# THE WELL STACKED PIZZA EN OPENGL

## MANUAL TÉCNICO

DANTE MOISES ARGÜELLO LEÓN
https://github.com/dargleon/thewellstackedpizza
8 de junio del 2023

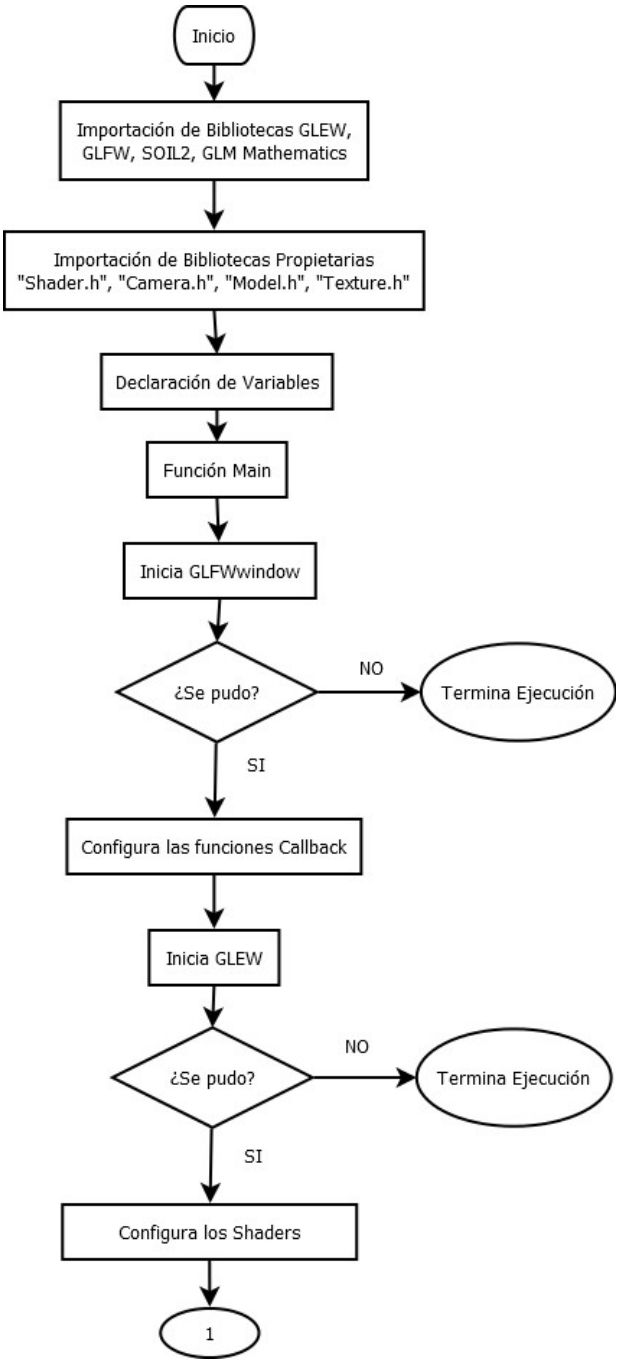
DANTE MOISÉS ARGÜELLO LEÓN

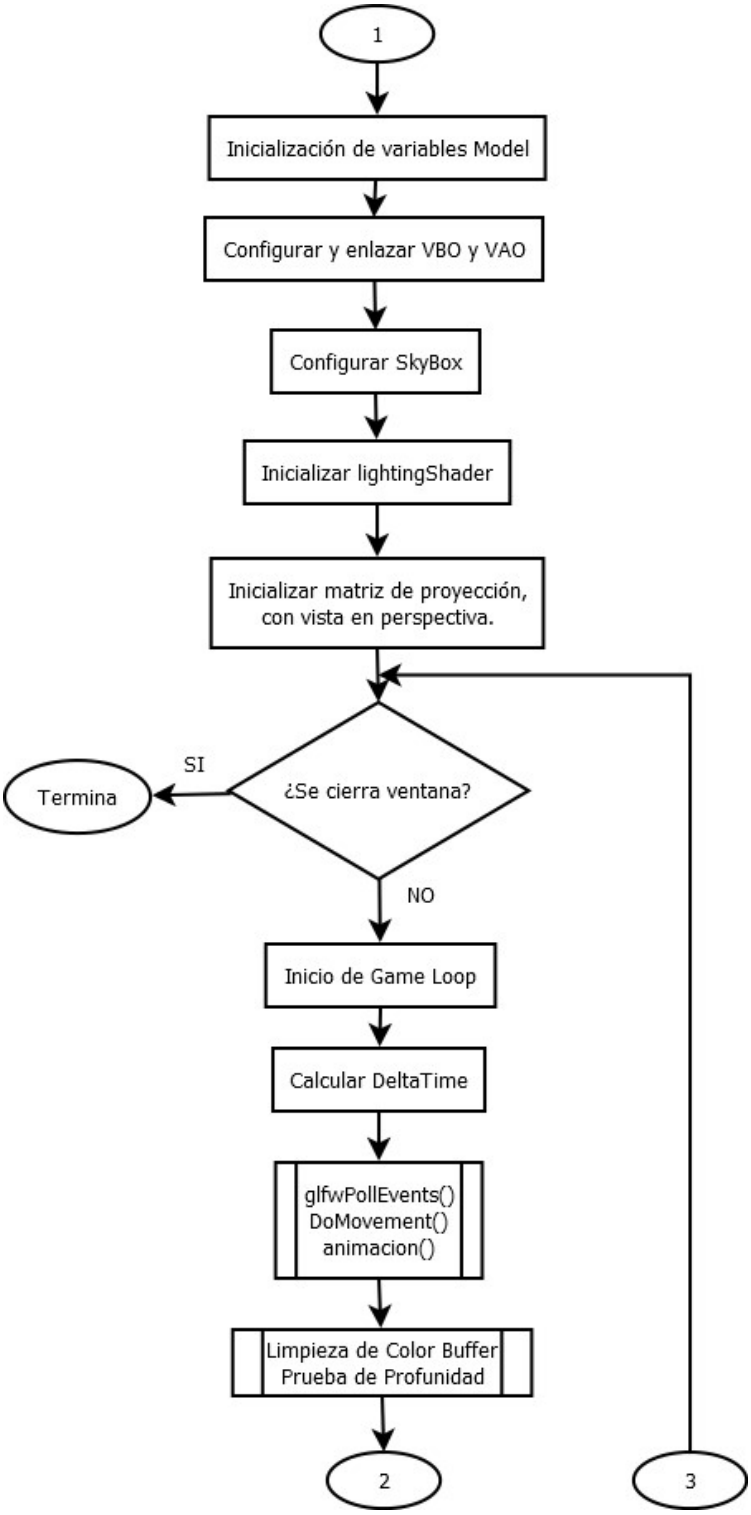
# Manual Técnico

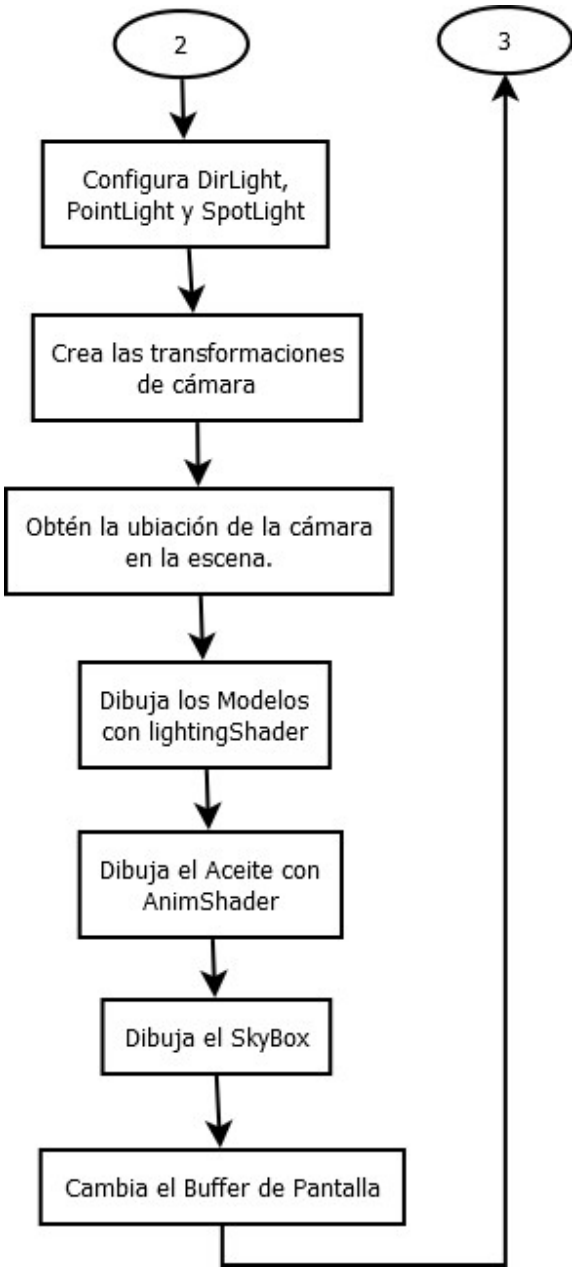
## Objetivo

- Comprender los elementos que forman parte de una implementación de un entorno virtual utilizando OpenGL and C++.

## Diagrama de Flujo del Software




1
Inicialización de variables Model
Configurar y enlazar VBO y VAO
Configurar SkyBox
Inicializar lightingShader
Inicializar matriz de proyección, con vista en perspectiva.
¿Se cierra ventana?
SI
Termina
NO
Inicio de Game Loop
Calcular DeltaTime
glfwPollEvents()
DoMovement()
animacion()
Limpieza de Color Buffer
Prueba de Profunidad
2
3


2
3
Configura DirLight, PointLight y SpotLight
Crea las transformaciones de cámara
Obtén la ubiación de la cámara en la escena.
Dibuja los Modelos con lightingShader
Dibuja el Aceite con AnimShader
Dibuja el SkyBox
Cambia el Buffer de Pantalla

## Diagrama de Gantt

| ACTIVIDAD | PERIODO | MARZO 2023 | | | | | ABRIL 2023 | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| MODELADO MESA | 16/MAR - 30/MAR | | | | | | | | | |
| MODELADO SILLA | 30/MAR - 4/ABRIL | | | | | | | | | |
| ESTUDIO DE MAYA | 4/ABRIL - 18/MAYO | | | | | | | | | |
| MODELADO DE PARRILLA | 10/ABRIL - 3/MAYO | | | | | | | | | |
| | | 6 | 12 | 18 | 24 | 31 | 7 | 14 | 21 | 30 |

| ACTIVIDAD | PERIODO | MAYO 2023 | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ESTUDIO DE MAYA | 4/ABRIL - 18/MAYO | | | | | | | | | | | | | | | | | | | | | | | | | |
| MODELADO DE PARRILLA | 10/ABRIL - 3/MAYO | | | | | | | | | | | | | | | | | | | | | | | | | |
| MODELADO SILLON | 5/MAYO - 12/MAYO | | | | | | | | | | | | | | | | | | | | | | | | | |
| CARGA DE OBJETOS EN GITHUB | 12/MAYO - 14/MAYO | | | | | | | | | | | | | | | | | | | | | | | | | |
| MODELADO DE PARRILLA LARGA | 12/MAYO - 15/MAYO | | | | | | | | | | | | | | | | | | | | | | | | | |
| DESARROLLO DE INTERIOR | 13/MAYO - 20/MAYO | | | | | | | | | | | | | | | | | | | | | | | | | |
| MODELADO DE PARRILLA CHICA | 15/MAYO - 16/MAYO | | | | | | | | | | | | | | | | | | | | | | | | | |
| MODELADO DE VITRINA DE PIZZAS | 16/MAYO - 17/MAYO | | | | | | | | | | | | | | | | | | | | | | | | | |
| USO DE CONJUNTOS DE OBJETOS | 17/MAYO- 19/MAYO | | | | | | | | | | | | | | | | | | | | | | | | | |
| INTEGRACION DE OBJETOS | 17/MAYO - 25/MAYO | | | | | | | | | | | | | | | | | | | | | | | | | |
| DESARROLLO DE FACHADA | 20/MAYO - 22/MAYO | | | | | | | | | | | | | | | | | | | | | | | | | |
| DESARROLLO DE ANIMACIONES | 22/MAYO - 24/MAYO | | | | | | | | | | | | | | | | | | | | | | | | | |
| SKYBOX | 25/MAYO - 25/MAYO | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 |

| ACTIVIDAD | PERIODO | JUNIO 2023 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| MODELADO DE MESA DE PREP. | 2/JUNIO - 3/JUNIO | | | | | | | | |
| MODELADO DE MESA CON PUERTA | 3/JUNIO - 5/JUNIO | | | | | | | | |
| MODELADO DE HORNO PIZZERO | 3/JUNIO - 5/JUNIO | | | | | | | | |
| MODELADO DE C. REGISTRADORA | 6/JUNIO - 8/JUNIO | | | | | | | | |
| MODELADO DE MICROONDAS | 6/JUNIO - 8/JUNIO | | | | | | | | |
| NUEVAS ANIMACIONES | 8/JUNIO | | | | | | | | |
| ENTREGA | 8/JUNIO | | | | | | | | |
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

## Alcance del Proyecto

Se plantea el modelado y texturizado de la Pizzería “The Well Stacked Pizza” presente en el videojuego GTA: San Andreas, siguiendo los modelos e imágenes de referencia del juego; se tratará de detallar bien el interior de la Pizzería pues es un entorno que se puede dividir en 2 cuartos, el comedor y la cocina, ambos tienen muchos elementos; así que de la cocina se plantea modelar correctamente los objetos Mesa Chica, Silla, Sillón, Mesa Larga y la Vitrina de Pizzas. Mientras que, de la cocina se plantea modelar la Parrilla Larga, la Parrilla Pequeña, la Freidora, la Mesa de Preparación y el Horno Pizzero. Así como la incorporación de al menos 5 animaciones y más modelos que complementen la ambientación de los interiores y del exterior.



https://github.com/dargleon/thewellstackedpizza

**Metodología de Software**

Se emplea una metodología Incremental, con enfoque en un desarrollo bottom-up; esto debido a que el proyecto da oportunidad de segmentar avances en distintos ámbitos, realizando commits algo pequeños, pero asegurando el avance.

En la metodología Incremental, el desarrollo de software se va construyendo el producto final de manera progresiva. En cada etapa incremental se agrega una nueva funcionalidad, lo que permite ver resultados de una forma más rápida en comparación con el modelo en cascada. El software se puede empezar a utilizar incluso antes de que se complete totalmente y, en general, es mucho más flexible que las demás metodologías.

El enfoque de desarrollo es dio bottom-up, ya que los primeros avances consisten en el desarrollo de objetos, que son elementos individuales dentro de todo el proyecto y que están en la base del modelado; posteriormente se modelaron las estructuras manteniendo la escala de los objetos, y con las estructuras realizadas, se incorporan conjuntos de objetos individuales como se da en el caso de los objetos “MesasYSillas” o “MesasYSillones” que se verán en la documentación del código.

Con los objetos individuales y los conjuntos de objetos colocados en las estructuras, es posible empezar a realizar las acciones de integración, que son parte “up” del desarrollo ya que requirieron de las demás bases. Dentro de las acciones de integración, se aprecia la separación de algunos objetos con tal de animarlos o la incorporación de iluminación de todos los modelos en una sola escena. Se aprecia la naturaleza de la metodología Incremental en cada uno de los commits de GitHub ya que todos son posibles de compilar para ver los respectivos avances.

**Tiempo de Desarrollo**

Con respecto a las horas usadas para el desarrollo, se planteó desde un principio mantener ejecutándose el videojuego GTA: San Andreas en segundo plano, pues en la plataforma de Steam se cuenta la cantidad de horas utilizadas. Fue necesario tener ejecutándose el videojuego ya que servía como referencia para hacer observaciones a detalle mientras se modelaba, y durante la programación para hacer ubicaciones espaciales. A continuación, se muestra el tiempo usado en el juego según la plataforma de Steam.


Antes de comenzar con el proyecto, se tenían registradas 48 hrs de juego, así que si hacemos la diferencia de 124 hrs – 48hrs, nos queda que el proyecto nos llevó aproximadamente 76 horas de modelado y parte de programación.

Para la parte completa de programación, se establece que se usaron 25 horas de desarrollo de animaciones, en la iluminación y lógica.

**Documentación del Código**

El código hace uso de las bibliotecas de GLEW, GLFW, GLM Mathematics y SOIL2.

El código comienza declarando los prototipos de funciones que usará, así como las variables necesarias para el manejo de la ventana y de la cámara. Se tiene un vector con la posición de cada una de las PointLigths.

Otro de sus vectores de inicio, es el llamado vértices que contiene los vértices de las 6 caras de un cubo conformado por triángulos, este cubo es necesario para cuando deseemos representar gráficamente la posición de las PointLigths.

También declaramos las variables flotantes y booleanas de animación, las cuales son:

Flotantes:

- movSilla, contendrá el desplazamiento de una silla individual en el eje X.
- movCanasta, contendrá el desplazamiento de la canasta de le freidora en el eje Y.
- movHornoPizz, contendrá el desplazamiento del interior del Horno Pizzero en eje Z.
- rotPuertaPrincipal, contendrá los radianes de rotación en el eje Y.
- rotPuertaC1, contendrá los radianes de rotación en el eje Y.

- rotPuertaC2, contendrá los radianes de rotación en el eje Y.
- rotPizzas, contendrá los radianes de rotación en el eje Y.
- rotMicroondas, contendrá los radianes de rotación en el eje Y.
- rotMesa, contendrá los radianes de rotación en el eje Y.
- tiempo, necesario para la animación por Shader del Aceite.
- speed, necesario para la animación por Shader del Aceite.

Booleanas, se encargan de activar o desactivar sus respectivas animaciones:

- sillaAnim
- puertaAnim
- puertaC1Anim
- puertaC2Anim
- pizzasAnim
- canastaAnim
- mesaAnim
- microondasAnim
- hornoPizzAnim

El DeltaTime es necesario, para homogeneizar la velocidad de los movimientos.

- **main()**

En la función principal tenemos la inicialización de GLFW, y la creación de un objeto GLFWwindow donde especificamos nuestro tamaño de ventana y el nombre de la ventana. Dado el caso que no se pueda abrir, se mostrará el mensaje "Failed to create GLFW window" y terminará la ejecución.

Si la creación del objeto ventana es exitoso, se definirá la ventana como nuestro contexto actual y se creará un buffer de frames. Para GLFW es necesario definir las funciones de interrupción, las cuales se llaman KeyCallBack y MouseCallBack, que posteriormente explicaremos. Para

tener un manejo de cámara más cómodo, en la creación de la ventana se especifica la deshabilitación del cursor con tal de que el movimiento del mouse no nos saque de la ventana creada.

Además de GLFW, debemos habilitar a GLEW para cargar extensiones de OpenGL de forma dinámica. Un detalle importante, es que en nuestro código habilitamos la propiedad experimental de GLEW con tal de poder emplear punteros y extensiones; si falló el inicio de GLEW se muestra un mensaje de notificación y termina la ejecución. Después de habilitar GLEW y GLFW, definimos las dimensiones de nuestro viewport para que coincidan con el tamaño de la ventana.

En nuestra función principal, también tenemos la declaración de nuestros Shaders que serán utilizados para la iluminación, dibujo y animación de nuestros objetos, tenemos a

- ligthingShader que hace uso de ligthing.vs y ligthing.frag
- lampShader que hace uso de lamp.vs y lamp.frag
- SkyBoxShader que hace uso de SkyBox.vs y SkyBox.frag
- Anim que hace uso de anim.vs y anim.frag

Con nuestros Shaders declarados, el siguiente paso es llamar a los objetos por medio de variables de tipo Model, se decidió la siguiente estructura para los modelos.

Para la estructura, se tienen los siguientes modelos:

- Interior. Corresponde a la estructura interior de la pizzería, con su publicidad interna y sin una pared.
- Fachada. Corresponde a la estructura exterior de la pizzería, con su banqueta y cercas y con 4 paredes, una de ellas la comparte con el interior de la pizzería. Es preciso mencionar que la estructura de Fachada “cubre” a la estructura del Interior al momento de ser dibujado.
- LogosFachada. Debido a que estos Logos presentan transparencias, se mantienen en modelos aparte para ser dibujados con su canal alfa.
- BasesLamp. Contiene las bases de las lámparas en el interior de la pizzería.
- Lamps. Contiene los vidrios de las lámparas en el interior de la pizzería, debido a que es necesaria su transparencia, se modelaron aparte.

Tenemos otro bloque correspondiente a las Puertas, que consta de:

- PuertaPrin. Es la puerta principal de la Pizzería.
- PuertaC1. Es la puerta interior que da al comedor.
- PuertaC2. Es la puerta interior que da a la cocina.

En el bloque correspondiente al comedor, tenemos:

- Silla. Es una silla individual que se modeló aparte debido a la implementación de su animación.
- MesasYSillas. Es un conjunto de objetos de sillas y mesas chicas, pues debido a la repetición de los objetos individuales de mesa y silla, se optó por manejarlos como un solo objeto.
- MesasYSillones. Es un conjunto de objetos de sillones y mesas largas, pues debido a la repetición de los objetos individuales de mesa y silla, se optó por manejarlos como un solo objeto.

En el bloque correspondiente a la cocina, tenemos:

- Freidora. Es la estructura de la Freidora.
- Aceite. Es el plano que representa el Aceite en la Freidora.
- Canasta. Es la canasta de la Freidra.
- Parrilla. Es una Parrilla Larga que va en la esquina de la cocina.
- ParrillaChica. Es una Parrilla Chica que va entre la Parrilla Larga y la Freidora.
- VitStructure. Es la estructura de metal de la vitrina de pizzas.
- VitGlass. Es el vidrio de la vitrina de pizzas, se modeló aparte debido a su dibujado con transparencia.
- VitPizzas. Son los 3 niveles de Pizzas de la Vitrina.
- MesaPreparación. Es una mesa de preparación de pizzas, se coloca pegada a la pared derecha en la cocina.

- MesaConPuertas. Es un modelo de mesa con una puerta, esta mesa sostiene al Horno Pizzero.
- PuertaMCP. Es la segunda puerta de la MesaConPuertas, se modeló aparte para ser animada.
- HornoStructure. Es la estructura exterior del Horno Pizzero.
- HornoInside. Es la plancha del interior del Horno Pizzero, se modeló aparte para ser animada.
- CajaRegistradora. Es una caja registradora que va en la misma barra que las Vitrinas de Pizzas.
- MicroondasHorno. Es un Horno de Microondas sin puerta.
- MicroondasPuerta. Es la puerta del Horno de Microondas, se modeló aparte para ser animada.

Esos son todos los objetos que debemos declarar, después de ello, tenemos el vector skyBoxVertices que contiene los vértices del SkyBox.

Para comenzar con el dibujado de objetos, primero debemos vincular el VAO y el VBO para garantizar que los datos de los vértices se almacenen en el VBO y los atributos de posición y normal se configuren en el VAO, para tener acceso a los datos de vértices durante la renderización. También configuramos un VAO y un VBO con los datos del SkyBox y se cargan sus caras con la función push_back en el vector faces, con tal de mapearlos en el cubo.

Justo antes de iniciar el ciclo principal del programa, se genera una matriz de proyección empleando una visión en perspectiva.

Al iniciar el ciclo principal, se calcula el DeltaTime del frame actual, pues es necesario calcularlo constantemente para que tenga efectos correctos. Se revisan las funciones de interrupción glfwPollEvents(), DoMovement() y animación() para revisar cambios en los periféricos y realizar las acciones respectivas.

También se limpia el colorbuffer antes de dibujar cada frame nuevo y se habilita el búfer de profundidad y la prueba de profundidad para almacenar la información de la distancia de los objetos desde la cámara correctamente.

También es necesario configurar las luces, para ello se inicia el ligthingShader proporcionándole la información de posición de la cámara, para que se dibujen correctamente las escenas. En el código se optó por configurar una DirectionalLight con componentes ambientales y difusas bajas, la especular un poco elevada. Su dirección da de frente a la fachada.

Se decidió tener 6 PointLigths, 1 de ellas ambientando levemente la cocina, otra representa al sol que da de frente a la fachada y las restantes simulan la entrada del sol por las ventanas del cuarto interior.

Se decidió tener 4 SpotLights, 2 de ellas simulan el foco amarillo que mantiene caliente a las pizzas en sus vitrinas, las otras 2 son la iluminación que otorgan las lámparas de techo en el interior de la cocina.

El shininess fue especificado con el valor 16.0f, esta propiedad afecta la forma en que se calcula el componente especular de la iluminación en los objetos, así que ese valor no otorgará mucho brillo, pero se verán bien.

Posteriormente, se calcula la matriz de vista de la cámara y se pasan las matrices de vista y proyección al ligthingShader utilizando las variables uniformes correspondientes. También se declara una matriz “model” normalizada y que será utilizada para aplicar transformaciones de modelo a los objetos. Antes de la carga de modelo, es necesario actualizar la matriz de vista con respecto a la posición y orientación de la cámara.

Utilizando el lightingShader, se dibujan todos los objetos, excepto el Aceite, pues ese se dibuja con el Shader de Anim. Es necesario especificar que para LogosFachada se activa la transparencia; y que para Lamps y VitGlass se dibuja con colorAlpha, para generar las transparencias, para finalizar el uso de lightingShader se usa glBindVertexArray.

Para finalizar con un Frame, se dibuja el SkyBox empleando el SkyBoxShader donde se emplea el mapeo cúbico de las texturas para su dibujado y se modifica la vista para lograr una profundidad inalcanzable, por ello es necesario dibujarlo al final.  Para el cambio de frame, se hace un swap de los buffers.

Dado el caso que se termine el ciclo de dibujado, esto es, cerrando la ventana, entonces se limpian todas las matrices de vértices y buffers, se termina GLFW con la función glfwTerminate() y se retorna 0. Todo lo anterior es con respecto a la función main.

**Animación y Movimientos de Cámara**

Las funciones que son llamadas por main() antes de dibujar un frame y que se encargan de las interrupciones de mouse y teclado para tener efectos en las animaciones y/o en el movimiento de la cámara son las siguientes.

- **DoMovement()**

Esta función permite realizar despalazamientos en los ejes X y Z de la cámara empleando las teclas:

- W o Up, para ir en -Z
- S o Down, para ir en +Z
- A o Left, para ir en +X
- D o Right, para ir en -Z

Hace uso de la biblioteca “camera.h” para acceder a la función ProcessKeyBoard del objeto declarado camera, con tal de realizar dichas funciones sobre la posición de la cámara y está normalizado con el deltaTime.

- **animacion()**

Esta función lee los valores de las variables booleanas que activan o desactivan una animación, dependiendo los siguientes casos.

puertaAnim. Si está activado, causa una rotación en la Puerta Principal de +0.05 radianes en el eje Y, hasta llegar a 90°; si está desactivado rota en sentido inverso hasta llegar a 0° (la posición original de la puerta).

PuertaC1Anim. Si está activado, causa una rotación en la Puerta C1 de +0.05 radianes en el eje Y, hasta llegar a 90°; si está desactivado rota en sentido inverso hasta llegar a 0° (la posición original de la puerta).

PuertaC2Anim. Si está activado, causa una rotación en la Puerta C2 de +0.05 radianes en el eje Y, hasta llegar a 90°; si está desactivado rota en sentido inverso hasta llegar a 0° (la posición original de la puerta).

sillaAnim. Si esta activo, causa un desplazamiento sobre el eje X de +0.1 hasta llegar a 0.8f, para simular que alguien se va a sentar, pero si está desactivado, el desplazamiento se da de manera inversa hasta llegar a su posición original.

pizzasAnim. Si está activo, causa una rotación de 0.3f en las Pizzas de la Vitrina sobre el eje Y, cuando no está activo, no rota.

aceiteAnim. Si está activo, fija una velocidad en 25.0f que afectará a su animación por Shader especificado en el shader Anim, cuando no está activo, la velocidad es 0.

canastaAnim. Si está activo, causa un desplazamiento sobre el eje Y de +0.025 hasta llegar a 0.5f, para simular que se levantó la canasta del aceite, pero si está desactivado, el desplazamiento se da de manera inversa hasta llegar a su posición original.

mesaAnim. Si está activo, causa una rotación de -0.5f sobre el eje Y hasta llegar a -90°, en el modelo PuertaMCP para simular que se abre una puerta de la mesa; cuando está inactivo regresa a su posición original.

microondasAnim. Si está activo, causa una rotación de -0.5f sobre el eje Y hasta llegar a -90°, en el modelo MicroondasPuerta, para simular que se abre la puerta del microondas; cuando está inactivo regresa a su posición original.

hornoPizzaAnim. Si está activo, causa un desplazamiento de +0.05f sobre el eje Z al modelo HornoInside, para simular que se saca la plancha del Horno Pizzero, si está inactivo regresa a su posición original.

- **KeyCallBack()**

Esta función lee las teclas con tal de realizar lo siguiente:

Espacio o ESC, sale de la ventana.

Tecla 1. Activa o desactiva puertaAnim.

Tecla 2. Activa o desactiva puertaC1Anim.

Tecla 3. Activa o desactiva puertaC2Anim.

Tecla 4. Activa o desactiva sillaAnim.

Tecla 5. Activa o desactiva pizzasAnim.

Tecla 6. Activa o desactiva aceiteAnim.

Tecla 7. Activa o desactiva canastaAnim.

Tecla 8. Activa o desactiva mesaAnim.

Tecla 9. Activa o desactiva microondasAnim.

Tecla P. Activa o desactiva hornoPizzAnim

Los efectos de las teclas numéricas están relacionados con las funciones de animación()

- **MouseCallBack()**

Esta función de interrupción se encarga de capturar los eventos de movimiento del mouse en la ventana, calcular el desplazamiento y enviar ese desplazamiento al objeto cámara para poder actualizar su orientación.

**Shaders Utilizados**

modelLoading

- modelLoading.vs se encarga de calcular la posición final del vértice en el espacio de recorte en función de su posición original y las matrices de transformación.
- modelLoading.frag muestra una textura en función de las coordenadas de textura del fragmento y descarta los fragmentos cuyo componente alfa sea menor a 0.1

lighting

- lighting.vs realiza las transformaciones necesarias en los vértices, calcula la posición final del vértice en el espacio de recorte, y envía información adicional como la normal, la posición en el espacio del mundo y las coordenadas
- lighting.frag calcula la iluminación para el fragmento actual, teniendo en cuenta la luz direccional, las luces puntuales y las luces focales. También aplica el color y la textura del material al fragmento y permite la activación de la transparencia según el valor de activaTransparencia.

anim

- anim.vs este shader es usado para animar el Aceite de de la freiodora, aplica una deformación ondulatoria a lo largo del eje Y a los vértices del objeto. Esta deformación se realiza en función del tiempo y de la distancia del vértice al origen. Las coordenadas de textura se mantienen sin cambios.

- anim.frag este shader de fragmentos muestrea una textura en función de las coordenadas de textura del fragmento y descarta los fragmentos con una transparencia muy baja. El resultado final es el color de la textura o una transparencia completa si el fragmento se ha descartado.

SkyBoxShader

- SkyBox.vs este shader de vértices aplica las transformaciones de proyección y vista a la posición del vértice y asigna las coordenadas del vértice a la variable de salida TexCoords. La posición transformada se asigna a gl_Position para ser utilizada en el pipeline de renderizado.
- SkyBox.frag este shader de fragmentos muestrea un cubemap de textura utilizando las coordenadas de textura del fragmento y asigna el color muestreado como el color final del fragmento.

**Otras Bibliotecas**

- Shader.h

La clase Shader encapsula la lógica para cargar, compilar, vincular y usar shaders de vértices y fragmentos en OpenGL.

- Camera.h

Esta clase de cámara encapsula la lógica para procesar la entrada del usuario y calcular las matrices y vectores necesarios para controlar la vista de la cámara en OpenGL. Puede utilizarse para implementar la funcionalidad de una cámara en una escena 3D.

- Model.h

La clase model contiene la lógica para la carga de modelos y sus respectivas texturas en OpenGL.

- Texture.h

La clase textura es utilizada para la representación del skybox, pues permite el mapeo correcto en las texturas de un cubo.

## Conclusiones

Con este proyecto, me di cuenta de que puede existir una gran complejidad en el desarrollo de entornos virtuales, y que lleva una gran cantidad de tiempo su desarrollo por lo que es muy importante el uso de control de versiones y una metodología de desarrollo, así como definir los costos de producción, de venta y utilidades para lograr una rentabilidad. También se observa que la creación de estos productos requiere de una buena capacidad de hardware para procesar las mallas de vértices en tiempos ágiles. El uso de OpenGL y bibliotecas como GLFW y GLEW facilitan la implementación de gráficos en 3D y se aprovechan en una capa de abstracción las capacidades de procesamiento gráfico de hardware. Tener preparada una documentación al mismo tiempo que se desarrolla es importante, pues facilitará la tarea de programación para cambios futuros.