GL_Scene

Generated by Doxygen 1.13.2

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 Camera Class Reference

Clase que gestiona la cámara en el entorno 3D, incluyendo el control de la posición, orientación y el movimiento de la cámara.

`#include <Camera.hpp>`

Public Member Functions

- Camera (glm::vec3 start_position, glm::vec3 up_direction, float start_yaw, float start_pitch)
  Constructor de la cámara.
- glm::mat4 get_view_matrix () const
  Obtiene la matriz de vista de la cámara.
- void process_keyboard (CameraMovement direction, float delta_time)
  Procesa la entrada de teclado para mover la cámara.
- void process_mouse_movement (float x_offset, float y_offset, bool constraint_pitch=true)
  Procesa el movimiento del ratón para rotar la cámara.

Public Attributes

- glm::vec3 position
  Posición actual de la cámara.
- glm::vec3 front
  Dirección hacia la cual está mirando la cámara.
- glm::vec3 up
  Vectores de orientación de la cámara en el eje Y (arriba).
- glm::vec3 right
  Vectores de la orientación de la cámara en el eje X (derecha).
- glm::vec3 world_up
  Dirección "arriba" global.
- float yaw
  Ángulo de orientación de la cámara alrededor del eje Y.
- float pitch
  Ángulo de orientación de la cámara alrededor del eje X.
- float movement_speed
  Velocidad de movimiento de la cámara.
- float mouse_sensitivity
  Sensibilidad al movimiento del ratón.
- float zoom
  Nivel de zoom de la cámara.

4.1.1 Detailed Description

Clase que gestiona la cámara en el entorno 3D, incluyendo el control de la posición, orientación y el movimiento de la cámara.

La clase `Camera` permite controlar la vista desde una cámara en 3D, proporcionando funcionalidades para mover la cámara en el espacio (adelante, atrás, izquierda, derecha, etc.), así como ajustar su orientación.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Camera()

```
Camera::Camera (
            glm::vec3 start_position,
            glm::vec3 up_direction,
            float start_yaw,
            float start_pitch)
```

Constructor de la cámara.

Inicializa una nueva cámara con la posición, dirección "arriba", yaw y pitch especificados.

Parameters

start_position	La posición inicial de la cámara en el espacio 3D.
up_direction	La dirección "arriba" de la cámara.
start_yaw	El ángulo de yaw inicial de la cámara.
start_pitch	El ángulo de pitch inicial de la cámara.

4.1.3 Member Function Documentation

4.1.3.1 get_view_matrix()

```
glm::mat4 Camera::get_view_matrix () const
```

Obtiene la matriz de vista de la cámara.

La matriz de vista se usa para transformar las coordenadas de la escena en relación con la posición y orientación de la cámara.

Returns

Una matriz 4x4 que representa la vista de la cámara.

4.1.3.2 process_keyboard()

```
void Camera::process_keyboard (
            CameraMovement direction,
            float delta_time)
```

Procesa la entrada de teclado para mover la cámara.

Cambia la posición de la cámara según la dirección especificada y el delta_time dado. El delta_time es usado para ajustar el movimiento en función del tiempo transcurrido.

Parameters

direction	La dirección en la que se desea mover la cámara (adelante, atrás, izquierda, derecha, etc.).
delta_time	El tiempo transcurrido desde el último fotograma, usado para controlar la velocidad.

4.1.3.3 process_mouse_movement()

```
void Camera::process_mouse_movement (
            float x_offset,
            float y_offset,
            bool constraint_pitch = true)
```

Procesa el movimiento del ratón para rotar la cámara.

Ajusta la orientación de la cámara en función de los movimientos del ratón. La sensibilidad de estos movimientos es controlada por el valor de `mouse_sensitivity`.

Parameters

x_offset	El cambio en la posición X del ratón.
y_offset	El cambio en la posición Y del ratón.
constraint_pitch	Si se debe restringir el ángulo de pitch para evitar una rotación excesiva.

4.1.4 Member Data Documentation

4.1.4.1 front

```
glm::vec3 Camera::front
```

Dirección hacia la cual está mirando la cámara.

Define la dirección en la que la cámara está mirando. Esto se utiliza para calcular la matriz de vista de la cámara.

4.1.4.2 mouse_sensitivity

```
float Camera::mouse_sensitivity
```

Sensibilidad al movimiento del ratón.

Controla cuánto se ajustan los ángulos de yaw y pitch cuando se mueve el ratón.

4.1.4.3 movement_speed

```
float Camera::movement_speed
```

Velocidad de movimiento de la cámara.

Define la rapidez con la que la cámara se mueve en función del delta_time.

4.1.4.4 pitch

```
float Camera::pitch
```

Ángulo de orientación de la cámara alrededor del eje X.

El ángulo de inclinación (pitch) controla la rotación de la cámara alrededor del eje horizontal.

4.1.4.5 position

```
glm::vec3 Camera::position
```

Posición actual de la cámara.

Esta es la posición de la cámara en el espacio 3D.

4.1.4.6 right

```
glm::vec3 Camera::right
```

Vectores de la orientación de la cámara en el eje X (derecha).

Define la dirección "derecha" de la cámara. Este vector es calculado en función del eje 'up' y 'front'.

4.1.4.7 up

```
glm::vec3 Camera::up
```

Vectores de orientación de la cámara en el eje Y (arriba).

Define la dirección del "arriba" de la cámara, utilizado para la orientación de la vista.

4.1.4.8 world_up

```
glm::vec3 Camera::world_up
```

Dirección "arriba" global.

Este es el vector global de "arriba" que se utiliza para la rotación de la cámara para mantener la orientación correcta de la cámara.

4.1.4.9 yaw

```
float Camera::yaw
```

Ángulo de orientación de la cámara alrededor del eje Y.

El ángulo de giro (yaw) se utiliza para girar la cámara alrededor del eje vertical.

4.1.4.10 zoom

`float Camera::zoom`

Nivel de zoom de la cámara.

Representa el zoom de la cámara, determinando el campo de visión (FOV).

The documentation for this class was generated from the following files:

- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Camera.hpp
- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Camera.cpp

4.2 Cube Class Reference

Clase que representa un cubo, heredando de la clase `Mesh`.

`#include <Cube.hpp>`

4.2.1 Detailed Description

Clase que representa un cubo, heredando de la clase `Mesh`.

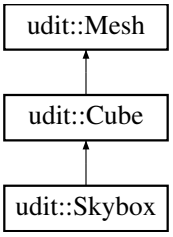
La clase `Cube` crea y gestiona un cubo 3D. Ofrece constructores para crear un cubo con un tamaño específico y con la opción de invertir las normales (útil para crear un Skybox, por ejemplo). Hereda de la clase `Mesh`..

The documentation for this class was generated from the following file:

- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Cube.hpp

4.3 udit::Cube Class Reference

Inheritance diagram for udit::Cube:



Public Member Functions

- Cube ()

  Constructor por defecto.
- Cube (bool inverted)

  Constructor con opción de invertir las normales.
- Cube (float size)

  Constructor con tamaño especificado.
- Cube (float size, bool inverted)

  Constructor con tamaño y opción de invertir las normales.

Public Member Functions inherited from udit::Mesh

- **Mesh** ()

  Constructor por defecto.
- Mesh (std::string &path)

  Constructor que carga una malla desde un archivo.
- virtual ~Mesh ()

  Destructor de la clase.
- virtual void translate (glm::vec3 translation)

  Realiza una traslación de la malla.
- virtual void rotate (glm::vec3 rotation, float angle)

  Rota la malla.
- virtual void orbit (glm::vec3 center, float distance, float speed)

  Orbita la malla alrededor de un punto.
- virtual void scale (glm::vec3 scale)

  Escala la malla.
- virtual void update ()

  Actualiza la malla.
- virtual void render (glm::mat4 view_matrix)

  Renderiza la malla.
- virtual void resize (glm::mat4 projection_matrix)

  Ajusta la matriz de proyección.
- virtual void set_shader (std::shared_ptr< udit::Shader > shader)

  Asocia un shader a la malla.
- GLuint get_shader_program_id () const

  Obtiene el ID del programa del shader asociado.
- std::vector< GLint > get_shader_matrix_ids ()

  Obtiene los IDs de las matrices del shader asociadas a la malla.
- glm::mat4 get_model_view_matrix () const

  Obtiene la matriz de transformación del modelo.
- void set_model_view_matrix (glm::mat4 matrix)

  Establece la matriz de transformación del modelo.
- void set_mesh_type (MeshType type)

  Establece el tipo de malla.

Additional Inherited Members

Static Public Member Functions inherited from udit::Mesh

- static std::shared_ptr< Mesh > make_mesh (MeshType type, const std::string &path="")

  Crea una malla de un tipo específico.

Protected Member Functions inherited from udit::Mesh

- void create_mesh (std::string mesh_name="")

  Crea los VBOs y el VAO necesarios para la malla.

Protected Attributes inherited from udit::Mesh

- std::vector< glm::vec3 > **coordinates**

  Vectores que almacenan las coordenadas de los vértices, colores, normales, índices y coordenadas de textura.
- std::vector< glm::vec3 > **colors**
- std::vector< glm::vec3 > **normals**
- std::vector< GLuint > **indices**
- std::vector< glm::vec2 > **texture_uvs**
- GLsizei **number_of_vertices**

  Número total de vértices de la malla.

4.3.1 Constructor & Destructor Documentation

4.3.1.1 Cube() [1/4]

```
Cube::Cube ()
```

Constructor por defecto.

Este constructor crea un cubo con un tamaño predeterminado y sin invertir las normales.

4.3.1.2 Cube() [2/4]

```
Cube::Cube (
            bool inverted)
```

Constructor con opción de invertir las normales.

Este constructor crea un cubo con un tamaño predeterminado. La opción de invertir las normales puede ser útil para efectos especiales como la renderización por dentro del cubo.

Parameters

inverted	Si es `true`, las normales del cubo se invierten.

4.3.1.3 Cube() [3/4]

```
Cube::Cube (
            float size)
```

Constructor con tamaño especificado.

Este constructor crea un cubo con un tamaño determinado y sin invertir las normales.

Parameters

size	El tamaño de los lados del cubo.

4.3.1.4 Cube() [4/4]

```
Cube::Cube (
            float size,
            bool inverted)
```

Constructor con tamaño y opción de invertir las normales.

Este constructor permite crear un cubo de cualquier tamaño, con la opción de invertir las normales. La inversión de las normales puede ser útil para representar el cubo desde dentro.

Parameters

size	El tamaño de los lados del cubo.
inverted	Si es `true`, las normales del cubo se invierten.

The documentation for this class was generated from the following files:

- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Cube.hpp
- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Cube.cpp

4.4 EventHandler Class Reference

Clase que maneja los eventos de entrada (teclado, ratón) en la escena.

`#include <EventHandler.hpp>`

Public Member Functions

- EventHandler (Camera &camera)

  Constructor que inicializa el EventHandler con una referencia a la cámara.
- void handle_events (bool &running, float delta_time)

  Procesa los eventos de entrada y actualiza el estado de la cámara.

4.4.1 Detailed Description

Clase que maneja los eventos de entrada (teclado, ratón) en la escena.

La clase `EventHandler` es responsable de gestionar los eventos de entrada provenientes de dispositivos como el teclado y el ratón. Se encarga de procesar dichos eventos y actualiza la cámara en consecuencia, permitiendo la navegación a través de la escena 3D.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 EventHandler()

```
EventHandler::EventHandler (
            Camera & camera)  [inline]
```

Constructor que inicializa el EventHandler con una referencia a la cámara.

Este constructor inicializa el manejador de eventos con la cámara a la que se le enviarán las actualizaciones. También establece valores predeterminados para el seguimiento del ratón.

Parameters

camera	La cámara que se actualizará en respuesta a los eventos.

4.4.3 Member Function Documentation

4.4.3.1 handle_events()

```
void EventHandler::handle_events (
            bool & running,
            float delta_time)
```

Procesa los eventos de entrada y actualiza el estado de la cámara.

Esta función maneja los eventos generados por el sistema (teclado, ratón) y, dependiendo del tipo de evento, realiza las actualizaciones necesarias en la cámara, como moverla o rotarla. Esta función debe ser llamada en cada ciclo del bucle de renderizado.

Parameters

running	Un parámetro que indica si el bucle de la aplicación sigue en ejecución. Si se establece a `false`, el bucle terminará.
delta_time	El tiempo transcurrido entre el fotograma actual y el anterior. Se utiliza para asegurar un movimiento suave de la cámara.

The documentation for this class was generated from the following files:

- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/EventHandler.hpp
- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/EventHandler.cpp

4.5 udit::Light Class Reference

Clase que representa una fuente de luz en la escena.

`#include <Light.hpp>`

Public Member Functions

- Light (const glm::vec3 &pos, const glm::vec3 &col, float ambient, float diffuse)

  Constructor de la clase Light.
- void send_to_shader (GLuint program_id) const

  Envía los parámetros de la luz al shader.

Static Public Member Functions

- static std::shared_ptr< Light > make_light (const glm::vec3 &pos, const glm::vec3 &col, float ambient, float diffuse)

  Crea una luz a partir de los parámetros especificados.

4.5.1 Detailed Description

Clase que representa una fuente de luz en la escena.

La clase `Light` es responsable de definir las características básicas de una fuente de luz, tales como su posición, color y las intensidades de la luz ambiental y difusa. Esta clase se utiliza para enviar la información de la luz a los shaders en OpenGL para que los efectos de luz sean aplicados en la escena 3D.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 Light()

```
Light::Light (
            const glm::vec3 & pos,
            const glm::vec3 & col,
            float ambient,
            float diffuse)
```

Constructor de la clase Light.

Este constructor inicializa los parámetros de la luz con valores específicos para su posición, color y las intensidades de luz ambiental y difusa.

Parameters

pos	Posición de la luz en el espacio 3D.
col	Color de la luz, especificado en formato RGB.
ambient	Intensidad de la luz ambiental.
diffuse	Intensidad de la luz difusa.

4.5.3 Member Function Documentation

4.5.3.1 make_light()

```
std::shared_ptr< Light > Light::make_light (
            const glm::vec3 & pos,
            const glm::vec3 & col,
            float ambient,
            float diffuse)  [static]
```

Crea una luz a partir de los parámetros especificados.

Esta función estática facilita la creación de un objeto `Light` compartido (`shared_ptr`) con los valores de posición, color e intensidades de luz ambiental y difusa.

Parameters

pos	Posición de la luz en el espacio 3D.
col	Color de la luz, especificado en formato RGB.
ambient	Intensidad de la luz ambiental.
diffuse	Intensidad de la luz difusa.

Returns

Un `std::shared_ptr<Light>` que apunta a la nueva luz creada.

4.5.3.2 send_to_shader()

```
void Light::send_to_shader (
            GLuint program_id) const
```

Envía los parámetros de la luz al shader.

Esta función toma los parámetros de la luz (posición, color, intensidad) y los envía al shader especificado a través de su programa de OpenGL. Esto permite que la luz sea utilizada en los cálculos de sombreado dentro del pipeline de gráficos.

Parameters

program↩ _id	El identificador del programa de shader de OpenGL.

The documentation for this class was generated from the following files:

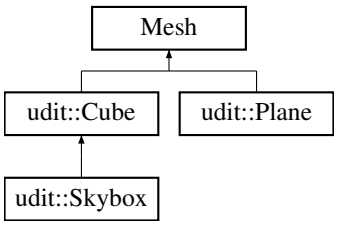
- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Light.hpp
- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Light.cpp

4.6 Mesh Class Reference

Clase que representa una malla 3D.

```
#include <Mesh.hpp>
```

Inheritance diagram for Mesh:



Public Member Functions

- **Mesh** ()

  Constructor por defecto.
- Mesh (std::string &path)

  Constructor que carga una malla desde un archivo.
- virtual ~Mesh ()

  Destructor de la clase.
- virtual void translate (glm::vec3 translation)

  Realiza una traslación de la malla.
- virtual void rotate (glm::vec3 rotation, float angle)

  Rota la malla.
- virtual void orbit (glm::vec3 center, float distance, float speed)

Orbita la malla alrededor de un punto.

- virtual void scale (glm::vec3 scale)

  Escala la malla.
- virtual void update ()

  Actualiza la malla.
- virtual void render (glm::mat4 view_matrix)

  Renderiza la malla.
- virtual void resize (glm::mat4 projection_matrix)

  Ajusta la matriz de proyección.
- virtual void set_shader (std::shared_ptr< udit::Shader > shader)

  Asocia un shader a la malla.
- GLuint get_shader_program_id () const

  Obtiene el ID del programa del shader asociado.
- std::vector< GLint > get_shader_matrix_ids ()

  Obtiene los IDs de las matrices del shader asociadas a la malla.
- glm::mat4 get_model_view_matrix () const

  Obtiene la matriz de transformación del modelo.
- void set_model_view_matrix (glm::mat4 matrix)

  Establece la matriz de transformación del modelo.
- void set_mesh_type (MeshType type)

  Establece el tipo de malla.

Static Public Member Functions

- static std::shared_ptr< Mesh > make_mesh (MeshType type, const std::string &path="")

  Crea una malla de un tipo específico.

Protected Member Functions

- void create_mesh (std::string mesh_name="")

  Crea los VBOs y el VAO necesarios para la malla.

Protected Attributes

- std::vector< glm::vec3 > **coordinates**

  Vectores que almacenan las coordenadas de los vértices, colores, normales, índices y coordenadas de textura.
- std::vector< glm::vec3 > **colors**
- std::vector< glm::vec3 > **normals**
- std::vector< GLuint > **indices**
- std::vector< glm::vec2 > **texture_uvs**
- GLsizei **number_of_vertices**

  Número total de vértices de la malla.

4.6.1 Detailed Description

Clase que representa una malla 3D.

La clase `Mesh` es la base para representar mallas 3D en OpenGL. Contiene todos los atributos y funciones necesarias para cargar, gestionar y renderizar mallas con vértices, normales, colores, coordenadas de textura y los índices que definen la topología de la malla. Esta clase también incluye funciones para transformar la malla (traslación, rotación, escala y órbita) y para actualizar y renderizar la malla en la escena.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 Mesh()

```
udit::Mesh::Mesh (
            std::string & path)
```

Constructor que carga una malla desde un archivo.

Este constructor carga los datos de la malla (coordenadas, normales, colores, etc.) desde un archivo y los almacena en los atributos correspondientes.

Parameters

path	Ruta al archivo que contiene la malla.

4.6.2.2 ∼Mesh()

```
udit::Mesh::∼Mesh ()  [virtual]
```

Destructor de la clase.

El destructor limpia los recursos de OpenGL, como los buffers y el VAO.

4.6.3 Member Function Documentation

4.6.3.1 create_mesh()

```
void udit::Mesh::create_mesh (
            std::string mesh_name = "")  [protected]
```

Crea los VBOs y el VAO necesarios para la malla.

Parameters

mesh_name	Nombre de la malla a crear.

4.6.3.2 get_model_view_matrix()

```
glm::mat4 udit::Mesh::get_model_view_matrix () const  [inline]
```

Obtiene la matriz de transformación del modelo.

Returns

La matriz de transformación del modelo.

4.6.3.3 get_shader_matrix_ids()

```
std::vector< GLint > udit::Mesh::get_shader_matrix_ids ()
```

Obtiene los IDs de las matrices del shader asociadas a la malla.

Devuelve los IDs de las matrices necesarias para renderizar la malla en el shader.

Returns

Un vector con los IDs de las matrices.

4.6.3.4 get_shader_program_id()

```
GLuint udit::Mesh::get_shader_program_id () const
```

Obtiene el ID del programa del shader asociado.

Returns

El ID del programa de shader asociado a la malla.

4.6.3.5 make_mesh()

```
std::shared_ptr< Mesh > udit::Mesh::make_mesh (
            MeshType type,
            const std::string & path = "")  [static]
```

Crea una malla de un tipo específico.

Este método estático permite crear una malla de un tipo específico, como terreno, malla básica, o malla cargada desde un archivo.

Parameters

type	Tipo de malla a crear.
path	Ruta al archivo de la malla (solo relevante si el tipo es MESH).

Returns

Un puntero compartido a la malla creada.

4.6.3.6 orbit()

```
void udit::Mesh::orbit (
            glm::vec3 center,
            float distance,
            float speed)  [virtual]
```

Orbita la malla alrededor de un punto.

Aplica una trayectoria de orbita.

Parameters

center	Punto central de orbita.
distance	Distancia al punto central.
speed	Velocidad de orbita

4.6.3.7 render()

```
void udit::Mesh::render (
            glm::mat4 view_matrix)  [virtual]
```

Renderiza la malla.

Función de renderizado de la malla en el bucle principal.

Utiliza el shader asociado y la matriz de vista para renderizar la malla.

Parameters

view_matrix	Matriz de vista.

4.6.3.8 resize()

```
void udit::Mesh::resize (
            glm::mat4 projection_matrix)  [virtual]
```

Ajusta la matriz de proyección.

Establece la matriz de proyección en el shader para la correcta visualización.

Parameters

projection_matrix	Matriz de proyección.

4.6.3.9 rotate()

```
void udit::Mesh::rotate (
            glm::vec3 rotation,
            float angle)  [virtual]
```

Rota la malla.

Aplica una rotación a la matriz de transformación de la malla.

Parameters

rotation	Eje de rotación.
angle	Ángulo de rotación en grados.

4.6.3.10 scale()

```
void udit::Mesh::scale (
            glm::vec3 scale)  [virtual]
```

Escala la malla.

Aplica una escala a la matriz de transformación de la malla.

Parameters

scale	Factor de escala.

4.6.3.11 set_mesh_type()

```
void udit::Mesh::set_mesh_type (
            MeshType type)  [inline]
```

Establece el tipo de malla.

Parameters

type	Tipo de malla.

4.6.3.12 set_model_view_matrix()

```
void udit::Mesh::set_model_view_matrix (
            glm::mat4 matrix)  [inline]
```

Establece la matriz de transformación del modelo.

Parameters

matrix	Nueva matriz de transformación del modelo.

4.6.3.13 set_shader()

```
void udit::Mesh::set_shader (
            std::shared_ptr< udit::Shader > shader)  [virtual]
```

Asocia un shader a la malla.

Permite asociar un shader para ser usado al renderizar la malla.

Parameters

shader	Puntero al shader a asociar.

4.6.3.14 translate()

```
void udit::Mesh::translate (
            glm::vec3 translation)  [virtual]
```

Realiza una traslación de la malla.

Aplica una traslación a la matriz de transformación de la malla.

Parameters

translation	Vector de traslación.

4.6.3.15 update()

```
void udit::Mesh::update ()  [virtual]
```

Actualiza la malla.

Función de actualizacion de la malla en el bucle principal.

Esta función puede ser utilizada para actualizar los datos de la malla, si es necesario.

The documentation for this class was generated from the following files:

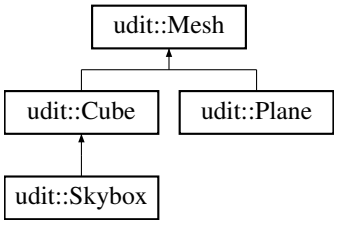
- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Mesh.hpp
- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Mesh.cpp

4.7 udit::Mesh Class Reference

Clase que representa una malla 3D.

```
#include <Mesh.hpp>
```

Inheritance diagram for udit::Mesh:



Public Member Functions

- **Mesh** ()

  Constructor por defecto.
- Mesh (std::string &path)

  Constructor que carga una malla desde un archivo.
- virtual ~Mesh ()

  Destructor de la clase.
- virtual void translate (glm::vec3 translation)

  Realiza una traslación de la malla.
- virtual void rotate (glm::vec3 rotation, float angle)

  Rota la malla.
- virtual void orbit (glm::vec3 center, float distance, float speed)

Orbita la malla alrededor de un punto.

- virtual void scale (glm::vec3 scale)

  Escala la malla.
- virtual void update ()

  Actualiza la malla.
- virtual void render (glm::mat4 view_matrix)

  Renderiza la malla.
- virtual void resize (glm::mat4 projection_matrix)

  Ajusta la matriz de proyección.
- virtual void set_shader (std::shared_ptr< udit::Shader > shader)

  Asocia un shader a la malla.
- GLuint get_shader_program_id () const

  Obtiene el ID del programa del shader asociado.
- std::vector< GLint > get_shader_matrix_ids ()

  Obtiene los IDs de las matrices del shader asociadas a la malla.
- glm::mat4 get_model_view_matrix () const

  Obtiene la matriz de transformación del modelo.
- void set_model_view_matrix (glm::mat4 matrix)

  Establece la matriz de transformación del modelo.
- void set_mesh_type (MeshType type)

  Establece el tipo de malla.

Static Public Member Functions

- static std::shared_ptr< Mesh > make_mesh (MeshType type, const std::string &path="")

  Crea una malla de un tipo específico.

Protected Member Functions

- void create_mesh (std::string mesh_name="")

  Crea los VBOs y el VAO necesarios para la malla.

Protected Attributes

- std::vector< glm::vec3 > **coordinates**

  Vectores que almacenan las coordenadas de los vértices, colores, normales, índices y coordenadas de textura.
- std::vector< glm::vec3 > **colors**
- std::vector< glm::vec3 > **normals**
- std::vector< GLuint > **indices**
- std::vector< glm::vec2 > **texture_uvs**
- GLsizei **number_of_vertices**

  Número total de vértices de la malla.

4.7.1 Detailed Description

Clase que representa una malla 3D.

La clase `Mesh` es la base para representar mallas 3D en OpenGL. Contiene todos los atributos y funciones necesarias para cargar, gestionar y renderizar mallas con vértices, normales, colores, coordenadas de textura y los índices que definen la topología de la malla. Esta clase también incluye funciones para transformar la malla (traslación, rotación, escala y órbita) y para actualizar y renderizar la malla en la escena.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 Mesh()

```
udit::Mesh::Mesh (
            std::string & path)
```

Constructor que carga una malla desde un archivo.

Este constructor carga los datos de la malla (coordenadas, normales, colores, etc.) desde un archivo y los almacena en los atributos correspondientes.

Parameters

path	Ruta al archivo que contiene la malla.

4.7.2.2 ∼Mesh()

```
udit::Mesh::∼Mesh ()  [virtual]
```

Destructor de la clase.

El destructor limpia los recursos de OpenGL, como los buffers y el VAO.

4.7.3 Member Function Documentation

4.7.3.1 create_mesh()

```
void udit::Mesh::create_mesh (
            std::string mesh_name = "")  [protected]
```

Crea los VBOs y el VAO necesarios para la malla.

Parameters

mesh_name	Nombre de la malla a crear.

4.7.3.2 get_model_view_matrix()

```
glm::mat4 udit::Mesh::get_model_view_matrix () const  [inline]
```

Obtiene la matriz de transformación del modelo.

Returns

La matriz de transformación del modelo.

4.7.3.3 get_shader_matrix_ids()

```
std::vector< GLint > udit::Mesh::get_shader_matrix_ids ()
```

Obtiene los IDs de las matrices del shader asociadas a la malla.

Devuelve los IDs de las matrices necesarias para renderizar la malla en el shader.

Returns

Un vector con los IDs de las matrices.

4.7.3.4 get_shader_program_id()

```
GLuint udit::Mesh::get_shader_program_id () const
```

Obtiene el ID del programa del shader asociado.

Returns

El ID del programa de shader asociado a la malla.

4.7.3.5 make_mesh()

```
std::shared_ptr< Mesh > udit::Mesh::make_mesh (
            MeshType type,
            const std::string & path = "")  [static]
```

Crea una malla de un tipo específico.

Este método estático permite crear una malla de un tipo específico, como terreno, malla básica, o malla cargada desde un archivo.

Parameters

type	Tipo de malla a crear.
path	Ruta al archivo de la malla (solo relevante si el tipo es MESH).

Returns

Un puntero compartido a la malla creada.

4.7.3.6 orbit()

```
void udit::Mesh::orbit (
            glm::vec3 center,
            float distance,
            float speed)  [virtual]
```

Orbita la malla alrededor de un punto.

Aplica una trayectoria de orbita.

Parameters

center	Punto central de orbita.
distance	Distancia al punto central.
speed	Velocidad de orbita

4.7.3.7 render()

```
void udit::Mesh::render (
            glm::mat4 view_matrix)  [virtual]
```

Renderiza la malla.

Función de renderizado de la malla en el bucle principal.

Utiliza el shader asociado y la matriz de vista para renderizar la malla.

Parameters

view_matrix	Matriz de vista.

4.7.3.8 resize()

```
void udit::Mesh::resize (
            glm::mat4 projection_matrix)  [virtual]
```

Ajusta la matriz de proyección.

Establece la matriz de proyección en el shader para la correcta visualización.

Parameters

projection_matrix	Matriz de proyección.

4.7.3.9 rotate()

```
void udit::Mesh::rotate (
            glm::vec3 rotation,
            float angle)  [virtual]
```

Rota la malla.

Aplica una rotación a la matriz de transformación de la malla.

Parameters

rotation	Eje de rotación.
angle	Ángulo de rotación en grados.

4.7.3.10 scale()

```
void udit::Mesh::scale (
            glm::vec3 scale)  [virtual]
```

Escala la malla.

Aplica una escala a la matriz de transformación de la malla.

Parameters

scale	Factor de escala.

4.7.3.11 set_mesh_type()

```
void udit::Mesh::set_mesh_type (
            MeshType type)  [inline]
```

Establece el tipo de malla.

Parameters

type	Tipo de malla.

4.7.3.12 set_model_view_matrix()

```
void udit::Mesh::set_model_view_matrix (
            glm::mat4 matrix)  [inline]
```

Establece la matriz de transformación del modelo.

Parameters

matrix	Nueva matriz de transformación del modelo.

4.7.3.13 set_shader()

```
void udit::Mesh::set_shader (
            std::shared_ptr< udit::Shader > shader)  [virtual]
```

Asocia un shader a la malla.

Permite asociar un shader para ser usado al renderizar la malla.

Parameters

shader	Puntero al shader a asociar.

4.7.3.14 translate()

```
void udit::Mesh::translate (
            glm::vec3 translation)  [virtual]
```

Realiza una traslación de la malla.

Aplica una traslación a la matriz de transformación de la malla.

Parameters

translation	Vector de traslación.

4.7.3.15 update()

```
void udit::Mesh::update ()  [virtual]
```

Actualiza la malla.

Función de actualizacion de la malla en el bucle principal.

Esta función puede ser utilizada para actualizar los datos de la malla, si es necesario.

The documentation for this class was generated from the following files:

- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Mesh.hpp
- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Mesh.cpp

4.8 udit::Window::OpenGL_Context_Settings Struct Reference

Public Attributes

- unsigned **version_major** = 3
- unsigned **version_minor** = 3
- bool **core_profile** = true
- unsigned **depth_buffer_size** = 24
- unsigned **stencil_buffer_size** = 0
- bool **enable_vsync** = true

The documentation for this struct was generated from the following file:

- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Window.hpp

4.9 Window::OpenGL_Context_Settings Struct Reference

Public Attributes

- unsigned **version_major** = 3
- unsigned **version_minor** = 3
- bool **core_profile** = true
- unsigned **depth_buffer_size** = 24
- unsigned **stencil_buffer_size** = 0
- bool **enable_vsync** = true

The documentation for this struct was generated from the following file:

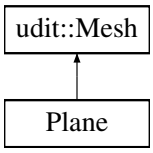
- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Window.hpp

4.10 Plane Class Reference

Clase que representa un plano 3D.

`#include <Plane.hpp>`

Inheritance diagram for Plane:



Public Member Functions

- Plane ()
  Constructor por defecto.
- Plane (float size)
  Constructor que define el tamaño del plano.
- Plane (float width, float height, unsigned columns, unsigned rows)
  Constructor que define el tamaño y la resolución del plano.

Public Member Functions inherited from udit::Mesh

- **Mesh** ()
  Constructor por defecto.
- Mesh (std::string &path)
  Constructor que carga una malla desde un archivo.
- virtual ~Mesh ()
  Destructor de la clase.
- virtual void translate (glm::vec3 translation)
  Realiza una traslación de la malla.
- virtual void rotate (glm::vec3 rotation, float angle)
  Rota la malla.
- virtual void orbit (glm::vec3 center, float distance, float speed)
  Orbita la malla alrededor de un punto.
- virtual void scale (glm::vec3 scale)
  Escala la malla.
- virtual void update ()
  Actualiza la malla.
- virtual void render (glm::mat4 view_matrix)
  Renderiza la malla.
- virtual void resize (glm::mat4 projection_matrix)
  Ajusta la matriz de proyección.
- virtual void set_shader (std::shared_ptr< udit::Shader > shader)
  Asocia un shader a la malla.
- GLuint get_shader_program_id () const
  Obtiene el ID del programa del shader asociado.
- std::vector< GLint > get_shader_matrix_ids ()

Obtiene los IDs de las matrices del shader asociadas a la malla.

- glm::mat4 get_model_view_matrix () const

  Obtiene la matriz de transformación del modelo.
- void set_model_view_matrix (glm::mat4 matrix)

  Establece la matriz de transformación del modelo.
- void set_mesh_type (MeshType type)

  Establece el tipo de malla.

Additional Inherited Members

Static Public Member Functions inherited from udit::Mesh

- static std::shared_ptr< Mesh > make_mesh (MeshType type, const std::string &path="")

  Crea una malla de un tipo específico.

Protected Member Functions inherited from udit::Mesh

- void create_mesh (std::string mesh_name="")

  Crea los VBOs y el VAO necesarios para la malla.

Protected Attributes inherited from udit::Mesh

- std::vector< glm::vec3 > **coordinates**

  Vectores que almacenan las coordenadas de los vértices, colores, normales, índices y coordenadas de textura.
- std::vector< glm::vec3 > **colors**
- std::vector< glm::vec3 > **normals**
- std::vector< GLuint > **indices**
- std::vector< glm::vec2 > **texture_uvs**
- GLsizei **number_of_vertices**

  Número total de vértices de la malla.

4.10.1 Detailed Description

Clase que representa un plano 3D.

La clase `Plane` hereda de `Mesh` y está diseñada para representar un plano 3D en OpenGL. El plano se define por su ancho, altura, y la cantidad de columnas y filas que tiene. Esta clase permite crear un plano con diferentes configuraciones, ya sea con un tamaño específico o con una distribución de vértices más compleja. El plano es útil para representar superficies planas, como terrenos o fondos.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 Plane() [1/3]

```
udit::Plane::Plane ()
```

Constructor por defecto.

Crea un plano con dimensiones predeterminadas.

4.10.2.2 Plane() [2/3]

```
udit::Plane::Plane (
            float size)
```

Constructor que define el tamaño del plano.

Crea un plano cuadrado con el tamaño especificado.

Parameters

size	Tamaño del plano en ambas dimensiones (ancho y alto).

4.10.2.3 Plane() [3/3]

```
udit::Plane::Plane (
            float width,
            float height,
            unsigned columns,
            unsigned rows)
```

Constructor que define el tamaño y la resolución del plano.

Crea un plano con el tamaño y la cantidad de columnas y filas especificados.

Parameters

width	Ancho del plano.
height	Alto del plano.
columns	Número de columnas del plano (resolución horizontal).
rows	Número de filas del plano (resolución vertical).

The documentation for this class was generated from the following files:

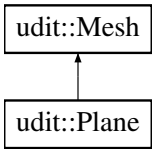
- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Plane.hpp
- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Plane.cpp

4.11 udit::Plane Class Reference

Clase que representa un plano 3D.

`#include <Plane.hpp>`

Inheritance diagram for udit::Plane:



Public Member Functions

- Plane ()

  Constructor por defecto.
- Plane (float size)

  Constructor que define el tamaño del plano.
- Plane (float width, float height, unsigned columns, unsigned rows)

  Constructor que define el tamaño y la resolución del plano.

Public Member Functions inherited from udit::Mesh

- **Mesh** ()

  Constructor por defecto.
- Mesh (std::string &path)

  Constructor que carga una malla desde un archivo.
- virtual ~Mesh ()

  Destructor de la clase.
- virtual void translate (glm::vec3 translation)

  Realiza una traslación de la malla.
- virtual void rotate (glm::vec3 rotation, float angle)

  Rota la malla.
- virtual void orbit (glm::vec3 center, float distance, float speed)

  Orbita la malla alrededor de un punto.
- virtual void scale (glm::vec3 scale)

  Escala la malla.
- virtual void update ()

  Actualiza la malla.
- virtual void render (glm::mat4 view_matrix)

  Renderiza la malla.
- virtual void resize (glm::mat4 projection_matrix)

  Ajusta la matriz de proyección.
- virtual void set_shader (std::shared_ptr< udit::Shader > shader)

  Asocia un shader a la malla.
- GLuint get_shader_program_id () const

  Obtiene el ID del programa del shader asociado.
- std::vector< GLint > get_shader_matrix_ids ()

  Obtiene los IDs de las matrices del shader asociadas a la malla.
- glm::mat4 get_model_view_matrix () const

  Obtiene la matriz de transformación del modelo.
- void set_model_view_matrix (glm::mat4 matrix)

  Establece la matriz de transformación del modelo.
- void set_mesh_type (MeshType type)

  Establece el tipo de malla.

Additional Inherited Members

Static Public Member Functions inherited from udit::Mesh

- static std::shared_ptr< Mesh > make_mesh (MeshType type, const std::string &path="")

  Crea una malla de un tipo específico.

Protected Member Functions inherited from udit::Mesh

- void create_mesh (std::string mesh_name="")

  Crea los VBOs y el VAO necesarios para la malla.

Protected Attributes inherited from udit::Mesh

- std::vector< glm::vec3 > **coordinates**

  Vectores que almacenan las coordenadas de los vértices, colores, normales, índices y coordenadas de textura.
- std::vector< glm::vec3 > **colors**
- std::vector< glm::vec3 > **normals**
- std::vector< GLuint > **indices**
- std::vector< glm::vec2 > **texture_uvs**
- GLsizei **number_of_vertices**

  Número total de vértices de la malla.

4.11.1 Detailed Description

Clase que representa un plano 3D.

La clase `Plane` hereda de `Mesh` y está diseñada para representar un plano 3D en OpenGL. El plano se define por su ancho, altura, y la cantidad de columnas y filas que tiene. Esta clase permite crear un plano con diferentes configuraciones, ya sea con un tamaño específico o con una distribución de vértices más compleja. El plano es útil para representar superficies planas, como terrenos o fondos.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 Plane() [1/3]

```
udit::Plane::Plane ()
```

Constructor por defecto.

Crea un plano con dimensiones predeterminadas.

4.11.2.2 Plane() [2/3]

```
udit::Plane::Plane (
            float size)
```

Constructor que define el tamaño del plano.

Crea un plano cuadrado con el tamaño especificado.

Parameters

size	Tamaño del plano en ambas dimensiones (ancho y alto).

4.11.2.3 Plane() [3/3]

```
udit::Plane::Plane (
            float width,
            float height,
            unsigned columns,
            unsigned rows)
```

Constructor que define el tamaño y la resolución del plano.

Crea un plano con el tamaño y la cantidad de columnas y filas especificados.

Parameters

width	Ancho del plano.
height	Alto del plano.
columns	Número de columnas del plano (resolución horizontal).
rows	Número de filas del plano (resolución vertical).

The documentation for this class was generated from the following files:

- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Plane.hpp
- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Plane.cpp

4.12 Scene Class Reference

Representa una escena 3D con un skybox, terreno, luz y otros elementos.

`#include <Scene.hpp>`

Public Member Functions

- Scene (unsigned width, unsigned height)
  Constructor de la escena.
- void update ()
  Actualiza la escena.
- void render ()
  Renderiza la escena.
- void resize (unsigned width, unsigned height)
  Redimensiona la escena.
- void set_view_matrix (const glm::mat4 &view)
  Establece la matriz de vista para la cámara.
- void set_projection_matrix (const glm::mat4 &projection)
  Establece la matriz de proyección para la cámara.
- void set_lights (GLuint shader_program_id)
  Establece las luces en el shader.

4.12.1 Detailed Description

Representa una escena 3D con un skybox, terreno, luz y otros elementos.

La clase `Scene` es responsable de gestionar la representación de una escena 3D, incluyendo los objetos gráficos principales y la iluminación. Los métodos permiten actualizar la escena, renderizarla y ajustar su tamaño.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 Scene()

```
udit::Scene::Scene (
            unsigned width,
            unsigned height)
```

Constructor de la escena.

Constructor.

Inicializa una nueva escena con el ancho y alto especificados.

Parameters

width	Ancho de la ventana de renderizado.
height	Alto de la ventana de renderizado.

Inicializa una escena

Parameters

width	Ancho de la escena
height	Alto de la escena

4.12.3 Member Function Documentation

4.12.3.1 render()

```
void udit::Scene::render ()
```

Renderiza la escena.

Renderiza los elementos de la escena.

Dibuja todos los elementos de la escena (skybox, terreno, objetos, luz) en la ventana de renderizado. Este método debe ser llamado en cada ciclo de renderizado.

4.12.3.2 resize()

```
void udit::Scene::resize (
            unsigned width,
            unsigned height)
```

Redimensiona la escena.

Ajusta la escena al nuevo tamaño de la ventana.

Parameters

width	Nuevo ancho de la ventana.
height	Nuevo alto de la ventana.

4.12.3.3 set_lights()

```
void udit::Scene::set_lights (
            GLuint shader_program_id)
```

Establece las luces en el shader.

Configura las luces de la escena dentro del shader, enviando los parámetros necesarios al programa de sombreado.

Parameters

shader_program_id	Identificador del programa de sombreado (shader).

4.12.3.4 set_projection_matrix()

```
void udit::Scene::set_projection_matrix (
            const glm::mat4 & projection)
```

Establece la matriz de proyección para la cámara.

Establece la matriz de proyección que será usada para renderizar la escena.

Parameters

projection	Matriz de proyección.

4.12.3.5 set_view_matrix()

```
void udit::Scene::set_view_matrix (
            const glm::mat4 & view)
```

Establece la matriz de vista para la cámara.

Establece la matriz de vista que será usada para renderizar la escena.

Parameters

view	Matriz de vista.

4.12.3.6 update()

```
void udit::Scene::update ()
```

Actualiza la escena.

Actualiza ciertos valores dentro del bucle principal.

Llama a las funciones necesarias para actualizar los objetos en la escena. Este método debe ser llamado cada vez que se desea actualizar el estado de la escena.

The documentation for this class was generated from the following files:

- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Scene.hpp
- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Scene.cpp

4.13 udit::Scene Class Reference

Representa una escena 3D con un skybox, terreno, luz y otros elementos.

`#include <Scene.hpp>`

Public Member Functions

- Scene (unsigned width, unsigned height)
  Constructor de la escena.
- void update ()
  Actualiza la escena.
- void render ()
  Renderiza la escena.
- void resize (unsigned width, unsigned height)
  Redimensiona la escena.
- void set_view_matrix (const glm::mat4 &view)
  Establece la matriz de vista para la cámara.
- void set_projection_matrix (const glm::mat4 &projection)
  Establece la matriz de proyección para la cámara.
- void set_lights (GLuint shader_program_id)
  Establece las luces en el shader.

4.13.1 Detailed Description

Representa una escena 3D con un skybox, terreno, luz y otros elementos.

La clase `Scene` es responsable de gestionar la representación de una escena 3D, incluyendo los objetos gráficos principales y la iluminación. Los métodos permiten actualizar la escena, renderizarla y ajustar su tamaño.

4.13.2 Constructor & Destructor Documentation

4.13.2.1 Scene()

```
udit::Scene::Scene (
            unsigned width,
            unsigned height)
```

Constructor de la escena.

Constructor.

Inicializa una nueva escena con el ancho y alto especificados.

Parameters

width	Ancho de la ventana de renderizado.
height	Alto de la ventana de renderizado.

Inicializa una escena

Parameters

width	Ancho de la escena
height	Alto de la escena

4.13.3 Member Function Documentation

4.13.3.1 render()

```
void udit::Scene::render ()
```

Renderiza la escena.

Renderiza los elementos de la escena.

Dibuja todos los elementos de la escena (skybox, terreno, objetos, luz) en la ventana de renderizado. Este método debe ser llamado en cada ciclo de renderizado.

4.13.3.2 resize()

```
void udit::Scene::resize (
            unsigned width,
            unsigned height)
```

Redimensiona la escena.

Ajusta la escena al nuevo tamaño de la ventana.

Parameters

width	Nuevo ancho de la ventana.
height	Nuevo alto de la ventana.

4.13.3.3 set_lights()

```
void udit::Scene::set_lights (
            GLuint shader_program_id)
```

Establece las luces en el shader.

Configura las luces de la escena dentro del shader, enviando los parámetros necesarios al programa de sombreado.

Parameters

shader_program↩ _id	Identificador del programa de sombreado (shader).

4.13.3.4 set_projection_matrix()

```
void udit::Scene::set_projection_matrix (
            const glm::mat4 & projection)
```

Establece la matriz de proyección para la cámara.

Establece la matriz de proyección que será usada para renderizar la escena.

Parameters

projection	Matriz de proyección.

4.13.3.5 set_view_matrix()

```
void udit::Scene::set_view_matrix (
            const glm::mat4 & view)
```

Establece la matriz de vista para la cámara.

Establece la matriz de vista que será usada para renderizar la escena.

Parameters

view	Matriz de vista.

4.13.3.6 update()

```
void udit::Scene::update ()
```

Actualiza la escena.

Actualiza ciertos valores dentro del bucle principal.

Llama a las funciones necesarias para actualizar los objetos en la escena. Este método debe ser llamado cada vez que se desea actualizar el estado de la escena.

The documentation for this class was generated from the following files:

- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Scene.hpp
- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Scene.cpp

4.14 Shader Class Reference

Representa un shader program en OpenGL.

```
#include <Shader.hpp>
```

Public Member Functions

- Shader ()

  Constructor por defecto.
- Shader (ShaderType type, const std::string &vertex_source, const std::string &fragment_source, const std::string &name)

  Constructor para crear un shader con tipos y fuentes especificadas.
- ~Shader ()

  Destructor.
- GLuint compile_shaders (const char *vertex_shader_code, const char *fragment_shader_code)

  Compila los shaders.
- GLint get_model_view_matrix_id ()

  Obtiene el identificador de la matriz de modelo-vista.
- GLint get_projection_matrix_id ()

  Obtiene el identificador de la matriz de proyección.
- GLint get_normal_matrix_id ()

  Obtiene el identificador de la matriz de normales.
- GLuint get_program_id () const

  Obtiene el identificador del programa de shader.
- void set_texture (const std::shared_ptr< Texture > &texture)

  Establece una textura para el shader.
- void use () const

  Activa y usa el programa de shader.
- void set_texture_scale (float scale)

  Establece la escala de las texturas asociadas al shader.
- bool has_textures ()

  Verifica si el shader tiene texturas asociadas.
- void set_name (const std::string &name)

  Establece el nombre del shader.
- std::string get_name ()

  Obtiene el nombre del shader.

Static Public Member Functions

- static std::shared_ptr< Shader > make_shader (udit::ShaderType type=udit::ShaderType::DEFAULT, const std::string &vertex_shader="", const std::string &fragment_shader="", const std::vector< std::string > &texture_paths={""}, const std::string &name="")

  Crea un shader.

4.14.1 Detailed Description

Representa un shader program en OpenGL.

La clase `Shader` gestiona la creación y uso de programas de sombreado en OpenGL. Permite compilar los shaders, vincularlos en un programa y usarlos para renderizar objetos en la escena. También proporciona funciones para gestionar texturas y matrices de transformación, como la matriz de modelo-vista, proyección y normales.

4.14.2 Constructor & Destructor Documentation

4.14.2.1 Shader() [1/2]

```
udit::Shader::Shader ()
```

Constructor por defecto.

Crea un objeto `Shader` sin especificar un tipo o fuentes de shader. Este constructor generalmente se usa para crear shaders más tarde con la función `make_shader`.

4.14.2.2 Shader() [2/2]

```
udit::Shader::Shader (
            ShaderType type,
            const std::string & vertex_source,
            const std::string & fragment_source,
            const std::string & name)
```

Constructor para crear un shader con tipos y fuentes especificadas.

Parameters

type	Tipo de shader (e.g., SKYBOX, GEOMETRY).
vertex_source	Código fuente para el vertex shader.
fragment_source	Código fuente para el fragment shader.
name	Nombre del shader.

4.14.2.3 ~Shader()

```
udit::Shader::~Shader ()
```

Destructor.

Libera los recursos asociados al shader.

4.14.3 Member Function Documentation

4.14.3.1 compile_shaders()

```
GLuint udit::Shader::compile_shaders (
            const char * vertex_shader_code,
            const char * fragment_shader_code)
```

Compila los shaders.

Compilador de los shaders construidos.

Compila un vertex shader y un fragment shader usando el código fuente proporcionado.

Parameters

vertex_shader_code	Código fuente del vertex shader.
fragment_shader_code	Código fuente del fragment shader.

Returns

Identificador del programa de shader compilado.

4.14.3.2 get_model_view_matrix_id()

```
GLint udit::Shader::get_model_view_matrix_id ()  [inline]
```

Obtiene el identificador de la matriz de modelo-vista.

Returns

Identificador de la matriz de modelo-vista.

4.14.3.3 get_name()

```
std::string udit::Shader::get_name ()  [inline]
```

Obtiene el nombre del shader.

Returns

Nombre del shader.

4.14.3.4 get_normal_matrix_id()

```
GLint udit::Shader::get_normal_matrix_id ()  [inline]
```

Obtiene el identificador de la matriz de normales.

Returns

Identificador de la matriz de normales.

4.14.3.5 get_program_id()

```
GLuint udit::Shader::get_program_id () const  [inline]
```

Obtiene el identificador del programa de shader.

Returns

Identificador del programa de shader.

4.14.3.6 get_projection_matrix_id()

```
GLint udit::Shader::get_projection_matrix_id ()  [inline]
```

Obtiene el identificador de la matriz de proyección.

Returns

Identificador de la matriz de proyección.

4.14.3.7 has_textures()

```
bool udit::Shader::has_textures ()  [inline]
```

Verifica si el shader tiene texturas asociadas.

Returns

`true` si el shader tiene texturas asociadas, `false` en caso contrario.

4.14.3.8 make_shader()

```
std::shared_ptr< Shader > udit::Shader::make_shader (
            udit::ShaderType type = udit::ShaderType::DEFAULT,
            const std::string & vertex_shader = "",
            const std::string & fragment_shader = "",
            const std::vector< std::string > & texture_paths = {""},
            const std::string & name = "")  [static]
```

Crea un shader.

Función estática para crear un shader con un tipo específico y fuentes de shader opcionales.

Parameters

type	Tipo de shader.
vertex_shader	Código fuente del vertex shader.
fragment_shader	Código fuente del fragment shader.
texture_paths	Rutas a las texturas asociadas.
name	Nombre del shader.

Returns

Objeto `Shader` creado.

4.14.3.9 set_name()

```
void udit::Shader::set_name (
            const std::string & name)  [inline]
```

Establece el nombre del shader.

Parameters

name	Nombre del shader.

4.14.3.10 set_texture()

```
void udit::Shader::set_texture (
            const std::shared_ptr< Texture > & texture)
```

Establece una textura para el shader.

Parameters

texture	Puntero a la textura que será asignada al shader.

4.14.3.11 set_texture_scale()

```
void udit::Shader::set_texture_scale (
            float scale)
```

Establece la escala de las texturas asociadas al shader.

Parameters

scale	Factor de escala para las texturas.

4.14.3.12 use()

```
void udit::Shader::use () const
```

Activa y usa el programa de shader.

Hace que el programa de shader sea el activo para su uso en la siguiente operación de renderizado.

The documentation for this class was generated from the following files:

- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Shader.hpp
- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Shader.cpp

4.15 udit::Shader Class Reference

Representa un shader program en OpenGL.

```
#include <Shader.hpp>
```

Public Member Functions

- Shader ()

  Constructor por defecto.
- Shader (ShaderType type, const std::string &vertex_source, const std::string &fragment_source, const std::string &name)

  Constructor para crear un shader con tipos y fuentes especificadas.
- ~Shader ()

  Destructor.
- GLuint compile_shaders (const char *vertex_shader_code, const char *fragment_shader_code)

  Compila los shaders.
- GLint get_model_view_matrix_id ()

  Obtiene el identificador de la matriz de modelo-vista.
- GLint get_projection_matrix_id ()

  Obtiene el identificador de la matriz de proyección.
- GLint get_normal_matrix_id ()

  Obtiene el identificador de la matriz de normales.
- GLuint get_program_id () const

  Obtiene el identificador del programa de shader.
- void set_texture (const std::shared_ptr< Texture > &texture)

  Establece una textura para el shader.
- void use () const

  Activa y usa el programa de shader.
- void set_texture_scale (float scale)

  Establece la escala de las texturas asociadas al shader.
- bool has_textures ()

  Verifica si el shader tiene texturas asociadas.
- void set_name (const std::string &name)

  Establece el nombre del shader.
- std::string get_name ()

  Obtiene el nombre del shader.

Static Public Member Functions

- static std::shared_ptr< Shader > make_shader (udit::ShaderType type=udit::ShaderType::DEFAULT, const std::string &vertex_shader="", const std::string &fragment_shader="", const std::vector< std::string > &texture_paths={""}, const std::string &name="")

  Crea un shader.

4.15.1 Detailed Description

Representa un shader program en OpenGL.

La clase `Shader` gestiona la creación y uso de programas de sombreado en OpenGL. Permite compilar los shaders, vincularlos en un programa y usarlos para renderizar objetos en la escena. También proporciona funciones para gestionar texturas y matrices de transformación, como la matriz de modelo-vista, proyección y normales.

4.15.2 Constructor & Destructor Documentation

4.15.2.1 Shader() [1/2]

```
udit::Shader::Shader ()
```

Constructor por defecto.

Crea un objeto `Shader` sin especificar un tipo o fuentes de shader. Este constructor generalmente se usa para crear shaders más tarde con la función `make_shader`.

4.15.2.2 Shader() [2/2]

```
udit::Shader::Shader (
            ShaderType type,
            const std::string & vertex_source,
            const std::string & fragment_source,
            const std::string & name)
```

Constructor para crear un shader con tipos y fuentes especificadas.

Parameters

type	Tipo de shader (e.g., SKYBOX, GEOMETRY).
vertex_source	Código fuente para el vertex shader.
fragment_source	Código fuente para el fragment shader.
name	Nombre del shader.

4.15.2.3 ~Shader()

```
udit::Shader::~Shader ()
```

Destructor.

Libera los recursos asociados al shader.

4.15.3 Member Function Documentation

4.15.3.1 compile_shaders()

```
GLuint udit::Shader::compile_shaders (
            const char * vertex_shader_code,
            const char * fragment_shader_code)
```

Compila los shaders.

Compilador de los shaders construidos.

Compila un vertex shader y un fragment shader usando el código fuente proporcionado.

Parameters

vertex_shader_code	Código fuente del vertex shader.
fragment_shader_code	Código fuente del fragment shader.

Returns

Identificador del programa de shader compilado.

4.15.3.2 get_model_view_matrix_id()

```
GLint udit::Shader::get_model_view_matrix_id ()  [inline]
```

Obtiene el identificador de la matriz de modelo-vista.

Returns

Identificador de la matriz de modelo-vista.

4.15.3.3 get_name()

```
std::string udit::Shader::get_name ()  [inline]
```

Obtiene el nombre del shader.

Returns

Nombre del shader.

4.15.3.4 get_normal_matrix_id()

```
GLint udit::Shader::get_normal_matrix_id ()  [inline]
```

Obtiene el identificador de la matriz de normales.

Returns

Identificador de la matriz de normales.

4.15.3.5 get_program_id()

```
GLuint udit::Shader::get_program_id () const  [inline]
```

Obtiene el identificador del programa de shader.

Returns

Identificador del programa de shader.

4.15.3.6 get_projection_matrix_id()

```
GLint udit::Shader::get_projection_matrix_id ()  [inline]
```

Obtiene el identificador de la matriz de proyección.

Returns

Identificador de la matriz de proyección.

4.15.3.7 has_textures()

```
bool udit::Shader::has_textures ()  [inline]
```

Verifica si el shader tiene texturas asociadas.

Returns

`true` si el shader tiene texturas asociadas, `false` en caso contrario.

4.15.3.8 make_shader()

```
std::shared_ptr< Shader > udit::Shader::make_shader (
            udit::ShaderType type = udit::ShaderType::DEFAULT,
            const std::string & vertex_shader = "",
            const std::string & fragment_shader = "",
            const std::vector< std::string > & texture_paths = {""},
            const std::string & name = "")  [static]
```

Crea un shader.

Función estática para crear un shader con un tipo específico y fuentes de shader opcionales.

Parameters

type	Tipo de shader.
vertex_shader	Código fuente del vertex shader.
fragment_shader	Código fuente del fragment shader.
texture_paths	Rutas a las texturas asociadas.
name	Nombre del shader.

Returns

Objeto `Shader` creado.

4.15.3.9 set_name()

```
void udit::Shader::set_name (
            const std::string & name)  [inline]
```

Establece el nombre del shader.

Parameters

name	Nombre del shader.

4.15.3.10 set_texture()

```
void udit::Shader::set_texture (
            const std::shared_ptr< Texture > & texture)
```

Establece una textura para el shader.

Parameters

texture	Puntero a la textura que será asignada al shader.

4.15.3.11 set_texture_scale()

```
void udit::Shader::set_texture_scale (
            float scale)
```

Establece la escala de las texturas asociadas al shader.

Parameters

scale	Factor de escala para las texturas.

4.15.3.12 use()

```
void udit::Shader::use () const
```

Activa y usa el programa de shader.

Hace que el programa de shader sea el activo para su uso en la siguiente operación de renderizado.

The documentation for this class was generated from the following files:

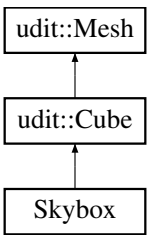
- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Shader.hpp
- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Shader.cpp

4.16 Skybox Class Reference

Representa un skybox, un cubo con texturas aplicadas en sus seis caras.

`#include <Skybox.hpp>`

Inheritance diagram for Skybox:



Public Member Functions

- Skybox ()

  Constructor por defecto.
- Skybox (float size, const std::vector< std::string > &faces)

  Constructor que permite especificar el tamaño y las texturas del skybox.
- unsigned int getCubemapTexture () const

  Obtiene el identificador de la textura cubemap cargada para el skybox.

Public Member Functions inherited from udit::Cube

- Cube ()

  Constructor por defecto.
- Cube (bool inverted)

  Constructor con opción de invertir las normales.
- Cube (float size)

  Constructor con tamaño especificado.
- Cube (float size, bool inverted)

  Constructor con tamaño y opción de invertir las normales.

Public Member Functions inherited from udit::Mesh

- **Mesh** ()

  Constructor por defecto.
- Mesh (std::string &path)

  Constructor que carga una malla desde un archivo.
- virtual ~Mesh ()

  Destructor de la clase.
- virtual void translate (glm::vec3 translation)

  Realiza una traslación de la malla.
- virtual void rotate (glm::vec3 rotation, float angle)

  Rota la malla.
- virtual void orbit (glm::vec3 center, float distance, float speed)

  Orbita la malla alrededor de un punto.
- virtual void scale (glm::vec3 scale)

  Escala la malla.
- virtual void update ()

  Actualiza la malla.
- virtual void render (glm::mat4 view_matrix)

  Renderiza la malla.
- virtual void resize (glm::mat4 projection_matrix)

  Ajusta la matriz de proyección.
- virtual void set_shader (std::shared_ptr< udit::Shader > shader)

  Asocia un shader a la malla.
- GLuint get_shader_program_id () const

  Obtiene el ID del programa del shader asociado.
- std::vector< GLint > get_shader_matrix_ids ()

  Obtiene los IDs de las matrices del shader asociadas a la malla.
- glm::mat4 get_model_view_matrix () const

  Obtiene la matriz de transformación del modelo.
- void set_model_view_matrix (glm::mat4 matrix)

  Establece la matriz de transformación del modelo.
- void set_mesh_type (MeshType type)

  Establece el tipo de malla.

Additional Inherited Members

Static Public Member Functions inherited from udit::Mesh

- static std::shared_ptr< Mesh > make_mesh (MeshType type, const std::string &path="")

  Crea una malla de un tipo específico.

Protected Member Functions inherited from udit::Mesh

- void create_mesh (std::string mesh_name="")

  Crea los VBOs y el VAO necesarios para la malla.

Protected Attributes inherited from udit::Mesh

- std::vector< glm::vec3 > **coordinates**

  Vectores que almacenan las coordenadas de los vértices, colores, normales, índices y coordenadas de textura.
- std::vector< glm::vec3 > **colors**
- std::vector< glm::vec3 > **normals**
- std::vector< GLuint > **indices**
- std::vector< glm::vec2 > **texture_uvs**
- GLsizei **number_of_vertices**

  Número total de vértices de la malla.

4.16.1 Detailed Description

Representa un skybox, un cubo con texturas aplicadas en sus seis caras.

Un skybox es un cubo que rodea la escena y sirve como fondo inmersivo en un entorno 3D. La clase `Skybox` hereda de la clase `Cube`, y se encarga de cargar las texturas y mostrar el cielo en una escena utilizando un cubo con caras texturizadas.

4.16.2 Constructor & Destructor Documentation

4.16.2.1 Skybox() [1/2]

```
udit::Skybox::Skybox ()
```

Constructor por defecto.

Este constructor crea un skybox con un tamaño por defecto y sin texturas cargadas.

4.16.2.2 Skybox() [2/2]

```
udit::Skybox::Skybox (
            float size,
            const std::vector< std::string > & faces)
```

Constructor que permite especificar el tamaño y las texturas del skybox.

Parameters

size	Tamaño del cubo que representará el skybox.
faces	Vector de rutas a las texturas que serán aplicadas a las caras del skybox.

4.16.3 Member Function Documentation

4.16.3.1 getCubemapTexture()

```
unsigned int udit::Skybox::getCubemapTexture () const  [inline]
```

Obtiene el identificador de la textura cubemap cargada para el skybox.

Returns

Identificador de la textura cubemap.

The documentation for this class was generated from the following files:

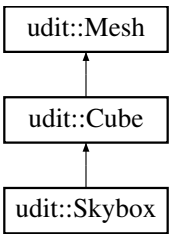
- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Skybox.hpp
- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Skybox.cpp

4.17 udit::Skybox Class Reference

Representa un skybox, un cubo con texturas aplicadas en sus seis caras.

```
#include <Skybox.hpp>
```

Inheritance diagram for udit::Skybox:



Public Member Functions

- Skybox ()

  Constructor por defecto.
- Skybox (float size, const std::vector< std::string > &faces)

  Constructor que permite especificar el tamaño y las texturas del skybox.
- unsigned int getCubemapTexture () const

  Obtiene el identificador de la textura cubemap cargada para el skybox.

Public Member Functions inherited from udit::Cube

- Cube ()

  Constructor por defecto.
- Cube (bool inverted)

  Constructor con opción de invertir las normales.
- Cube (float size)

  Constructor con tamaño especificado.
- Cube (float size, bool inverted)

  Constructor con tamaño y opción de invertir las normales.

Public Member Functions inherited from udit::Mesh

- **Mesh** ()

  Constructor por defecto.
- Mesh (std::string &path)

  Constructor que carga una malla desde un archivo.
- virtual ~Mesh ()

  Destructor de la clase.
- virtual void translate (glm::vec3 translation)

  Realiza una traslación de la malla.
- virtual void rotate (glm::vec3 rotation, float angle)

  Rota la malla.
- virtual void orbit (glm::vec3 center, float distance, float speed)

  Orbita la malla alrededor de un punto.
- virtual void scale (glm::vec3 scale)

  Escala la malla.
- virtual void update ()

  Actualiza la malla.
- virtual void render (glm::mat4 view_matrix)

  Renderiza la malla.
- virtual void resize (glm::mat4 projection_matrix)

  Ajusta la matriz de proyección.
- virtual void set_shader (std::shared_ptr< udit::Shader > shader)

  Asocia un shader a la malla.
- GLuint get_shader_program_id () const

  Obtiene el ID del programa del shader asociado.
- std::vector< GLint > get_shader_matrix_ids ()

  Obtiene los IDs de las matrices del shader asociadas a la malla.
- glm::mat4 get_model_view_matrix () const

  Obtiene la matriz de transformación del modelo.
- void set_model_view_matrix (glm::mat4 matrix)

  Establece la matriz de transformación del modelo.
- void set_mesh_type (MeshType type)

  Establece el tipo de malla.

Additional Inherited Members

Static Public Member Functions inherited from udit::Mesh

- static std::shared_ptr< Mesh > make_mesh (MeshType type, const std::string &path="")

  Crea una malla de un tipo específico.

Protected Member Functions inherited from udit::Mesh

- void create_mesh (std::string mesh_name="")

  Crea los VBOs y el VAO necesarios para la malla.

Protected Attributes inherited from udit::Mesh

- std::vector< glm::vec3 > **coordinates**

  Vectores que almacenan las coordenadas de los vértices, colores, normales, índices y coordenadas de textura.
- std::vector< glm::vec3 > **colors**
- std::vector< glm::vec3 > **normals**
- std::vector< GLuint > **indices**
- std::vector< glm::vec2 > **texture_uvs**
- GLsizei **number_of_vertices**

  Número total de vértices de la malla.

4.17.1 Detailed Description

Representa un skybox, un cubo con texturas aplicadas en sus seis caras.

Un skybox es un cubo que rodea la escena y sirve como fondo inmersivo en un entorno 3D. La clase `Skybox` hereda de la clase `Cube`, y se encarga de cargar las texturas y mostrar el cielo en una escena utilizando un cubo con caras texturizadas.

4.17.2 Constructor & Destructor Documentation

4.17.2.1 Skybox() [1/2]

```
udit::Skybox::Skybox ()
```

Constructor por defecto.

Este constructor crea un skybox con un tamaño por defecto y sin texturas cargadas.

4.17.2.2 Skybox() [2/2]

```
udit::Skybox::Skybox (
            float size,
            const std::vector< std::string > & faces)
```

Constructor que permite especificar el tamaño y las texturas del skybox.

Parameters

size	Tamaño del cubo que representará el skybox.
faces	Vector de rutas a las texturas que serán aplicadas a las caras del skybox.

4.17.3 Member Function Documentation

4.17.3.1 getCubemapTexture()

```
unsigned int udit::Skybox::getCubemapTexture () const  [inline]
```

Obtiene el identificador de la textura cubemap cargada para el skybox.

Returns

Identificador de la textura cubemap.

The documentation for this class was generated from the following files:

- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Skybox.hpp
- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Skybox.cpp

4.18 Texture Class Reference

Representa una textura en OpenGL.

```
#include <Texture.hpp>
```

Public Member Functions

- Texture (const std::string &path, GLenum texture_unit, Texture_Type type=Texture_Type::COLOR)
  Constructor que crea la textura a partir de un archivo.
- ~**Texture** ()
  Destructor que libera la textura cargada.
- void bind () const
  Enlaza la textura a la unidad de textura actual.
- void unbind () const
  Desenlaza la textura de la unidad de textura.
- void load_texture ()
  Carga la textura desde el archivo especificado.
- void set_type (Texture_Type type)
  Establece el tipo de la textura (COLOR o HEIGHT).
- bool is_loaded ()
  Indica si la textura ha sido cargada exitosamente.

Public Attributes

- GLuint **texture_id**
  Identificador de la textura cargada.
- GLenum **texture_unit**
  Unidad de textura a la que la textura está asignada.
- std::string **file_path**
  Ruta del archivo de la textura.

4.18.1 Detailed Description

Representa una textura en OpenGL.

La clase `Texture` permite la carga y manejo de texturas en OpenGL. Estas texturas pueden ser utilizadas en diferentes tipos de materiales y objetos 3D dentro de la escena. La clase gestiona el enlace y des-enlace de texturas, permitiendo su uso en shaders.

4.18.2 Constructor & Destructor Documentation

4.18.2.1 Texture()

```
Texture::Texture (
            const std::string & path,
            GLenum texture_unit,
            Texture_Type type = Texture_Type::COLOR)
```

Constructor que crea la textura a partir de un archivo.

Este constructor carga la textura desde una ruta de archivo específica. Se puede especificar el tipo de textura (por defecto es COLOR).

Parameters

path	Ruta al archivo de la textura (imagen).
texture_unit	Unidad de textura (GL_TEXTURE0, GL_TEXTURE1, etc.).
type	Tipo de la textura (por defecto COLOR).

4.18.3 Member Function Documentation

4.18.3.1 bind()

```
void Texture::bind () const
```

Enlaza la textura a la unidad de textura actual.

Este método enlaza la textura al contexto de OpenGL, permitiendo que sea utilizada por los shaders para renderizar objetos con la textura aplicada.

4.18.3.2 is_loaded()

```
bool udit::Texture::is_loaded ()  [inline]
```

Indica si la textura ha sido cargada exitosamente.

Returns

true si la textura ha sido cargada, false en caso contrario.

4.18.3.3 load_texture()

```
void Texture::load_texture ()
```

Carga la textura desde el archivo especificado.

Este método lee el archivo de imagen y crea una textura en OpenGL. Se encarga de configurar los parámetros y cargar la imagen a la memoria de GPU.

4.18.3.4 set_type()

```
void udit::Texture::set_type (
            Texture_Type type)  [inline]
```

Establece el tipo de la textura (COLOR o HEIGHT).

Parameters

type	Tipo de la textura a establecer.

4.18.3.5 unbind()

```
void Texture::unbind () const
```

Desenlaza la textura de la unidad de textura.

Este método desenlaza la textura, liberando la unidad de textura para ser utilizada por otras texturas.

The documentation for this class was generated from the following files:

- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Texture.hpp
- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Texture.cpp

4.19 udit::Texture Class Reference

Representa una textura en OpenGL.

`#include <Texture.hpp>`

Public Member Functions

- Texture (const std::string &path, GLenum texture_unit, Texture_Type type=Texture_Type::COLOR)
  Constructor que crea la textura a partir de un archivo.
- ~**Texture** ()
  Destructor que libera la textura cargada.
- void bind () const
  Enlaza la textura a la unidad de textura actual.
- void unbind () const
  Desenlaza la textura de la unidad de textura.
- void load_texture ()
  Carga la textura desde el archivo especificado.
- void set_type (Texture_Type type)
  Establece el tipo de la textura (COLOR o HEIGHT).
- bool is_loaded ()
  Indica si la textura ha sido cargada exitosamente.

Public Attributes

- GLuint **texture_id**

  Identificador de la textura cargada.
- GLenum **texture_unit**

  Unidad de textura a la que la textura está asignada.
- std::string **file_path**

  Ruta del archivo de la textura.

4.19.1 Detailed Description

Representa una textura en OpenGL.

La clase `Texture` permite la carga y manejo de texturas en OpenGL. Estas texturas pueden ser utilizadas en diferentes tipos de materiales y objetos 3D dentro de la escena. La clase gestiona el enlace y des-enlace de texturas, permitiendo su uso en shaders.

4.19.2 Constructor & Destructor Documentation

4.19.2.1 Texture()

```
Texture::Texture (
            const std::string & path,
            GLenum texture_unit,
            Texture_Type type = Texture_Type::COLOR)
```

Constructor que crea la textura a partir de un archivo.

Este constructor carga la textura desde una ruta de archivo específica. Se puede especificar el tipo de textura (por defecto es COLOR).

Parameters

path	Ruta al archivo de la textura (imagen).
texture_unit	Unidad de textura (GL_TEXTURE0, GL_TEXTURE1, etc.).
type	Tipo de la textura (por defecto COLOR).

4.19.3 Member Function Documentation

4.19.3.1 bind()

```
void Texture::bind () const
```

Enlaza la textura a la unidad de textura actual.

Este método enlaza la textura al contexto de OpenGL, permitiendo que sea utilizada por los shaders para renderizar objetos con la textura aplicada.

4.19.3.2 is_loaded()

```
bool udit::Texture::is_loaded ()  [inline]
```

Indica si la textura ha sido cargada exitosamente.

Returns

true si la textura ha sido cargada, false en caso contrario.

4.19.3.3 load_texture()

```
void Texture::load_texture ()
```

Carga la textura desde el archivo especificado.

Este método lee el archivo de imagen y crea una textura en OpenGL. Se encarga de configurar los parámetros y cargar la imagen a la memoria de GPU.

4.19.3.4 set_type()

```
void udit::Texture::set_type (
            Texture_Type type)  [inline]
```

Establece el tipo de la textura (COLOR o HEIGHT).

Parameters

type	Tipo de la textura a establecer.

4.19.3.5 unbind()

```
void Texture::unbind () const
```

Desenlaza la textura de la unidad de textura.

Este método desenlaza la textura, liberando la unidad de textura para ser utilizada por otras texturas.

The documentation for this class was generated from the following files:

- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Texture.hpp
- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Texture.cpp

4.20 udit::Window Class Reference

Classes

- struct OpenGL_Context_Settings

Public Types

- enum **Position** { **UNDEFINED** = SDL_WINDOWPOS_UNDEFINED , **CENTERED** = SDL_WINDOWPOS_↩ CENTERED }

Public Member Functions

- **Window** (const std::string &title, int left_x, int top_y, unsigned width, unsigned height, const OpenGL_Context_Settings &context_details)
- Window (const char *title, int left_x, int top_y, unsigned width, unsigned height, const OpenGL_Context_Settings &context_details)

  Constructor de la ventana.
- ~**Window** ()

  Destructor de la ventana.
- **Window** (const Window &)=delete
- Window & **operator=** (const Window &)=delete
- **Window** (Window &&other) noexcept
- Window & **operator=** (Window &&other) noexcept
- void **swap_buffers** ()

  Intercambiar los buffers de OpenGL.

4.20.1 Constructor & Destructor Documentation

4.20.1.1 Window()

```
udit::Window::Window (
            const char * title,
            int left_x,
            int top_y,
            unsigned width,
            unsigned height,
            const OpenGL_Context_Settings & context_details)
```

Constructor de la ventana.

Parameters

title	Titulo de la ventana
left_x	Posicion de la ventana en el eje x
top_y	Posicion de la ventana en el eje y
width	Ancho de la ventana
height	Alto de la ventana
context_details	Ajustes el contexto de OpenGL

The documentation for this class was generated from the following files:

- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Window.hpp
- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Window.cpp

4.21 Window Class Reference

Classes

- struct OpenGL_Context_Settings

Public Types

- enum **Position** { **UNDEFINED** = SDL_WINDOWPOS_UNDEFINED , **CENTERED** = SDL_WINDOWPOS_↩ CENTERED }

Public Member Functions

- **Window** (const std::string &title, int left_x, int top_y, unsigned width, unsigned height, const OpenGL_Context_Settings &context_details)
- Window (const char *title, int left_x, int top_y, unsigned width, unsigned height, const OpenGL_Context_Settings &context_details)

  Constructor de la ventana.
- **Window** (const Window &)=delete
- **Window** (Window &&other) noexcept
- ~**Window** ()

  Destructor de la ventana.
- Window & **operator=** (const Window &)=delete
- Window & **operator=** (Window &&other) noexcept
- void **swap_buffers** ()

  Intercambiar los buffers de OpenGL.

4.21.1 Constructor & Destructor Documentation

4.21.1.1 Window()

```
udit::Window::Window (
            const char * title,
            int left_x,
            int top_y,
            unsigned width,
            unsigned height,
            const OpenGL_Context_Settings & context_details)
```

Constructor de la ventana.

Parameters

title	Titulo de la ventana
left_x	Posicion de la ventana en el eje x
top_y	Posicion de la ventana en el eje y
width	Ancho de la ventana
height	Alto de la ventana
context_details	Ajustes el contexto de OpenGL

The documentation for this class was generated from the following files:

- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Window.hpp
- /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Window.cpp

Chapter 5

File Documentation

5.1 Camera.hpp

```
00001 //
00002 //  Camera.hpp
00003 //  GL_Scene
00004 //
00005 //  Created by Alonso García on 23/12/24.
00006 //
00007
00008 #pragma once
00009
00010 #include "glm.hpp"
00011 #include <gtc/matrix_transform.hpp>
00012 #include <gtc/constants.hpp>
00013
00014 enum class CameraMovement
00015 {
00016     FORWARD,
00017     BACKWARD,
00018     LEFT,
00019     RIGHT,
00020     UP,
00021     DOWN
00022 };
00032 class Camera
00033 {
00034 public:
00040     glm::vec3 position;
00041
00048     glm::vec3 front;
00049
00055     glm::vec3 up;
00056
00063     glm::vec3 right;
00064
00071     glm::vec3 world_up;
00072
00078     float yaw;
00079
00085     float pitch;
00086
00092     float movement_speed;
00093
00099     float mouse_sensitivity;
00100
00106     float zoom;
00107
00118     Camera(glm::vec3 start_position, glm::vec3 up_direction, float start_yaw, float start_pitch);
00119
00128     glm::mat4 get_view_matrix() const;
00129
00139     void process_keyboard(CameraMovement direction, float delta_time);
00140
00151     void process_mouse_movement(float x_offset, float y_offset, bool constraint_pitch = true);
00152
00153 private:
00160     void update_camera_vectors();
00161 };
```

5.2 Cube.hpp

```
00001 //
00002 //  Cube.hpp
00003 //  GL_Geometry
00004 //
00005 //  Created by Alonso García on 21/12/24.
00006 //
00007
00008 #pragma once
00009
00010 #include "glad.h"
00011
00012 #include "Mesh.hpp"
00013
00021 namespace udit
00022 {
00023     class Cube : public Mesh
00024     {
00025     private:
00031         float size;
00032
00033     public:
00039         Cube();
00040
00049         Cube(bool inverted);
00050
00058         Cube(float size);
00059
00069         Cube(float size, bool inverted);
00070
00071     private:
00080         void create_cube(bool inverted = false);
00081     };
00082 }
```

5.3 EventHandler.hpp

```
00001 //
00002 //  EventHandler.hpp
00003 //  GL_Scene
00004 //
00005 //  Created by Alonso García on 23/12/24.
00006 //
00007
00008 #pragma once
00009
00010 #include "SDL.h"
00011 #include "glm.hpp"
00012 #include "Camera.hpp"
00013
00022 class EventHandler
00023 {
00024 public:
00034     EventHandler(Camera& camera)
00035         : camera(camera), first_mouse(true), last_x(0.0f), last_y(0.0f) {}
00036
00049     void handle_events(bool & running, float delta_time);
00050
00051 private:
00055     Camera & camera;
00056
00063     bool first_mouse;
00064
00068     float last_x;
00069
00073     float last_y;
00074
00084     void process_mouse_motion(const SDL_Event & event);
00085
00097     void process_keyboard(const Uint8 * keystate, float delta_time);
00098 };
```

5.4 Light.hpp

```
00001 //
00002 //  NewLight.hpp
```

```
00003 //  GL_Scene
00004 //
00005 //  Created by Alonso García on 13/1/25.
00006 //
00007
00008 #pragma once
00009
00010 #include "glm.hpp"
00011 #include <iostream>
00012
00013 #include "Cube.hpp"
00014
00015 namespace udit
00016 {
00026     class Light {
00027     private:
00031         glm::vec3 position;
00032
00036         glm::vec3 color;
00037
00043         float ambientIntensity;
00044
00051         float diffuseIntensity;
00052
00053     public:
00065         Light(const glm::vec3& pos, const glm::vec3& col, float ambient, float diffuse);
00066
00080         static std::shared_ptr <Light> make_light(const glm::vec3& pos, const glm::vec3& col, float
      ambient, float diffuse);
00081
00091         void send_to_shader(GLuint program_id) const;
00092     };
00093
00094 }
00095
```

5.5 Mesh.hpp

```
00001 //
00002 //  Mesh.hpp
00003 //  GL_Geometry
00004 //
00005 //  Created by Alonso García on 11/12/24.
00006 //
00007
00008 #pragma once
00009
00010 #include <vector>
00011
00012 #include "glm.hpp"
00013 #include <gtc/matrix_transform.hpp>
00014 #include <gtc/type_ptr.hpp>
00015 #include "glad.h"
00016
00017 #include "Shader.hpp"
00018
00019 namespace udit
00020 {
00021     enum class MeshType
00022     {
00023         BASIC,
00024         MESH,
00025         TERRAIN,
00026         SKYBOX
00027     };
00028
00039     class Mesh
00040     {
00041     private:
00045         enum
00046         {
00047             COORDINATES_VBO,
00048             COLORS_VBO,
00049             NORMALS_VBO,
00050             INDEXES_VBO,
00051             TEXTURE_UV_VBO,
00052             VBO_COUNT
00053         };
00054
00058         MeshType m_mesh_type;
00059
00060     protected:
00064         std::vector<glm::vec3> coordinates;
```

```
00065         std::vector<glm::vec3> colors;
00066         std::vector<glm::vec3> normals;
00067         std::vector<GLuint> indices;
00068         std::vector<glm::vec2> texture_uvs;
00069 
00073         GLsizei number_of_vertices;
00074 
00080         void create_mesh(std::string mesh_name = "");
00081 
00082     private:
00086         GLuint vbo_ids[VBO_COUNT];
00087         GLuint vao_id;
00088 
00092         glm::mat4 model_view_matrix;
00093         glm::mat4 normal_matrix;
00094 
00098         float angle;
00099 
00103         std::shared_ptr < udit::Shader > m_shader;
00104 
00105     public:
00109         Mesh();
00110 
00119         Mesh(std::string & path);
00120 
00131         static std::shared_ptr <Mesh> make_mesh(MeshType type, const std::string &path = "");
00132 
00138         virtual ~Mesh();
00139 
00147         virtual void translate(glm::vec3 translation);
00148 
00157         virtual void rotate(glm::vec3 rotation, float angle);
00167         virtual void orbit(glm::vec3 center, float distance, float speed);
00168 
00176         virtual void scale(glm::vec3 scale);
00177 
00183         virtual void update();
00184 
00192         virtual void render(glm::mat4 view_matrix);
00193 
00201         virtual void resize(glm::mat4 projection_matrix);
00202 
00210         virtual void set_shader(std::shared_ptr < udit::Shader > shader);
00211 
00217         GLuint get_shader_program_id() const;
00218 
00226         std::vector < GLint > get_shader_matrix_ids();
00227 
00233         glm::mat4 get_model_view_matrix() const { return model_view_matrix; }
00234 
00240         void set_model_view_matrix(glm::mat4 matrix) { model_view_matrix = matrix; }
00241 
00247         void set_mesh_type(MeshType type) { m_mesh_type = type; }
00248     };
00249 
00250 }
```

5.6 Plane.hpp

```
00001 //
00002 //  Plane.hpp
00003 //  GL_Geometry
00004 //
00005 //  Created by Alonso García on 11/12/24.
00006 //
00007 
00008 #pragma once
00009 
00010 #include "glad.h"
00011 
00012 #include "Mesh.hpp"
00013 
00014 namespace udit
00015 {
00025     class Plane : public Mesh
00026     {
00027     private:
00031         float width;
00032 
00036         float height;
00037 
00041         unsigned columns;
00042 
```

```
00046         unsigned rows;
00047
00048     public:
00054         Plane();
00055
00063         Plane(float size);
00064
00075         Plane(float width, float height, unsigned columns, unsigned rows);
00076
00077     private:
00084         void create_plane();
00085     };
00086
00087 }
```

5.7 /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Scene.hpp File Reference

Clase que representa una escena 3D, gestionando objetos como el fondo, terreno, luz, etc.

```
#include <string>
#include "Shader.hpp"
#include "Light.hpp"
#include "Skybox.hpp"
#include "Plane.hpp"
```

Classes

- class udit::Scene

  Representa una escena 3D con un skybox, terreno, luz y otros elementos.

5.7.1 Detailed Description

Clase que representa una escena 3D, gestionando objetos como el fondo, terreno, luz, etc.

Esta clase es responsable de mantener y gestionar la escena 3D, incluyendo el fondo (skybox), el terreno, las luces y el resto de elementos gráficos. Permite actualizar, renderizar y redimensionar la escena, además de configurar las matrices de vista y proyección, y la luz del entorno.

5.8 Scene.hpp

Go to the documentation of this file.

```
00001 //
00002 //  Scene.hpp
00003 //  GL_Geometry
00004 //
00005 //  Created by Alonso García on 9/12/24.
00006 //
00015
00016 #pragma once
00017
00018 #include <string>
00019 #include "Shader.hpp"
00020 #include "Light.hpp"
00021 #include "Skybox.hpp"
00022 #include "Plane.hpp"
00023
```

```
00024 namespace udit
00025 {
00034     class Scene
00035     {
00036     private:
00043         std::vector<std::string> skybox_faces =
00044         {
00045                "skybox_east.jpg", "skybox_west.jpg", "skybox_up.jpg",
00046                "skybox_down.jpg", "skybox_north.jpg", "skybox_south.jpg"
00047         };
00048
00050         float angle = 0.0f;
00051
00053         std::shared_ptr<Skybox> skybox;
00054
00056         std::shared_ptr<Plane> terrain;
00057
00059         std::shared_ptr<Plane> floor;
00060
00062         std::shared_ptr<Mesh> bull;
00063
00065         std::shared_ptr<Mesh> statue;
00066
00068         std::shared_ptr<Mesh> car;
00069
00071         std::shared_ptr<Light> light;
00072
00074         unsigned width, height;
00075
00077         glm::mat4 view_matrix;
00078
00080         glm::mat4 projection_matrix;
00081
00082     public:
00090         Scene(unsigned width, unsigned height);
00091
00098         void update();
00099
00106         void render();
00107
00115         void resize(unsigned width, unsigned height);
00116
00123         void set_view_matrix(const glm::mat4& view);
00124
00131         void set_projection_matrix(const glm::mat4& projection);
00132
00140         void set_lights(GLuint shader_program_id);
00141     };
00142 }
```

5.9 /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_↩ Scene/code/Shader.hpp File Reference

Clase que representa un shader en OpenGL, gestionando la compilación y uso de programas de sombreado.

```
#include <iostream>
#include "glad.h"
#include "Texture.hpp"
```

Classes

- class udit::Shader

  Representa un shader program en OpenGL.

Enumerations

- enum class udit::ShaderType {
  SKYBOX , GEOMETRY , SINGLE_TEXTURE , TERRAIN ,
  DEFAULT }

  Enumeración que define los diferentes tipos de shaders.

5.9.1 Detailed Description

Clase que representa un shader en OpenGL, gestionando la compilación y uso de programas de sombreado.

La clase `Shader` gestiona la creación, compilación y uso de shaders en OpenGL, incluyendo tanto el vertex shader como el fragment shader. Además, permite la gestión de texturas asociadas al shader y la configuración de matrices para la proyección, vista y normales en el contexto de la cámara.

5.9.2 Enumeration Type Documentation

5.9.2.1 ShaderType

```
enum class udit::ShaderType  [strong]
```

Enumeración que define los diferentes tipos de shaders.

Define los tipos de shaders que la clase `Shader` puede usar para diferentes efectos visuales, como el skybox, geometría, textura única, terreno y por defecto.

Enumerator

SKYBOX	Shader para el skybox.
GEOMETRY	Shader para la geometría.
SINGLE_TEXTURE	Shader para una textura única.
TERRAIN	Shader para el terreno.
DEFAULT	Shader por defecto.

5.10 Shader.hpp

Go to the documentation of this file.

```
00001 //
00002 //  Shader.hpp
00003 //  GL_Geometry
00004 //
00005 //  Created by Alonso García on 11/12/24.
00006 //
00015
00016 #pragma once
00017
00018 #include <iostream>
00019 #include "glad.h"
00020 #include "Texture.hpp"
00021
00022 namespace udit
00023 {
00031     enum class ShaderType
00032     {
00033         SKYBOX,
00034         GEOMETRY,
00035         SINGLE_TEXTURE,
00036         TERRAIN,
00037         DEFAULT
00038     };
00039
00049     class Shader
00050     {
00051     private:
00053         GLuint program_id;
00054
00056         ShaderType m_type;
00057
```

```
00059         std::string m_name;
00060
00062         std::string m_vertex_source;
00063
00065         std::string m_fragment_source;
00066
00068         GLint  model_view_matrix_id;
00069         GLint  projection_matrix_id;
00070         GLint  normal_matrix_id;
00071
00073         static const std::string    default_vertex_shader_code;
00074         static const std::string    default_fragment_shader_code;
00075
00077         std::vector <std::shared_ptr<Texture» textures;
00078
00079     public:
00086         Shader();
00087
00096         Shader(ShaderType type, const std::string & vertex_source, const std::string &
      fragment_source, const std::string & name);
00097
00103         ~Shader();
00104
00116         static std::shared_ptr < Shader > make_shader(
00117             udit::ShaderType type = udit::ShaderType::DEFAULT,
00118             const std::string & vertex_shader = "",
00119             const std::string & fragment_shader = "",
00120             const std::vector<std::string> & texture_paths = {""},
00121             const std::string & name = ""
00122         );
00123
00132         GLuint compile_shaders(const char * vertex_shader_code, const char * fragment_shader_code);
00133
00138         GLint get_model_view_matrix_id() { return model_view_matrix_id; }
00139
00144         GLint get_projection_matrix_id() { return projection_matrix_id; }
00145
00150         GLint get_normal_matrix_id() { return normal_matrix_id; }
00151
00156         GLuint get_program_id() const { return program_id; }
00157
00163         void set_texture(const std::shared_ptr<Texture> & texture);
00164
00170         void use() const;
00171
00176         void set_texture_scale(float scale);
00177
00182         bool has_textures() { return !textures.empty(); }
00183
00188         void set_name(const std::string & name) { m_name = name; }
00189
00194         std::string get_name() { return m_name; }
00195
00196     private:
00201         void show_compilation_error(GLuint shader_id);
00202
00207         void show_linkage_error(GLuint program_id);
00208     };
00209 }
```

5.11 /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_↩ Scene/code/Skybox.hpp File Reference

Clase para representar y gestionar un skybox en OpenGL.

```
#include "Cube.hpp"
#include <vector>
#include <string>
```

Classes

- class udit::Skybox

  Representa un skybox, un cubo con texturas aplicadas en sus seis caras.

5.11.1 Detailed Description

Clase para representar y gestionar un skybox en OpenGL.

La clase `Skybox` hereda de `Cube` y permite la carga y visualización de un cubo que actúa como el fondo de la escena, utilizando una serie de texturas que representan las caras del cielo. Se utiliza para crear una atmósfera inmersiva en la escena renderizada.

5.12 Skybox.hpp

Go to the documentation of this file.

```
00001 //
00002 //  Skybox.hpp
00003 //  GL_Scene
00004 //
00005 //  Created by Alonso García on 21/12/24.
00006 //
00015
00016 #pragma once
00017
00018 #include "Cube.hpp"
00019 #include <vector>
00020 #include <string>
00021
00022 namespace udit
00023 {
00032     class Skybox : public Cube
00033     {
00034     private:
00036         unsigned int cubemapTexture;
00037
00039         std::string filepath =
      "/Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/resources/skybox/";
00040
00041     public:
00047         Skybox();
00048
00055         Skybox(float size, const std::vector<std::string>& faces);
00056
00062         unsigned int getCubemapTexture() const { return cubemapTexture; }
00063
00064     private:
00074         void loadCubemap(const std::vector<std::string>& faces);
00075     };
00076 }
```

5.13 /Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/code/Texture.hpp File Reference

Clase para gestionar las texturas en OpenGL.

```
#include <string>
#include <glad.h>
```

Classes

- class udit::Texture

  Representa una textura en OpenGL.

Enumerations

- enum class udit::Texture_Type { COLOR , HEIGHT }

  Enum que define los tipos de texturas disponibles.

5.13.1 Detailed Description

Clase para gestionar las texturas en OpenGL.

La clase `Texture` permite la carga, enlace y liberación de texturas en OpenGL. Se utiliza para manejar imágenes que se aplican a los objetos 3D en la escena, permitiendo efectos visuales como color, relieve, etc.

5.13.2 Enumeration Type Documentation

5.13.2.1 Texture_Type

```
enum class udit::Texture_Type  [strong]
```

Enum que define los tipos de texturas disponibles.

Los tipos de texturas permiten diferenciar entre distintos tipos de efectos visuales:

- `COLOR`: Textura normal, utilizada para representar colores o imágenes en 3D.
- `HEIGHT`: Textura de altura, generalmente utilizada en mapas de relieve.

Enumerator

COLOR	Textura de color (imagen normal).
HEIGHT	Textura de altura (mapa de relieve).

5.14 Texture.hpp

Go to the documentation of this file.

```
00001 //
00002 //  Texture.hpp
00003 //  GL_Scene
00004 //
00005 //  Created by Alonso García on 24/12/24.
00006 //
00015
00016 #pragma once
00017
00018 #include <string>
00019 #include <glad.h>
00020
00021 namespace udit
00022 {
00031     enum class Texture_Type
00032     {
00033         COLOR,
00034         HEIGHT
00035     };
00036
00045     class Texture
00046     {
```

```
00047     private:
00049         bool loaded = false;
00050
00052         Texture_Type m_type;
00053
00054     public:
00065         Texture(const std::string & path, GLenum texture_unit, Texture_Type type =
      Texture_Type::COLOR);
00066
00068         ~Texture();
00069
00076         void bind() const;
00077
00084         void unbind() const;
00085
00087         GLuint texture_id;
00088
00090         GLenum texture_unit;
00091
00093         std::string file_path;
00094
00101         void load_texture();
00102
00108         void set_type(Texture_Type type) { m_type = type; }
00109
00115         bool is_loaded() { return loaded; }
00116     };
00117 }
00118
```

5.15 Window.hpp

```
00001 //
00002 //  Window.hpp
00003 //  GL_Geometry
00004 //
00005 //  Created by Alonso García on 9/12/24.
00006 //
00007
00008 #pragma once
00009
00010 #include <SDL.h>
00011 #include <string>
00012 #include <utility>
00013
00014 namespace udit
00015 {
00016
00017     class Window
00018     {
00019     public:
00020
00021         enum Position
00022         {
00023             UNDEFINED = SDL_WINDOWPOS_UNDEFINED,
00024             CENTERED  = SDL_WINDOWPOS_CENTERED,
00025         };
00026
00027         struct OpenGL_Context_Settings
00028         {
00029             unsigned version_major       = 3;
00030             unsigned version_minor       = 3;
00031             bool     core_profile        = true;
00032             unsigned depth_buffer_size   = 24;
00033             unsigned stencil_buffer_size = 0;
00034             bool     enable_vsync        = true;
00035         };
00036
00037     private:
00038
00039         SDL_Window  * window_handle;
00040         SDL_GLContext opengl_context;
00041
00042     public:
00043
00044         Window
00045         (
00046             const std::string & title,
00047             int      left_x,
00048             int      top_y,
00049             unsigned width,
00050             unsigned height,
00051             const OpenGL_Context_Settings & context_details
```

```
00052         )
00053         :
00054             Window(title.c_str (), left_x, top_y, width, height, context_details)
00055         {
00056         }
00057
00058         Window
00059         (
00060             const char * title,
00061             int      left_x,
00062             int      top_y,
00063             unsigned width,
00064             unsigned height,
00065             const OpenGL_Context_Settings & context_details
00066         );
00067
00068        ~Window();
00069
00070     public:
00071
00072         Window(const Window & ) = delete;
00073
00074         Window & operator = (const Window & ) = delete;
00075
00076         Window(Window && other) noexcept
00077         {
00078             this->window_handle  = std::exchange (other.window_handle,  nullptr);
00079             this->opengl_context = std::exchange (other.opengl_context, nullptr);
00080         }
00081
00082         Window & operator = (Window && other) noexcept
00083         {
00084             this->window_handle  = std::exchange (other.window_handle,  nullptr);
00085             this->opengl_context = std::exchange (other.opengl_context, nullptr);
00086
00087             return  * this;
00088         }
00089
00090     public:
00091
00092         void swap_buffers ();
00093
00094     };
00095
00096 }
```

Index