Documentación de la Clase Shader

Gerstep

August 28, 2024

1 Introducción

La clase `Shader` es responsable de la gestión y uso de shaders en aplicaciones de gráficos 3D utilizando OpenGL. Un `Shader` es un pequeño programa que se ejecuta en la GPU y se utiliza para calcular el color de los píxeles en la pantalla, lo que permite realizar efectos visuales avanzados. Esta clase maneja la compilación de shaders de vértices y fragmentos, la creación del programa de shader, y la gestión de recursos asociados.

Esta clase es parte del espacio de nombres `OpenTK_Tarea_3.Clases_Base` y es crucial para la aplicación de técnicas de sombreado en escenas 3D.

2 Atributos Privados de la Clase

2.1 Handle

- **Tipo de Dato:** `int`
- **Descripción:** Este atributo almacena el identificador del programa de shader creado por OpenGL. Es utilizado para referenciar y activar el programa cuando se necesita en el proceso de renderizado.
- **Acceso:** Privado. El `Handle` se maneja internamente dentro de la clase para asegurar que el programa de shader sea gestionado correctamente.

3 Constructor de la Clase

```
public Shader(string vertexPath, string fragmentPath)
```

3.1 Descripción

El constructor de la clase `Shader` se utiliza para crear un programa de shader a partir de los archivos fuente de los shaders de vértices y fragmentos. Este constructor compila los shaders, los enlaza en un programa de shader, y verifica que el proceso se haya completado correctamente.

3.2 Parámetros

- `vertexPath:` Una cadena de texto (`string`) que contiene la ruta o el código fuente del shader de vértices.
- `fragmentPath:` Una cadena de texto (`string`) que contiene la ruta o el código fuente del shader de fragmentos.

3.3 Funcionamiento

El constructor realiza las siguientes acciones:

1. Compila el shader de vértices utilizando el método `CompileShader` y almacena su identificador.
2. Compila el shader de fragmentos utilizando el método `CompileShader` y almacena su identificador.
3. Crea un programa de shader con `GL.CreateProgram` y almacena su identificador en el atributo `Handle`.
4. Adjunta los shaders de vértices y fragmentos al programa de shader con `GL.AttachShader`.
5. Enlaza (link) el programa de shader utilizando `GL.LinkProgram`.
6. Verifica que el programa de shader se haya enlazado correctamente. Si no, lanza una excepción con el mensaje de error.
7. Elimina los shaders individuales, ya que su código ya ha sido vinculado al programa de shader y ya no son necesarios.

3.4 Ejemplo de Uso

```
// Crear un shader a partir de los archivos fuente
Shader shader = new Shader("vertexShader.vert", "fragmentShader.frag");

// Esto compila los shaders, los enlaza en un programa y verifica que no haya errores.
```

4 Métodos de la Clase

4.1 CompileShader (Compilar un Shader)

```
private int CompileShader(ShaderType type, string source)
```

4.1.1 Descripción

Este método compila un shader a partir del código fuente proporcionado. Dependiendo del tipo de shader (vértices o fragmentos), OpenGL compilará el código y generará un identificador para el shader compilado.

4.1.2 Parámetros

- `type`: Un enumerador (`ShaderType`) que especifica el tipo de shader que se está compilando, como `VertexShader` o `FragmentShader`.
- `source`: Una cadena de texto (`string`) que contiene el código fuente del shader.

4.1.3 Funcionamiento

El método `CompileShader` sigue los siguientes pasos:

1. Crea un nuevo shader en OpenGL con `GL.CreateShader`, especificando el tipo de shader.
2. Asigna el código fuente al shader con `GL.ShaderSource`.
3. Compila el shader con `GL.CompileShader`.
4. Verifica que la compilación haya sido exitosa. Si no, obtiene el mensaje de error y lanza una excepción, eliminando el shader creado.
5. Si la compilación es exitosa, devuelve el identificador del shader compilado.

4.1.4 Ejemplo de Uso

```
// Este método se utiliza internamente en el constructor de la clase Shader
int vertexShader = CompileShader(ShaderType.VertexShader, vertexShaderSource);

// Esto compila el shader de vértices y devuelve su identificador.
```

4.2 Use (Activar el Programa de Shader)

```
public void Use()
```

4.2.1 Descripción

Este método activa el programa de shader para que se utilice en el próximo ciclo de renderizado. Es necesario llamar a este método antes de dibujar cualquier objeto que utilice este shader.

4.2.2 Funcionamiento

El método `Use` realiza las siguientes acciones:

1. Llama a `GL.UseProgram` con el identificador almacenado en `Handle`, activando el programa de shader para su uso inmediato.

4.2.3 Ejemplo de Uso

```
// Activar el shader antes de renderizar
shader.Use();

// Ahora este shader se utilizará para renderizar los objetos en la escena.
```

4.3 SetMatrix4 (Enviar una Matriz al Shader)

```
public void SetMatrix4(string name, Matrix4 matrix)
```

4.3.1 Descripción

Este método envía una matriz 4x4 al shader, donde puede ser utilizada para transformaciones como modelado, vista y proyección. Las matrices son fundamentales en gráficos 3D para transformar vértices en el espacio 3D.

4.3.2 Parámetros

- `name`: Una cadena de texto (`string`) que contiene el nombre de la variable uniforme en el shader donde se almacenará la matriz.
- `matrix`: Un objeto de tipo `Matrix4` que contiene los valores de la matriz a enviar al shader.

4.3.3 Funcionamiento

El método `SetMatrix4` sigue los siguientes pasos:

1. Obtiene la ubicación de la variable uniforme en el shader usando `GL.GetUniformLocation`.
2. Envía la matriz al shader con `GL.UniformMatrix4`, asegurándose de que la matriz sea utilizada en las operaciones de renderizado.

4.3.4 Ejemplo de Uso

```
// Enviar la matriz de modelo al shader
shader.SetMatrix4("model", modelMatrix);

// Esto permite que el shader utilice la matriz de modelo para transformar los vértices.
```

4.4 Dispose (Liberar Recursos)

```
public void Dispose()
```

4.4.1 Descripción

Este método libera los recursos asociados con el programa de shader. Es importante llamar a `Dispose` cuando el shader ya no sea necesario para evitar pérdidas de memoria o recursos innecesarios.

4.4.2 Funcionamiento

El método `Dispose` realiza las siguientes acciones:

1. Llama a `GL.DeleteProgram` con el identificador almacenado en `Handle`, eliminando el programa de shader de la memoria de la GPU.

4.4.3 Ejemplo de Uso

```
// Liberar recursos cuando el shader ya no sea necesario
shader.Dispose();

// Esto asegura que no haya pérdidas de memoria ni recursos innecesarios en la GPU.
```

5 Conclusión

La clase `Shader` es fundamental para la gestión y uso de shaders en aplicaciones de gráficos 3D con OpenTK. Al proporcionar una interfaz para compilar, enlazar y utilizar programas de shader, esta clase permite aplicar efectos visuales avanzados y realizar transformaciones complejas en los objetos renderizados. Además, al implementar la interfaz `IDisposable`, la clase asegura que los recursos se liberen correctamente, evitando problemas de rendimiento y pérdidas de memoria.