Documentación de la Clase camera

Gerstep

August 28, 2024

1 Introducción

La clase `camera` es una implementación de una cámara en un entorno 3D utilizando OpenTK. Esta clase gestiona la posición, orientación y movimiento de la cámara, permitiendo que los usuarios se desplacen y giren la cámara en un espacio tridimensional. Es fundamental para crear experiencias interactivas en aplicaciones gráficas, simulando la vista de un observador en la escena.

Esta clase es parte del espacio de nombres `OpenTK_Tarea_3.Controlador` y proporciona una interfaz para manejar entradas del teclado y del ratón para controlar la cámara.

2 Atributos Privados de la Clase

2.1 _front

- **Tipo de Dato:** `Vector3`
- **Descripción:** Vector que indica la dirección hacia la que está mirando la cámara. Inicialmente apunta hacia el eje negativo Z, lo que significa que la cámara está mirando hacia adelante en el espacio 3D.

2.2 _up

- **Tipo de Dato:** `Vector3`
- **Descripción:** Vector que indica la dirección hacia arriba desde la perspectiva de la cámara. Este vector es fijo y apunta hacia el eje positivo Y.

2.3 _right

- **Tipo de Dato:** `Vector3`
- **Descripción:** Vector que indica la dirección hacia la derecha desde la perspectiva de la cámara. Se calcula como el producto vectorial del vector `_front` y `_up`, y se normaliza para asegurar que sea un vector unitario.

3 Propiedades de la Clase

3.1 Position (Posición)

- **Tipo de Dato:** `Vector3`
- **Descripción:** La posición actual de la cámara en el espacio 3D. Esta propiedad es clave para determinar desde dónde se está observando la escena.
- **Acceso:** Solo lectura externa. La posición de la cámara se puede modificar internamente a través de los métodos de la clase, pero no se puede establecer directamente desde fuera de la clase.

3.2 Yaw (Rotación Horizontal)

- **Tipo de Dato:** `float`
- **Descripción:** Representa la rotación de la cámara alrededor del eje Y (rotación horizontal), medida en grados. El valor por defecto es $-90.0f$, lo que apunta la cámara hacia el eje negativo Z.
- **Acceso:** Solo lectura externa.

3.3 Pitch (Rotación Vertical)

- **Tipo de Dato:** `float`
- **Descripción:** Representa la rotación de la cámara alrededor del eje X (rotación vertical), medida en grados. El valor por defecto es $0.0f$, lo que significa que la cámara no tiene inclinación hacia arriba o abajo al iniciar.
- **Acceso:** Solo lectura externa.

3.4 Speed (Velocidad de Movimiento)

- **Tipo de Dato:** `float`
- **Descripción:** Controla la velocidad de movimiento de la cámara. Este valor determina cuán rápido se mueve la cámara cuando se procesa la entrada del teclado.
- **Acceso:** Lectura y escritura. Se puede modificar desde fuera de la clase para ajustar la velocidad de la cámara.

3.5 Sensitivity (Sensibilidad del Ratón)

- **Tipo de Dato:** `float`
- **Descripción:** Controla la sensibilidad del ratón al mover la cámara. Este valor determina cuán rápido responde la cámara a los movimientos del ratón.
- **Acceso:** Lectura y escritura. Se puede modificar desde fuera de la clase para ajustar la sensibilidad de la cámara al movimiento del ratón.

3.6 bloquear_raton (Bloquear Ratón)

- **Tipo de Dato:** `bool`
- **Descripción:** Esta propiedad controla si la cámara debe responder a los movimientos del ratón. Si está establecida en `true`, la cámara no responderá a los movimientos del ratón.
- **Acceso:** Lectura y escritura. Se puede modificar desde fuera de la clase para activar o desactivar la respuesta al movimiento del ratón.

4 Constructor de la Clase

```
public camera(Vector3 position)
```

4.1 Descripción

El constructor de la clase `camera` inicializa la posición de la cámara en el espacio 3D. Establece las propiedades iniciales, como la dirección hacia adelante (`_front`), hacia arriba (`_up`), y hacia la derecha (`_right`).

4.2 Parámetro

- `position`: Un vector (`Vector3`) que define la posición inicial de la cámara en el espacio 3D.

4.3 Funcionamiento

El constructor realiza las siguientes acciones:

1. Establece la posición de la cámara utilizando el parámetro `position`.
2. Inicializa las direcciones `_front`, `_up`, y `_right` con sus valores predeterminados.

4.4 Ejemplo de Uso

```
Vector3 posicionInicial = new Vector3(0.0f, 0.0f, 3.0f);
camera camara = new camera(posicionInicial);

// Esto crea una cámara en la posición (0.0, 0.0, 3.0).
```

5 Métodos de la Clase

5.1 GetViewMatrix (Obtener la Matriz de Vista)

```
public Matrix4 GetViewMatrix()
```

5.1.1 Descripción

Este método devuelve la matriz de vista de la cámara, que es una combinación de la posición y orientación actuales de la cámara. La matriz de vista es fundamental para transformar los vértices de los objetos en el espacio 3D relativo a la cámara.

5.1.2 Funcionamiento

El método `GetViewMatrix` realiza las siguientes acciones:

1. Calcula la matriz de vista utilizando `Matrix4.LookAt`, pasando como parámetros la posición de la cámara, la posición objetivo (`Position + _front`), y el vector `_up`.
2. Devuelve la matriz de vista calculada.

5.1.3 Ejemplo de Uso

```
Matrix4 viewMatrix = camara.GetViewMatrix();

// Esto obtiene la matriz de vista actual de la cámara.
```

5.2 ProcessKeyboardInput (Procesar Entrada del Teclado)

```
public void ProcessKeyboardInput(Vector3 direction, float deltaTime)
```

5.2.1 Descripción

Este método procesa las entradas del teclado para mover la cámara en las direcciones especificadas (adelante, atrás, izquierda, derecha, arriba, abajo). Es útil para controlar la posición de la cámara en respuesta a las entradas del usuario.

5.2.2 Parámetros

- `direction`: Un vector (`Vector3`) que indica la dirección en la que la cámara debe moverse. Los valores posibles son 1, -1 y 0 para cada componente (X, Y, Z).
- `deltaTime`: Un valor flotante (`float`) que representa el tiempo transcurrido desde la última actualización, utilizado para mantener el movimiento de la cámara independiente del rendimiento.

5.2.3 Funcionamiento

El método `ProcessKeyboardInput` sigue los siguientes pasos:

1. Calcula la velocidad de movimiento multiplicando la `Speed` por el `deltaTime`.
2. Ajusta la posición de la cámara en función de la dirección especificada:
   - Movimiento hacia adelante o atrás según la dirección Z.
   - Movimiento hacia la derecha o izquierda según la dirección X.
   - Movimiento hacia arriba o abajo según la dirección Y.

5.2.4 Ejemplo de Uso

```
// Mover la cámara hacia adelante durante un tiempo delta
camara.ProcessKeyboardInput(new Vector3(0, 0, 1), deltaTime);

// Esto mueve la cámara hacia adelante en la dirección _front.
```

5.3 ProcessMouseMovement (Procesar Movimiento del Ratón)

```
public void ProcessMouseMovement(float xOffset, float yOffset, bool
constrainPitch = true)
```

5.3.1 Descripción

Este método procesa el movimiento del ratón para ajustar la rotación de la cámara. Es esencial para implementar la rotación de la cámara en respuesta a los movimientos del ratón, permitiendo al usuario mirar alrededor en un entorno 3D.

5.3.2 Parámetros

- `xOffset`: Un valor flotante (`float`) que representa el desplazamiento en el eje X del ratón.
- `yOffset`: Un valor flotante (`float`) que representa el desplazamiento en el eje Y del ratón.

- `constrainPitch`: Un valor booleano (`bool`) que indica si se debe restringir el ángulo de `Pitch` para evitar la inversión de la vista. Por defecto es `true`.

5.3.3 Funcionamiento

El método `ProcessMouseMovement` sigue los siguientes pasos:

1. Si `bloquear_raton` es `false`, se procesan los movimientos del ratón.
2. Multiplica los desplazamientos X y Y por la `Sensitivity` para ajustar la velocidad de rotación.
3. Ajusta los valores de `Yaw` y `Pitch` en función de los desplazamientos del ratón.
4. Si `constrainPitch` es `true`, restringe el `Pitch` entre $-89.0f$ y $89.0f$ grados para evitar la inversión de la vista.
5. Actualiza los vectores de la cámara (`_front`, `_right`, `_up`) llamando al método `UpdateCameraVectors`.

5.3.4 Ejemplo de Uso

```
// Rotar la cámara en respuesta al movimiento del ratón
camara.ProcessMouseMovement(mouseXOffset, mouseYOffset);

// Esto ajusta la rotación de la cámara en función del movimiento del ratón.
```

5.4 UpdateCameraVectors (Actualizar los Vectores de la Cámara)

```
private void UpdateCameraVectors()
```

5.4.1 Descripción

Este método recalcula los vectores de la cámara (`_front`, `_right`) en función de los valores actuales de `Yaw` y `Pitch`. Es crucial para asegurar que la cámara se mueva y gire correctamente en el espacio 3D.

5.4.2 Funcionamiento

El método `UpdateCameraVectors` realiza las siguientes acciones:

1. Calcula el nuevo vector `_front` utilizando las funciones trigonométricas `cos` y `sin`, basadas en los valores de `Yaw` y `Pitch`.
2. Normaliza el vector `_front` para asegurar que sea un vector unitario.
3. Recalcula el vector `_right` como el producto vectorial de `_front` y `_up`, y lo normaliza.

5.4.3 Ejemplo de Uso

```
// Este método es llamado internamente para actualizar los vectores de la cámara
camara.UpdateCameraVectors();
```

6 Conclusión

La clase `camera` es esencial para la gestión de la cámara en aplicaciones de gráficos 3D con OpenTK. Permite a los desarrolladores controlar la posición, orientación y movimiento de la cámara mediante entradas del teclado y del ratón. Al proporcionar métodos para calcular la matriz de vista y procesar las entradas del usuario, esta clase facilita la creación de experiencias interactivas y dinámicas en entornos tridimensionales.