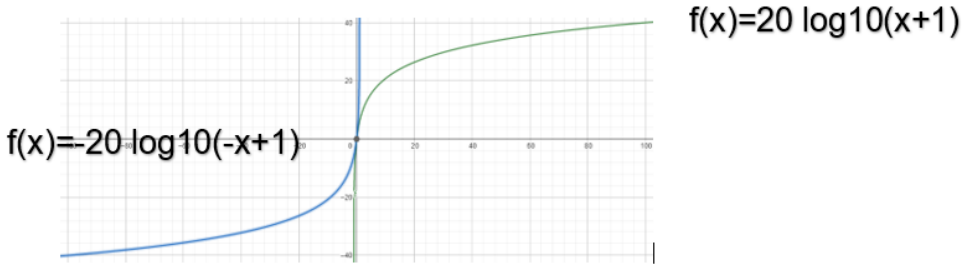
La bob-omb inicialmente se encuentra en una posición estática y quieta hasta que el usuario activa su animación mediante el botón correspondiente. Una vez que se inicia la animación, la manecilla de la bob-omb comienza a rotar en el eje Z mientras las patitas suben y bajan. Al mismo tiempo, la cabeza de la bob-omb rota en el eje Y para darle más dinamismo al movimiento.

Para lograr esta animación, se utilizan variables incrementables y decrementadles controladas por booleanos que afectan la posición de los pies y la cabeza de la bob-omb. Asimismo, se emplea una perilla que incrementa de 0 a 360 y luego reinicia su ciclo. Cuando el tiempo de animación alcanza su límite, la bob-omb vuelve a su posición inicial y se vuelve estática nuevamente, esperando que el usuario active la animación una vez más.

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 enérgico. 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.

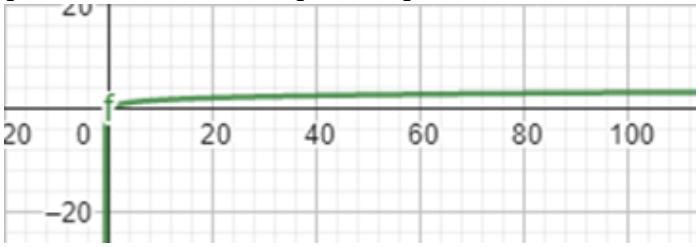


Durante la animación, el brazo de la posición adelantada se mantiene por un breve momento extendido hacia adelante, antes de realizar el cambio de posición. Este efecto se logra

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.

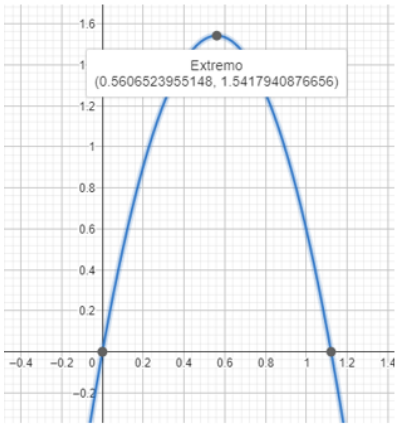
Paratroopa volando dentro y fuera del cuadro

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. Como adicional, se usó un shader para que la pintura a la que atraviesa se mueva, dando un efecto de gelatina, recreando la animación y lógica de los cuadros de Mario 64, usando siempre como referencia la posición del vuelo del paratroopa.



Champiñón dentro del bloque

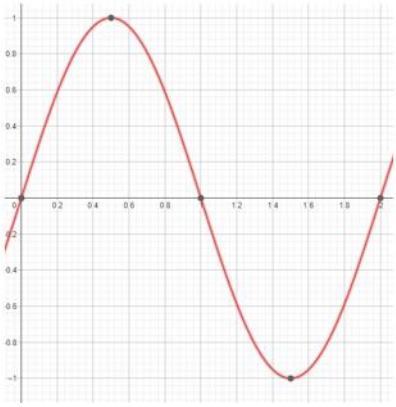
El champiñón es disparado desde un question block y se lanza hacia arriba siguiendo las reglas de la física. A medida que asciende, va perdiendo velocidad hasta alcanzar su punto más alto y luego comienza a caer. Durante este proceso, el cubo se escalará, volviéndose más pequeño, mientras que otro cubo 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.



$$f(x) = 5.5\,x - 0.5 * 9.81\,x^2$$

Mario Tanooki volando

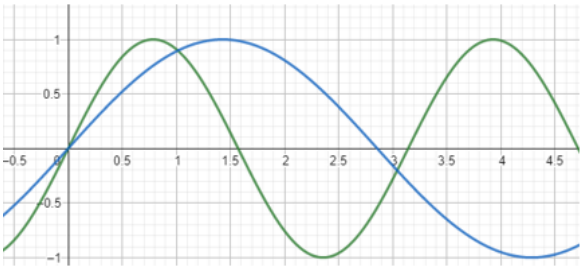
Mario asciende y desciende mientras mueve su colita arriba y abajo, lo cual se logra mediante una función Sen(x) que determina el aumento y disminución de su altura. Esta función crea un efecto suave y fluido en el movimiento vertical de Mario. Para evitar el llenado de memoria la variable X se reinicia 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.



$$f(x) = sen(x * 3.1416)$$

Luigi jugando con una Nintendo Switch

En la animación de Luigi jugando con su Nintendo Switch, el personaje se encuentra dentro del castillo mientras se divierte con la consola. Mientras juega, Luigi mueve sus brazos y cabeza de un lado a otro para simular que está utilizando el giroscopio de la consola. Sus pies no rotan, pero se mueven ligeramente de arriba a abajo para simular el efecto del balanceo que realiza con sus brazos. Para lograr este efecto, se utilizan dos funciones seno de x. Cada variable aumenta a diferente velocidad, lo que provoca que los brazos y la cabeza de Luigi giren a diferentes ritmos. Sin embargo, al estar construidos en una jerarquía, esto crea un movimiento más fluido y realista en la animación de Luigi jugando.



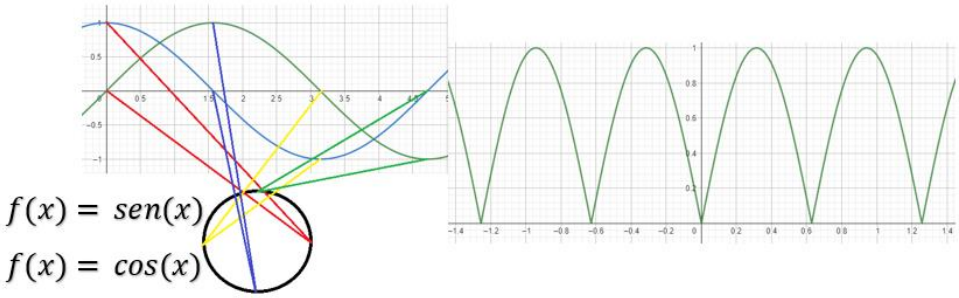
$$f(x) = sen(x)$$

$$f(Y) = sen(Y)$$

El resultado es que Luigi parece estar sumergido en su juego, moviendo sus brazos y cabeza de manera dinámica mientras disfruta de la experiencia de juego con su Nintendo Switch. El ligero movimiento de sus pies de arriba a abajo agrega un toque adicional de realismo y naturalidad al movimiento general.

Chilly Willy escapando de Chain Chomp

En la animación de Chilly Willy escapando de Chain Chomp, el personaje Chilly Willy estará conduciendo un vehículo de Mario Kart alrededor de un arbusto con forma de Mario. Detrás de él, un Chain Chomp seguirá la misma ruta mientras abre y cierra la boca y da pequeños brincos característicos de los juegos de Mario Kart. Esto generará una persecución en círculos alrededor del arbusto con forma de Mario. Para lograr esta animación, se utilizan funciones seno y coseno en cada objeto involucrado. Estas funciones se utilizan para aprovechar el principio de un círculo. Ambas fórmulas reciben los mismos valores, pero generan resultados diferentes. Al aplicar esta transformación a las coordenadas X y Z de los objetos, se traza un círculo en el plano. Para los brincos del Chain Chomp, se utiliza solo una función seno. La variable utilizada en esta función solo varía de 0 a un valor que resulta en 0, evitando valores negativos y generando así los brincos característicos. Este enfoque garantiza que los brincos se mantengan en una trayectoria ascendente y descendente. Finalmente, para lograr que los objetos giren en el eje Y de acuerdo a la posición que llevan, se utiliza una variable auxiliar. Cuando esta variable alcanza su valor máximo, se multiplica por un número que resulta en 360 grados. Como 0 y 360 grados son equivalentes, no se nota el reinicio de los valores y la animación se repite una y otra vez.



$$f(x) = sen(x)$$

$$f(x) = cos(x)$$

Así se logra una animación cautivadora en la que Chilly Willy escapa del Chain Chomp mientras circulan en círculos alrededor del arbusto con forma de Mario. Los brincos y movimientos característicos de los personajes agregan dinamismo y diversión a la animación.

Cheep Cheep nadando

Para lograr este efecto, se utiliza una función seno que determina el movimiento ascendente y descendente del pez. Esta función seno crea una trayectoria suave y ondulante, generando la sensación de nado en el agua. El Cheep Cheep sube y baja en armonía con esta función, imitando los movimientos de un pez nadando. Además, las aletas del Cheep Cheep también

están animadas y se mueven en función del mismo patrón seno. Al estar construidas en una jerarquía, las aletas siguen los movimientos generales del Cheep Cheep, dando la ilusión de que las usa para nadar y aportando un mayor realismo a la animación.

Princesa Peach bailando con las monedas

se presentarán 8 monedas que formarán un círculo y rotarán rápidamente, similar a la animación de Chilly Willy. Mientras las monedas giran, Peach estará realizando una rotación constante de 360 grados mientras se mueve en forma de elipse, simulando un baile.

Se reutiliza el principio de la fórmula para trazar un círculo utilizando funciones seno y coseno. Sin embargo, en este caso, la fórmula se repite continuamente, pero se le cambia el valor dentro de la función seno. Debido a la naturaleza cíclica y continua de esta función, no importa si el valor es positivo o negativo. Esto permite que todas las monedas recorran el mismo círculo, pero cada una en una posición diferente, formando así el aro de monedas. Peach seguirá el mismo principio, pero en uno de los ejes, la distancia entre los puntos será menor para crear la forma de elipse. Esto le dará un movimiento más dinámico y estilizado a su baile en el aro de monedas. Para la rotación de Peach, se utilizará la misma variable auxiliar mencionada anteriormente, similar a la animación de Chilly Willy. Sin embargo, en este caso, se aumentará la velocidad de la variable para que los giros de Peach no se completen a la misma velocidad que las monedas se mueven sobre la ruta. Esto añadirá un efecto visual interesante y diferenciará los movimientos de Peach del resto de la animación.

Puro Huesos haciendo hamburguesas

se presentará Puro Huesos frente a una parrilla sosteniendo una espátula. Mientras sostiene la espátula, Puro Huesos moverá su brazo de manera jerárquica, al mismo tiempo que una hamburguesa es lanzada al aire utilizando las propiedades de un tiro vertical. La hamburguesa volverá a caer en la parrilla y Puro Huesos volverá a colocar la espátula debajo de la hamburguesa para lanzarla nuevamente. Habrá un tiempo de espera entre cada inicio de la animación para dar la sensación de que está cocinando las hamburguesas.

Para lograr esta animación, se utilizará una animación simple para el brazo de Puro Huesos y una animación más compleja para la hamburguesa. Se empleará un incrementable para controlar el tiempo, y cuando alcance un punto determinado, se aumentará o disminuirá la rotación del brazo y el antebrazo para simular el movimiento de dar vuelta a la hamburguesa en la parrilla. La animación de la hamburguesa servirá como una especie de "bandera" para indicar cuándo debe realizarse el lanzamiento vertical. Una vez que la hamburguesa cae, se utilizará nuevamente como "bandera" para indicar al brazo cuándo debe bajar y repetir el proceso. Este ciclo continuo de lanzamiento de hamburguesas y volteo de Puro Huesos creará una animación divertida y dinámica que simula el proceso de cocinar hamburguesas.

Koopas esperando su hamburguesa

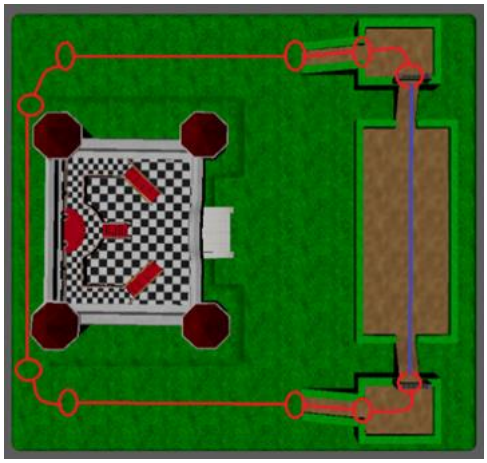
se agregarán tres Koopas frente a Puro Huesos. Cada Koopa estará rotando su cuerpo y moviendo sus brazos alegremente, simulando que están esperando su hamburguesa y generando un ambiente de clientela para Puro Huesos.

Para lograr esto, cada Koopa será construido de forma jerárquica, de modo que el caparazón y los brazos puedan rotar y generar un movimiento fluido y alegre. Se utilizará un ciclo de funciones seno para controlar estos movimientos, lo que les dará una sensación de alegría y entusiasmo mientras esperan su hamburguesa. Esta adición de los Koopas crea un ambiente

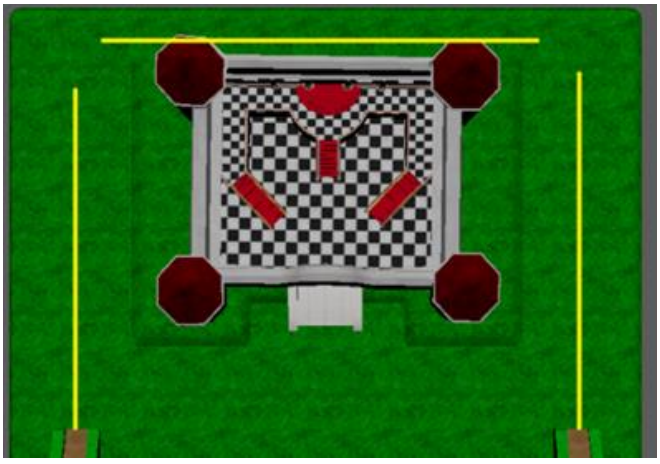
animado y festivo en la escena, aportando vida y dinamismo al entorno de Puro Huesos mientras prepara unas hamburguesas.

Conduciendo un kart

En esta animación, varios karts estarán recorriendo un circuito alrededor del castillo. Cada kart realizará diferentes acciones, como dar vueltas en las curvas correspondientes, subir y bajar rampas en el jardín, y volar con planeadores sobre dicho jardín, utilizando elementos característicos de los juegos de Mario Kart 8.

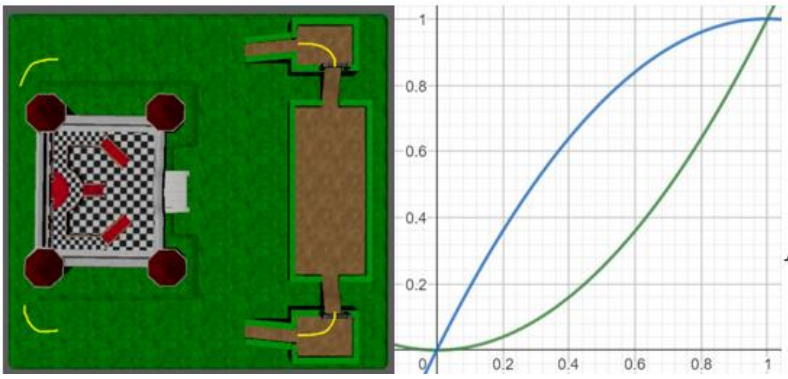


Esta animación es la más compleja, ya que se ha desarrollado un sistema de 10 banderas que permitirá a los karts recorrer todo el circuito de manera fluida. Para lograr esto, se han establecido tres rutas principales que consisten en aumentar o disminuir los valores de X y Z para las secciones rectas del circuito.



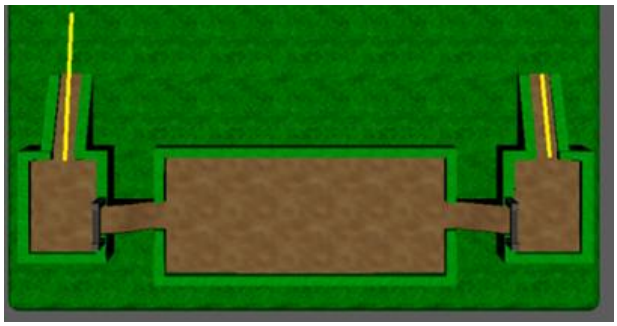
$$f(x) = 1 - (x-1)^2$$

Cuatro banderas se utilizarán para realizar giros suaves en el circuito. Para ello, se ha empleado la propiedad de las funciones cuadráticas, utilizando la fórmula x^2 y 1-(x-1)^2. Estas funciones se aplican a las coordenadas X y Z, respectivamente, permitiendo que el kart disminuya su velocidad de forma gradual al acercarse a una bandera, y luego la aumente gradualmente después de pasarla, logrando así curvas suaves y realistas.

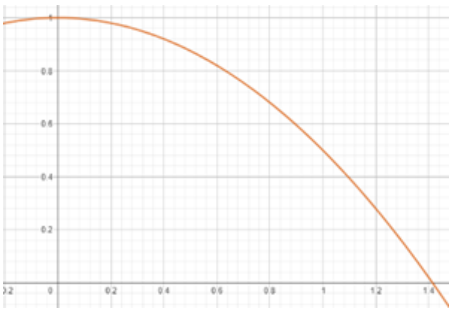


$$f(x) = x^2$$

Durante estos cambios de velocidad y dirección, se ajustará la rotación del kart para simular los giros que realiza en el circuito, creando una sensación de conducción más auténtica.
Además, se han agregado banderas especiales para simular subidas y bajadas en el circuito. Al aproximarse a una bandera de subida, se incrementará el valor de Z mientras se aumenta también la coordenada Y para simular que el kart sube por una rampa en el jardín. Al final de la rampa, se ajustará nuevamente la rotación para indicar que el kart ha completado la subida.

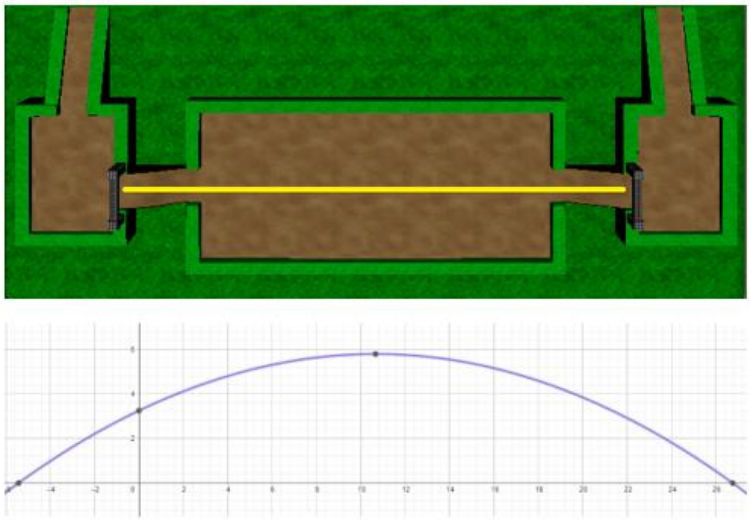


Por otro lado, al acercarse a una bandera de bajada, se optó por hacer que el kart baje mientras continúa avanzando rápidamente en la coordenada Z y desciende en la coordenada Y utilizando la fórmula de caída libre bajo la influencia de la gravedad. Durante este descenso, el kart rotará ligeramente para simular una suspensión en el aire y, al alcanzar Y = 0, se restaurarán las posiciones iniciales para simular el impacto con el suelo y continuar con el recorrido.



La última bandera se utilizará para simular el uso de los planeadores de Mario Kart 7 y 8. El kart realizará un giro de 380 grados para simular una acrobacia y luego abrirá su paracaídas para volar. Durante el vuelo, el kart se desplazará a lo largo del eje X, mientras que el eje Y seguirá una trayectoria en forma de parábola para simular la sensación de vuelo. Al final del vuelo, se ajustarán los grados adicionales para complementar el efecto de aterrizaje del kart.

También se menciona que en una de las banderas de curva se añade la escala del paracaídas a 0 para simular que se guarda en el kart.



$$f(x) = 2.283 - 0.02233919\,x^2 + 0.47640569\,x + 0.96004303$$

Para dar más vida al entorno, se han agregado un total de cuatro karts que recorrerán el circuito. Estos karts representan personajes de diferentes franquicias, como Mulan, Bob Esponja, Goku y Ladybug. Cada kart ha sido construido de forma jerárquica para permitir que las ruedas giren con libertad y sigan la posición del kart. Del mismo modo, el paracaídas se ajustará en la parte trasera del kart correspondiente en el momento indicado, creando un efecto visual dinámico y realista.

Con todas estas acciones combinadas, se logrará una animación compleja y emocionante, en la que los karts recorren el circuito alrededor del castillo, realizando movimientos y acciones propias de los juegos de Mario Kart 7 y 8.

Animacion por Keyframes

La espada maestra

Cuando se active la animación de la Espada Maestra, se realizarán una serie de acciones para dar vida a este efecto. Primero, se encenderá una SpotLight que iluminará tanto la espada como la piedra en la que está clavada. Esta luz resaltará los detalles de la espada y creará un ambiente misterioso alrededor de la piedra. A continuación, la espada se elevará desde su posición inicial para iniciar un recorrido en forma de círculo alrededor de la piedra. Durante este recorrido, la espada girará sobre su eje Y para añadir un efecto visual dinámico y llamativo. Una vez que la espada haya completado su recorrido y haya vuelto a su posición original, descenderá gradualmente para volver a clavarse en la piedra. Esta acción añade un elemento dramático a la animación, simulando el regreso de la espada a su lugar de reposo. Finalmente, al finalizar la animación, se apagará la SpotLight que ilumina la espada y la piedra, lo que contribuirá a crear un ambiente misterioso y enigmático en la escena.

Para lograr lo anterior, los KeyFrames se almacenan en un archivo de texto. Al ejecutar el programa, se comienza leyendo el archivo de texto mediante una función implementada:

```
void readFile() {   // funcion para leer archivo que contiene los keyframes
    int i = 0, linead = 0;
    std::string indice, valor, content;
    int index;
    float value;
    std::ifstream fileStream("KeyFrames.txt", std::ios::in);

    if (!fileStream.is_open()) {
        printf("Failed to read %s! File doesn't exist.", "KeyFrames.txt");
        content = "";
    }
```

Dentro de esta función, se lee cada línea del archivo y se extrae el valor para cada eje en la posición y la rotación:

```
std::string line = "";
while (!fileStream.eof()) {
    std::getline(fileStream, line);

    if (linead < 10) indice = line.substr(9, 1);
    else indice = line.substr(9, 2);
    index = std::stoi(indice);
    switch (i) {
    case 0: //posicion en X
        if (linead < 10) valor = line.substr(19, line.size() - 21);
        else valor = line.substr(20, line.size() - 22);
        value = std::stof(valor);
        KeyFrame[index].posX = value;
        std::cout << "KeyFrame[" << index << "].posX = " << value << ";" << std::endl;
        i++;
        break;
```

El Switch itera sobre 4 casos: cuando se lee la posición en X, Y, Z o la rotación del objeto, a su vez, se cuida extraer correctamente cada subcadena. Esto se logra con ayuda de las variables indice, y el contador i.

Dentro del código ya se contaba con las funciones que realizan la interpolación lineal para la animación, y el reseteo de las variables a los valores del primer frame.

Finalmente, se agregó la activación de la animación con 1 y el reseteo de la animación con 0.

```
if (keys[GLFW_KEY_1]) {
    if (reproduciranimacion < 1) {
        if (play == false && (FrameIndex > 1)) {
            espadaLuz = 1.0f;
            resetElements();
            //First Interpolation
            interpolation();
            play = true;
            playIndex = 0;
            i_curr_steps = 0;
            reproduciranimacion++;
            printf("\n presiona 0 para habilitar reproducir de nuevo la animación'\n");
            habilitaranimacion = 0;
        }
        else play = false;
    }
}
if (keys[GLFW_KEY_0]) {
    if (habilitaranimacion < 1) reproduciranimacion = 0;
}
```

SpotLigth unida a la animación

Para que la animación se vea de mejor forma, la primer Spot Light declarada se activa cuando la animación por KeyFrames inicia y permanece activada hasta que la animación termine.

Esto se ve en la variable espadaLuz, la cual toma el valor de 1 para que sea asignada como componente ambiental y difusa de la luz.
Dentro de la función animateKeyFrame(void), cuando se llegue al último Frame, la variable espadaLuz se asignará a 0.

```
void animateKeyFrame(void) {
    //Movimiento del objeto
    if (play) {
        if (i_curr_steps >= i_max_steps) {//end of animation between frames?
            playIndex++;
            printf("playindex : %d\n", playIndex);
            if (playIndex > FrameIndex - 2) {//end of total animation?
                printf("Frame index= %d\n", FrameIndex);
                printf("termina anim\n");
                espadaLuz = 0.0f;
                playIndex = 0;
                play = false;
            }
            else {//Next frame interpolations
                i_curr_steps = 0; //Reset counter
                interpolation();
            }
        }
```

Diccionario de variables para las animaciones

Funciones	Explicación
main	La función main es el punto de entrada a un programa en C++, el código establece el entorno y las configuraciones iniciales para un programa gráfico utilizando GLFW y OpenGL. Se crea una ventana, se inicializan bibliotecas y se cargan shaders y modelos 3D. Luego, se entra en un bucle principal donde ocurre el renderizado y la interacción con el usuario.
animacion	En esta función se declaran todas las condiciones que presenta cada objeto para realizar una acción al activar o desactivar una variable.
KeyCallback	Esta función define que al presionar la tecla declarada dentro de la función hará la acción correspondiente que se define en cada paso.
MouseCallback	Se codifica la forma en la que va a actuar el mouse cuando se esté dentro de la ventana y las acciones que tendrá al hacer alguna interacción con la misma.
DoMovement	En este bloque de código se define la forma que va a actuar el código en caso de que se presiona una tecla, y como su nombre lo dice, va a generar movimiento.

Variable	Tipo	Funcionalidad
LaHora	float	Contador que sirve para medir la duración del día y la noche durante la ejecución.
EstadoDia	bool	Bandera que indica al programa en qué estado del día se encuentra el lugar (día o noche).
PointlightTrue	float	Multiplicador que ayudara a encender y apagar los pointlights.
ambiental1 ambiental2	float	Valores que ayudan a cambiar el color utilizado en los modelos para que parezcan que están en el día o en la noche.
directionalAmbiental	float	Variable que ayuda a poner durante el día o quitar durante la noche la direccional de la ambiental para un mayor realismo.
PosteLuzRG PosteLuzB	float	Valores que ayudan a cambiar el color utilizado en los modelos de poste de luz para que parezcan prendidos o apagados.
espadaLuz	float	Variable que ayuda a iluminar la hoja de la espada durante el día.
Reflectores	float	Variable que aumenta o disminuye la iluminación de los reflectores dependiendo si están prendidos o apagados.

VariableRadioGiros VariableRadioBrincos	float	Variables auxiliares que aumentaran constantemente y ayudan al movimiento de chillywilly y al chain chomp
TraslacionXChilly TraslacionZChilly	float	Calcularan la posición en X y Z del auto de chilly Willy usando la fórmula de circunferencia.
TraslacionXChainChomp TraslacionZChainChomp	float	Misma funcionalidad, pero para el chainchomp con su propia variable.
TraslacionYChainChomp	float	Calcula la altura de los brincos del chain chomp con el uso de la función seno
VariableJuego VariableCabezaLuigi	float	Variables auxiliares que ayudaran a mover a Luigi a diferente ritmo sus brazos y cabeza.
MovimientoJuego MovimientoCabezaLuigi	float	Calculan la rotación de los brazos y la cabeza de Luigi a diferentes ritmos.
VariablePeachGiros	float	Variable giro que ayuda a calcular la posición y giro de Peach
TraslacionXPeach TraslacionZPeach	float	Calcula la posición en X y Z para que Peach gire al en un ovalo dentro de las monedas.
VariableMonedasGiros	float	Variable que ayuda al cálculo de la posición de las monedas y su rotación.
TraslacionXMonedas TraslacionZMonedas TraslacionXMonedas1 TraslacionZMonedas1 TraslacionXMonedas2 TraslacionZMonedas2 TraslacionXMonedas3 TraslacionZMonedas3 TraslacionXMonedas4 TraslacionZMonedas4 TraslacionXMonedas5 TraslacionZMonedas5 TraslacionXMonedas6 TraslacionZMonedas6 TraslacionXMonedas7 TraslacionZMonedas7	float	Calcula la posición de cada moneda para que cada una tenga su propia posición, pero sigan la misma ruta, creando así un aro hecho con monedas que dan vueltas en una misma ruta alrededor de Peach.
RotCaparazon	float	Incrementable y decrementable que controla la dirección de movimiento de los koopas.
sentidoCaparazon	bool	Determina el sentido y el límite de la rotación de los koopas para su movimiento.
CheepFloat	float	Variable auxiliar que aumentará o disminuirá la altura del cheep cheep.
AlturaNadado	float	Calcula la posición en Y del cheep cheep para su nadado dentro de la bolsa.
AletasCheep	float	Variable que incrementa y deceremenda para rotar las aletas del cheep cheep.

IDLE	float	Variable incrementable y decrementable que controla todas las animaciones estáticas.
sentidoIDLE	bool	Bandera que controla el sentido de la animación idle
RotBrazo	float	Variable incrementable y decrementable que controla la rotación de los brazos de los todas.
sentidoBrazo	bool	Bandera que controla el sentido de la rotación de los brazos de los todas
Giro	float	Calcula la posición de un toad para que rote haciendo uso de la propiedad de la función logarítmica.
Brinco	float	Variable que controla la pequeña altura que gana el toad cuando realiza su animación de mover los brazos.
auxilar1 auxilar2 auxilar3	float	Variables incrementables/decrementables que apoya a ciertas animaciones con sus intervalos de números, cada uno a su propio ritmo
Sentido1	bool	Bandera que controla el sentido del auxiliar 1.
Sentido2	bool	Bandera que controla el sentido del auxiliar 2.
RoatParatroopa	float	Variable que controla la rotación del paratroopa para dar dirección a si entra o sale del cuadro.
VueloParatroopa	float	Controla la posición del paratroopa mientras hace su recorrido dentro y fuera de la pintura.
Rotacion2	float	Variable que incrementa cíclicamente para controlar las rotaciones de las hélices presentes en el diorama.
Times	float	Variable que ayuda al shader de animación a crear el efecto de oleaje en la pintura del castillo.
Reducir	float	Variable que controla el movimiento del oleaje para que solo se mueva cuando el paratroopa entra en contacto con el cuadro.
Hamburguesa	bool	Bandera que indica el inicio de la animación del lanzamiento de la hamburguesa.
Espera	float	Contador que indicara cuando avanzar a la siguiente etapa de la animación de purohuesos.
SentidoHamburguesa	bool	Bandera que indica la dirección del brazo de purohueso que sostiene la espátula.
roatAnteBrazo	float	Variable que controla la rotación de su brazo.
TiempovueloHamburguesa	float	Variable auxiliar que mide el tiempo de vuelo de la hamburguesa.
HamburVuelo	float	Valor que calcula la altura de la hamburguesa mediante la fórmula del tiro vertical.
AnimacionGoomba	bool	Bandera que controla la animación de la caminata Goomba.
CaminataGoomba	float	Variable incrementable/decrementable que controla la distancia que recorre el Goomba.

SentidoGoomba	bool	Bandera que controla el sentido en el que se mueve el goomba.
rotCabezaGoomba rotPieDerGoomba rotPieIzqGoomba	float	Variables incrementables/decrementables que controlan las rotaciones de cada parte del cuerpo del goomba al caminar
RotSentidoGoomba	bool	Controla el sentido en el que el goomba rota sus pies y cabeza.
RoatFlyGuy TrasladoFlyguy	float	Variables que controlan la traslación e inclinación del flyguy mientras vuela.
SentidoFlyGuy	bool	Bandera que controla el sentido del vuelo del flyguy
RoatBomba RoatLlave pieIzqBomba pieDerBomba	float	Variables que controlan cada parte de la bobomba para rotar la cabeza y llave mientras traslada los pies para simular la marcha característica de los bob omb.
TiempoanimaiconBomba	float	Incrementable que controla el tiempo de duración de la marcha de la bob omba.
MarchaBomba	Bool	Bandera que controla que pie sube y cual baja para simular la marcha de la bob omba.
AnimacionBomba	Bool	Bandera que indica cuando activar la animación de la bomba.
piranaPlant	float	Variable incrementable/decrementable que sube o baja a la planta piraña para simular que entra y sale de su tubo.
PiranaplantSentido	bool	Determina el sentido en que la planta se dirige.
tiempoMordida	float	Contador que mide el tiempo que mantiene su boca en una posición.
AnguloMorida	float	Variable que cambia el ángulo de la boca de 0 a 45 grados, el cual depende del contador antes mencionado.
AnimPlantaPriaña	bool	Bandera que controla cuando se activa la animación de la planta piraña.
LadyBugmovKitX LadyBugmovKitZ LadyBugmovKitY GokumovKitX GokumovKitZ GokumovKitY BobEsponjamovKitX BobEsponjamovKitZ BobEsponjamovKitY MulanmovKitX MulanmovKitZ MulanmovKitY	float	Variables que controlan las posiciones en X,Y y Z de cada uno de los karts, los cuales pueden ser incrementables/decrementables o formulas. Para identificar qué valor corresponde cada uno, se usa el nombre del personaje al principio de cada variable.
LadyBugrotKit		

LadyBugGiroExtremo GokurotKit GokuGiroExtremo BobEsponjarotKit BobEsponjaGiroExtremoM ulanrotKit MulanGiroExtremo	float	Variables de para cada corredor que controla el grado de giro en los ejes Y y X para dar mayor realismo al manejo de los coches.
LadyBugTiempoDelta GokuTiempoDelta BobEsponjaTiempoDelta MulanTiempoDelta	float	Auxiliar que se usa en las curvas para que se realicen las vueltas con mayor naturalidad al pasarlas a dos fórmulas diferentes, lo que hará que se reduzca la velocidad en un eje y aumente en otro.
LadyBugrecorrido1 LadyBugrecorrido2 LadyBugrecorrido3 LadyBugrecorrido3Subida LadyBugrecorrido4 LadyBugrecorrido5 LadyBugrecorrido6 LadyBugrecorrido6Descens LadyBugrecorrido7 LadyBugrecorrido8	bool	Banderas que marcan el final e inicio de un recorrido, el cual en conjunto forman el circuito completo, cada uno maneja una lógica diferente para que el coche correspondiente realice ciertas acciones antes de pasar al siguiente recorrido. La razón por la cual cada personaje tiene su propio recorrido, pero usando la misma base es para que se inicialicen en diferentes puntos y hagan exactamente la misma acción para darle más dinamismo y realismo al escenario.
LadyBugapertura LadyBugCierre Gokuapertura GokuCierre BobEsponjaapertura BobEsponjaCierre	bool	Banderas que indican el momento en que los paracaídas de cada personaje se escalan a 1 o a 0 en los momentos en que el carro quedo suspendido en el aire y vuelve a aterrizar al suelo.
LadyBugXscale LadyBugYscale LadyBugZscale GokuXscale GokuYscale GokuZscale BobEsponjaXscale BobEsponjaYscale BobEsponjaZscale MulanXscale MulanYscale MulanZscale	float	Variables que cambian la escala de cada paracaídas, las cuales se encargaran de ir de 0 a 1 y viceversas dependiendo de la bandera del personaje.
LadyBugDescender GokuDescender BobEsponjaDescender MulanDescender	bool	Bandera que activa la función de decremento a su rotación para lograr el efecto de cambio de ángulo debido al descenso del vuelo.
Rotacion1	float	Variable que aumenta constantemente para hacer girar las llantas.

SentidoViento	bool	Bandera que indica el sentido en el que se mueve el viento.
Viento	float	Variable que aumenta y disminuye rápidamente, lo cual hace que los paracaídas se muevan de lado a lado simulando que hay un viento presente.
Puerta	bool	Bandera que indica en qué estado se encuentra la puerta, si cerradas o abiertas.
AnimPuerta	bool	Bandera que indica si la animación esta activa y mueva las puertas dependiendo de su estado.
RotPuerta	float	Indican el ángulo que se encuentran las puertas para lograr que se abran o cierren.
AnimCubo	bool	Bandera que indica si la animación del question block se activa o no
EstadoCubo	bool	Bandera que indica si se usa el cubo con textura amarilla o el cubo con textura oscura.
EscalaCubo EscalaVacia	float	Variables que escalan los cubos amarillo y oscuro para apreciar uno y ocultar el otro
RoatChampi	float	Variable que controla el ángulo de rotación del champiñón cuando está en el aire.
TiroVertical	float	Formula que calcula la posición en Y del champiñón para lograr el efecto de tiro vertical.
Golpe	float	Variable que mueve un poco hacia arriba y abajo para simular el pequeño brinco que hace el cubo cuando uno lo golpea.
tiempo	float	Variable auxiliar que mide el tiempo de vuelo, el cual se usa para calcular la altura del champiñón.
FlorBrinco	float	Calcula la posición de la flor dependiendo de su rotacion para simular los brincos que realiza la flor.
FlorGiro	float	Variable incrementable y decrementadle que determina el ángulo de rotación de la flor.
FlorSentido	bool	Bandera que indica el sentido en que gira
VueloAltura	float	Formula que calcula mediante el uso de senos la posicion de la altura del mario Tanooki con una variable auxiliar.
Tanooki	float	Variable auxiliar que aumenta constantemente para apoyar a calcular la altura de Mario Tanooki

Audio

irrKlang es un motor de sonido y una biblioteca de audio de alto nivel que ofrece numerosas ventajas para la implementación de audio en tu proyecto de Computación Gráfica. Debido a una investigación previa se decidió usar esta biblioteca. Por lo que se implementó en el código la carga de biblioteca, la declaración de variables que contendrán los audios implementados y se colocaron sus reproducciones sin loop dentro del while, la música de día o de noche se

encontrará en la lógica de cambio de estado día, en el que aparte de cambiar los valores de iluminación, reproducirá la música correspondiente al cambió de estado. Mientras que el rugido de Bowser solo será cuando se presione la tecla R y la animación de Bowser este apagada, para así encenderla y reproducir el audio para dar vida a la escena.