Informe: Sphere Pov Ray

Computación Gráfica

Fecha de entrega: 27/10/2024

Docente: John Coppens

Integrantes del grupo:

- **Mateo Negri Ocampo 2103108**
- **Pedro Diaz Romagnoli 2223997**
- **Manuela Simes 2103975**

1. Introducción

En este informe se presenta el desarrollo de una aplicación gráfica para la visualización de esferas en 3D utilizando las bibliotecas GTK y GooCanvas. La aplicación permite crear, rotar y renderizar una esfera en tres proyecciones diferentes (XY, YZ y ZX), facilitando la comprensión de su geometría y la interacción con la misma.

El objetivo principal de este proyecto es implementar un modelo de esfera que puede ser manipulado a través de transformaciones como la rotación y la subdivisión. La aplicación se centra en ofrecer una interfaz gráfica dinámica que permite al usuario observar los efectos de las transformaciones en tiempo real.

A continuación, vamos a exponer cuáles fueron las tecnologías que utilizamos, como está compuesto el código y cómo funciona.

2. Tecnologías utilizadas

a. **Python 3**: Es el lenguaje principal con el que se escribe el programa.
b. **GTK (GIMP Toolkit)** GTK es una biblioteca de herramientas para crear interfaces gráficas de usuario (GUIs) en aplicaciones de escritorio.
c. **GooCanvas** GooCanvas es una biblioteca gráfica en 2D basada en GTK. Se usa para crear gráficos vectoriales en 2D, lo que permite dibujar y manipular figuras geométricas, como líneas, círculos, polígonos, etc., en una interfaz GTK.
d. **POV-Ray (Persistence of Vision Raytracer)** POV-Ray es un software de raytracing que genera imágenes fotorrealistas de escenas tridimensionales.

3. Código

El proyecto crea una interfaz interactiva donde el usuario puede manipular una esfera 3D, cumpliendo con los requerimientos del práctico para visualizar objetos tridimensionales en distintas proyecciones, subdividir y rotar el objeto, y controlarlo mediante una GUI.
Tenemos varios archivos para poder realizarlo.
En primer lugar analicemos el archivo **Povview.py:**

Importación y configuración de módulos:

Se importan módulos `Gtk` y `GooCanvas` de `gi.repository` para la interfaz gráfica, junto con `Vec3`, `Cone`, `Sphere` de archivos personalizados (que explicaremos y desarrollaremos más adelante), y `pyparsing` para el análisis del código POV-Ray.

Clase Views:

Esta clase representa el contenedor de vistas (representación 2D de la esfera en distintas proyecciones: `xy`, `yz`, `zx`). Está construida como un `Gtk.Grid` que alberga tres marcos

(frames) etiquetados, cada uno con un canvas (`GooCanvas.Canvas`) para visualizar las proyecciones. Se trata de una “pantalla” que nos permite ver la esfera desde diferentes “ventanas”, en cada una de las cuales se muestra la esfera desde una proyección específica: **Plano xy** (como si miraras la esfera desde arriba), **Plano yz** (de un lado) y **Plano zx** (de otro lado).

Métodos principales de `Views`:

1. **`add_object`**:
   - Agrega un objeto tridimensional (en este caso, una esfera) al contenedor y lo dibuja en cada una de las vistas. La esfera se configura con su posición y radio definidos.
2. **`clear` y `clear_all`**:
   - Estos métodos eliminan los elementos gráficos del canvas. `clear` elimina solo los gráficos sin borrar el objeto en la lista, mientras que `clear_all` borra tanto gráficos como la lista de objetos.
3. **`on_subdiv_change`, `on_size_change`, `on_rotation_change`**:
   - Estos métodos son "callbacks" . Los cuales son funciones que se ejecutan en respuesta a ciertos eventos en la interfaz de usuario. En este caso, los *callbacks* están diseñados para ejecutarse cada vez que el usuario ajusta alguno de los *sliders* (controles deslizantes) en la aplicación, que permiten modificar la apariencia de la esfera en tres aspectos:
   - **`on_subdiv_change`**: Ajusta el nivel de subdivisión (detallado) de la esfera, cambiando su apariencia en las vistas.
   - **`on_size_change`**: Modifica el tamaño de la esfera en las vistas. Al mover el *slider*, la esfera se agranda o se reduce de acuerdo con el valor seleccionado.
   - **`on_rotation_change`**: Ajusta la rotación, mostrándote la esfera desde diferentes ángulos al moverla en el espacio.

Clase `MainWindow`:

- Define la ventana principal de la aplicación, configurando el menú, las vistas y los controles de la interfaz.
- Crear el Contenedor de Vistas: Se crea una instancia de la clase `Views`.
- Contiene un menú principal (`make_main_menu`) con opciones para cargar una esfera (`on_add_sphere_clicked`) o abrir un archivo de escena POV (`on_open_pov_clicked`). Proporcionamos una mejor explicación a continuación:
  - **Agregar Esfera** → `on_add_sphere clicked`: Este método se llama cuando se selecciona la opción "Agregar esfera" del menú y añade un objeto (una esfera) a la vista.

    - **Abrir Archivo POV** → `on_open_pov_clicked`: Se abre un cuadro de diálogo para seleccionar un archivo .pov. Luego, se analiza el contenido del archivo y se añade a la vista.
    - **Salir** → `on_quit_clicked`: Este método finaliza la aplicación cuando se selecciona la opción de salir del menú.
- Incluye controles (sliders) para cambiar subdivisión, tamaño y rotación de la esfera en los ejes `x`, `y`, `z`. Estos controles están conectados a sus respectivas funciones en la clase `Views`.
- Dentro del constructor de la clase de la ventana, se crea un objeto de tipo `Gtk.Grid`. Este objeto se encargará de organizar otros widgets en filas y columnas.

Ahora, analizaremos el archivo **povview_things.py:**

Importación y configuración de módulos:

En primer lugar se hacen las importaciones necesarias, tenemos `gi` es para la interfaz gráfica, `math` para funciones matemáticas, `numpy` para operaciones con matrices, y `pdb` para depuración.

Luego, se define una constante `SUBDIV`, la cual contiene el número de subdivisiones para los objetos 3D.

Clase Vec3

Representa un vector tridimensional. Almacena las coordenadas **x**, **y** y **z** de un punto o vector en el espacio tridimensional. Es esencial en gráficos 3D, donde se necesita manipular y calcular posiciones de objetos.

Clase RGB y RGBA

Representan colores en formato RGB y RGBA (con transparencia).

- RGB permite inicializar un color con valores o como una lista.
- RGBA es similar, pero incluye un componente alfa para la transparencia.

Clase Cone

Extiende ThreeD_object y representa un cono.

- Atributos:
    - **tc, tr**: Centro y radio en la parte superior.
    - **bc, br:** Centro y radio en la parte inferior.
- Métodos:
    - `__init__`: Inicializa el cono y llama a `create_wireframe` para generar su geometría.

- `create_wireframe`: El método `create_wireframe` se encarga de calcular los puntos que forman la representación alámbrica del cono. Esto puede incluir los vértices de la base y la parte superior del cono, así como las líneas que conectan estos puntos.
- `to_svg`: Genera la representación SVG del cono para diferentes vistas (xy, yz, zx). Esto implica proyectar las coordenadas 3D del cono en un plano 2D para la visualización.
- `draw_on`: es responsable de dibujar el cono en un contexto gráfico, en el canvas proporcionado. Este método utiliza los puntos calculados en `create_wireframe` para representar visualmente el cono. Este método recibe un objeto de canvas. Utiliza los puntos almacenados en self.wireframe para dibujar líneas que conectan los vértices y representan la geometría del cono.

Clase Sphere

De forma análoga a la clase Como, esta clase representa a una **esfera**.

El método `__init__` es el constructor de la clase. Se utiliza para establecer los atributos iniciales de la esfera, como su centro y radio, y llama a `create_wireframe` para calcular su geometría. ¿Cómo funciona?

- Inicialización de Atributos: Se definen los atributos como el centro de la esfera y su radio.
- Llamada a `create_wireframe` : Se invoca el método `create_wireframe` para calcular los puntos de la superficie de la esfera.

El método `create_wireframe` es responsable de calcular los puntos que definen la superficie de la esfera. ¿Cómo funciona?

- Se inicializan las variables dtheta y dphi. Se calculan los incrementos angulares para theta (longitud) y phi (latitud) en función del número de subdivisiones (SUBDIV, variable definida en el inicio del código). Con estos valores se determina la resolución de la esfera.
- Luego, se inicializan listas vacías (`self.tx`, `self.ty`, `self.tz`, `self.bx`, `self.by`, `self.bz`) para almacenar las coordenadas de los puntos en las divisiones horizontales y verticales.
- Bucle para divisiones horizontales:
  - El primer bucle **(`for i in range(SUBDIV + 1)`)** recorre las divisiones en la dirección horizontal (longitud).
  - Para cada paso i, se calcula el ángulo de theta.
  - Se inicializan listas para almacenar coordenadas x, y, z de los puntos en la división actual.
  - Bucle interno **(`for j in range(SUBDIV + 1)`)** recorre las divisiones en la dirección vertical (latitud). En cada paso jm se calcula

el ángulo phi. Se utilizan fórmulas para calcular coordenadas cartesianas en función de theta y phi. Se aplica la transformación de rotación a las coordenadas con rotate_point que se va a explicar más adelante. Las coordenadas rotadas se añaden a las listas correspondientes (`tx`, `ty`, `tz`).

    - Al finalizar el bucle interno, se añaden las listas de coordenadas `tx`, `ty`, `tz` a las listas principales `self.tx`, `self.ty`, `self.tz`.

- Bucle para divisiones verticales:
    - Segundo Bucle **(`for j in range(SUBDIV + 1)`)** recorre las divisiones en la dirección vertical (latitud).
    - Para cada paso de j, se calcula el ángulo phi.
    - Se inicializan listas para almacenar las coordenadas `x`, `y`, `z` de los puntos en la división vertical actual.
    - Bucle Interno **(`for i in range(SUBDIV + 1)`)** recorre las divisiones en la dirección horizontal (longitud).
        - Se calcula el ángulo `theta` para cada paso `i.`
        - Se utilizan las mismas fórmulas para calcular las coordenadas cartesianas.
        - Se aplica la rotación utilizando el mismo método `rotate_point`.
        - Las coordenadas rotadas se añaden a las listas correspondientes (`bx`, `by`, `bz`).
    - Al finalizar este bucle, las listas `bx`, `by`, `bz` se añaden a las listas principales `self.bx`, `self.by`, `self.bz`.

El método `rotate_point`, aplica transformaciones de rotación a un punto en el espacio tridimensional utilizando matrices de rotación para los ejes X, Y y Z. Primero, para poder rotar un punto alrededor de un centro, necesitamos trasladarlo al origen del sistema de coordenadas. Esto significa que le restamos el vector del centro de la esfera (es decir, `self.center`) de las coordenadas del punto que queremos rotar. Sin este paso, la rotación se aplicaría en relación a un punto que no es el centro, lo que daría resultados incorrectos.

Una vez que el punto está en el origen, aplicamos las rotaciones usando matrices. Las matrices de rotación para los ejes X, Y y Z se multiplican en un orden específico: primero la matriz de rotación alrededor del eje Z, luego la del eje Y, y finalmente la del eje X. Esta multiplicación se hace sobre el vector del punto que ya trasladamos. La razón por la que seguimos este orden es que las rotaciones en 3D son dependientes del eje, y el orden en el que las aplicamos afecta el resultado final. Cada matriz transforma el punto de manera secuencial, así que el resultado de la multiplicación nos dará la nueva posición del punto en el espacio.

Finalmente, después de haber aplicado las rotaciones, necesitamos devolver el punto a su posición original. Para esto, simplemente sumamos el vector del centro de la esfera a las nuevas coordenadas del punto. Esto es crucial, ya que queremos que el punto rotado esté

en la posición correcta dentro de la esfera, no en el origen. Este paso nos asegura que el punto mantenga su relación con el centro de la esfera después de la rotación.

A continuación, tenemos el método `__str__`, el cual devuelve una representación en cadena de la esfera, mostrando su centro, radio y color.

Luego, `to_svg`, genera una representación SVG de la esfera en forma de alambre (wireframe) según la vista seleccionada (XY, YZ o ZX). Usa comandos SVG para mover y dibujar líneas que representan la forma de la esfera proyectada en el plano correspondiente.

Continuamos con `draw_on`, dibuja la esfera en las vistas proporcionadas. Para cada vista (XY, YZ, ZX), obtiene el objeto raíz del lienzo y crea un nuevo `CanvasPath` utilizando la representación SVG generada por `to_svg`.

El método `reddraw` limpia el lienzo eliminando las formas existentes y luego vuelve a dibujar la esfera. Es útil cuando se han realizado cambios en las propiedades de la esfera, como el tamaño o la rotación.

Luego, el método `update_sphere_size`, actualiza el radio de la esfera, limpia las coordenadas existentes y vuelve a crear la geometría de la esfera. Luego, redibuja la esfera en las vistas proporcionadas.

`Update_sphere_subdivision` permite cambiar la subdivisión de la esfera, lo que afecta a la calidad de la malla. Al igual que en el método anterior, se limpian los puntos existentes y se vuelve a crear la geometría antes de redibujar la esfera.

Luego, `Update_rotation` Este método actualiza el ángulo de rotación para un eje específico (X, Y o Z) y luego redibuja la esfera. Convierte el ángulo de grados a radianes antes de aplicarlo.

Clase MainWindow

La clase `MainWindow` gestiona la interfaz gráfica de usuario, creando la ventana y el lienzo donde se dibuja la esfera. Esta estructura permite una visualización interactiva y gráfica de la esfera en un entorno 3D.

Uno de sus métodos es `set_scale`, que establece la escala del lienzo. Ajusta el grosor de la línea del camino de la esfera en función de la escala para mantener una apariencia consistente.

Ahora, analizaremos el archivo **povview_parser.py:**

El objetivo principal de **povview_parser.py** es interpretar cadenas de texto de archivos POV-Ray y extraer información clave de los objetos descritos, como esferas, colores y luces.

Importación y Configuración de Pyparsing: En la primera línea de código se importa el módulo `pyparsing` que se usa para definir y ejecutar reglas de análisis sintáctico. Esta

biblioteca permite desglosar cadenas de texto en componentes interpretables, en este caso, los parámetros geométricos y de color de los objetos de la escena.

Luego se define la función **`make_pov_parser`**. La cual es el núcleo del archivo y define el parser de la escena POV-Ray. Su configuración incluye:

- **Directivas de Inclusión y Comentarios**: Se ignoran líneas que contienen `#include` y comentarios, asegurando que solo se analicen las líneas de interés para la estructura geométrica y la visualización.
- **Definición de Tipos de Datos**: Se establecen reglas para reconocer enteros y flotantes, así como vectores en distintas dimensiones. Lo cual es fundamental para interpretar las coordenadas y valores de escala en las descripciones de la escena.
- **Reconocimiento de Objetos Específicos**: Las reglas permiten identificar elementos clave de la escena, como:
    - **Vec3 y Vec4**: Representaciones de vectores en 3D y 4D.
    - **Pigmento y Color**: Define el color de los objetos mediante valores RGB, un detalle importante para representar con precisión el aspecto visual en la interfaz.
    - **Esfera (`sphere`)**: Representa una esfera con una posición en el espacio y un radio. También se permite incluir pigmento, lo cual permite darle color a la esfera en la escena.
    - **Fuente de Luz (`light_source`)**: Define una fuente de luz con una posición y un color, contribuyendo a la iluminación de la escena.

Luego tenemos las **funciones de Prueba**:

Las cuales no solo garantizan que el parser sea preciso y confiable, sino que también permiten un desarrollo más eficiente y ayudan a que el proyecto sea más mantenible y menos propenso a errores.

- **`test_basic_parser`**: Verifica el parser básico que analiza números y vectores en varias configuraciones.
- **`test_object_parser`**: Prueba el análisis de objetos completos, como esferas y fuentes de luz con pigmento y color. Los resultados se imprimen en pantalla, mostrando el éxito o el fallo en la interpretación de la cadena de texto.

Función `main`: Ejecuta la función `test_object_parser` para realizar una prueba de análisis de una descripción de escena específica de POV-Ray, permitiendo verificar si el parser interpreta correctamente los objetos definidos.

Luego, tenemos el archivo **escena_basica.pov**:

Este archivo de escena (`.pov`) contiene instrucciones específicas para representar una esfera, una fuente de luz y una cámara en un espacio tridimensional. Define atributos como:

- **Posición, radio, y color** de la **esfera**
- **Posición y color** de la **fuente de luz**
- **Ubicación y orientación** de la **cámara**

Estas instrucciones se organizan en bloques reconocibles por un parser, como el que tenemos en `povview_parser.py`, y se ejecutan posteriormente en un motor de visualización para renderizar la imagen resultante.

Por ultimo, analizaremos el archivo **main_menu.py:**

El archivo `main_menu.py` define la clase `Main_menu`, que se encarga de crear y gestionar el menú principal de la interfaz gráfica de la aplicación. Este menú es fundamental para dar al usuario una forma de interactuar con la escena en 3D, simplificando la carga y manipulación de los archivos `.pov` y optimizando el flujo de trabajo de edición y visualización.

Al inicio del archivo, se importan las bibliotecas necesarias y se define la clase `Main_menu`. El constructor de esta clase, inicializa una instancia de `Gtk.MenuBar` y una lista de elementos de menú.

Tenemos el método `add_items_to`. El cual permite añadir opciones de menú a la barra de menú. Recibe como parámetros el nombre del menú y una lista de elementos que deben añadirse. Dentro del método:

- Se crea un nuevo objeto Gtk.Menu para el menú especificado.
- Se itera sobre items, añadiendo cada elemento al menú. Si el primer elemento de la tupla es None, se añade un separador.
- Cada Gtk.MenuItem se conecta a su respectiva acción (función) utilizando connect('activate', item[1]).
- Finalmente, se añade el menú completo a la barra de menú (self.menubar) y se almacena en self.menu_items.

Método `attach`. Este método se utiliza para adjuntar el menú a un contenedor en la interfaz gráfica, permitiendo que se muestre en la ventana principal de la aplicación.

En el archivo `povview.py`, se utiliza la clase `Main_menu` definida en `main_menu.py` para crear el menú principal de la interfaz gráfica de la aplicación. Este menú permite al usuario interactuar con diferentes opciones, como abrir archivos y ejecutar pruebas.

Dentro de la clase `MainWindow`, se llama al método `make_main_menu` para crear una instancia del menú. La función `make_main_menu` inicializa `Main_menu` con las opciones de menú básicas: `_File`, `_Tests`, y `_Help`.

En el método `make_main_menu`, se añaden varias opciones de menú a la instancia `mm` mediante el método `add_items_to`. Por ejemplo, para el menú de archivos se añaden las opciones "Open POV scene..." y "Quit". Finalmente, el menú `mm` se agrega a la interfaz gráfica mediante la función `attach` del contenedor `Gtk.Grid.` Esto asegura que el menú se muestre en la parte superior de la ventana de la aplicación, permitiendo al usuario acceder a las opciones definidas.

4. Funcionamiento

Habiendo explicado y desarrollado que contiene cada archivo, podemos entender como es la relación general entre estos y el flujo de trabajo:

a. **Lectura del archivo `.pov`**: `povview_parser.py` lee el archivo (escena_basica.pov) y aplica las reglas para identificar los objetos y sus atributos (como la esfera, la luz, y la cámara).
b. **Creación de Objetos**: `POVViewThings` toma la información procesada por `povview_parser.py` y crea instancias de objetos de la escena en base a las especificaciones.
c. **Visualización en `POVView`**: Finalmente, `POVView` utiliza los objetos generados en `POVViewThings` para mostrar la escena en una interfaz gráfica. De esta manera, el usuario puede visualizar y verificar la disposición y características de los elementos 3D en la escena. A su vez, utiliza a main_menu.py de la forma que explicamos anteriormente, permitiendo al usuario interactuar con diferentes opciones, como abrir archivos y ejecutar pruebas.

5. Conclusión

En conclusión, la implementación del conjunto de archivos de la aplicación de visualización en 3D en formato POV (Persistence of Vision) demuestra un enfoque integral para la creación, manipulación y visualización de escenas gráficas. Este sistema modular nos permite una interacción dinámica con gráficos en 3D. Al realizarlo, observamos como la separación de funcionalidades en módulos permite un desarrollo más organizado y fácil de mantener. Nos demuestra cómo esta estructura modular facilita la incorporación de nuevas características sin necesidad de reescribir el código existente. Es así como al dividir el proyecto en partes más manejables, se mejora la legibilidad y se reduce la complejidad.
Por otro lado aprendimos cómo implementar sliders y menús en `main_menu.py` y como le permite a los usuarios manipular fácilmente las propiedades de los objetos en tiempo real. Resaltando la importancia de diseñar interfaces que sean tanto funcionales como accesibles.
Fue un proceso divertido y desafiante poder combinar las habilidades vistas en clase, con las que se requirieron aprender y desarrollar durante el desarrollo de este proyecto para poder lograr los objetivos planteados del trabajo.