3D Graphics Engine for ESP32

Generated by Doxygen 1.13.2

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 basics Namespace Reference

Namespaces

- namespace internal

Typedefs

- typedef unsigned int Id

Functions

- template<size_t LENGTH>
  constexpr uint32_t static_fnv32 (const char(&chars)[LENGTH])
- template<size_t LENGTH>
  constexpr uint64_t static_fnv64 (const char(&chars)[LENGTH])
- template<size_t LENGTH>
  constexpr unsigned static_fnv (const char(&chars)[LENGTH])
- template<size_t LENGTH>
  uint32_t fnv32 (const char(&chars)[LENGTH])
- uint32_t fnv32 (const std::string &s)

5.1.1 Typedef Documentation

5.1.1.1 Id

```
typedef unsigned int basics::Id
```

5.1.2 Function Documentation

5.1.2.1 fnv32() [1/2]

```
template<size_t LENGTH>
uint32_t basics::fnv32 (
            const char(&) chars[LENGTH])
```

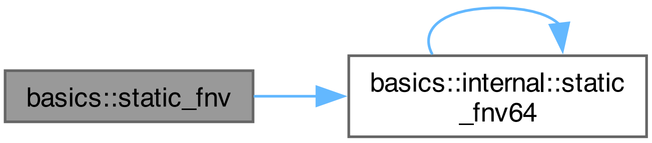
5.1.2.2 fnv32() [2/2]

```
uint32_t basics::fnv32 (
             const std::string & s)  [inline]
```

5.1.2.3 static_fnv()

```
template<size_t LENGTH>
unsigned basics::static_fnv (
             const char(&) chars[LENGTH])  [constexpr]
```

Here is the call graph for this function:



5.1.2.4 static_fnv32()

```
template<size_t LENGTH>
uint32_t basics::static_fnv32 (
             const char(&) chars[LENGTH])  [constexpr]
```

Implements the Fowler–Noll–Vo hash function (FNV-1a) which calculates a 32 bit hash code from a C string literal (or array of chars) at compile time.

Template Parameters

LENGTH	Length of the string literal or array of chars.

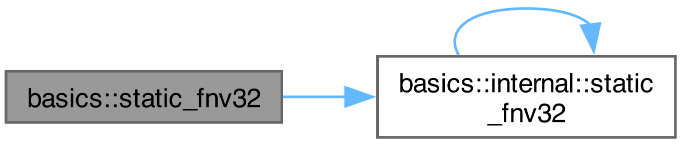
Parameters

chars	The string literal or array of chars to hash.

Returns

The hash code.

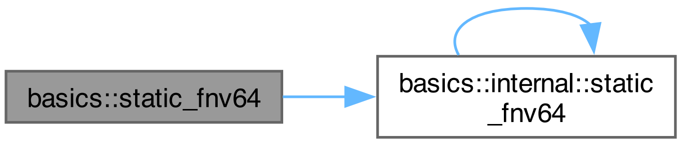
Here is the call graph for this function:



5.1.2.5 static_fnv64()

```
template<size_t LENGTH>
uint64_t basics::static_fnv64 (
            const char(&) chars[LENGTH])  [constexpr]
```

Here is the call graph for this function:



5.2 basics::internal Namespace Reference

Functions

- template<size_t LENGTH>
  constexpr uint32_t static_fnv32 (const char ∗chars)
- template<> constexpr uint32_t static_fnv32< 1 > (const char ∗)
- template<size_t LENGTH>
  constexpr uint64_t static_fnv64 (const char ∗chars)
- template<> constexpr uint64_t static_fnv64< 1 > (const char ∗)

Variables

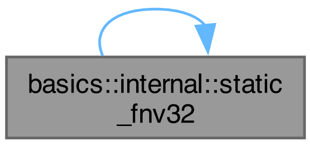
- constexpr uint32_t fnv_basis_32 = 0x811c9dc5u
- constexpr uint32_t fnv_prime_32 = 0x01000193u
- constexpr uint64_t fnv_basis_64 = 0xcbf29ce484222325u
- constexpr uint64_t fnv_prime_64 = 0x00000100000001b3u

5.2.1 Function Documentation

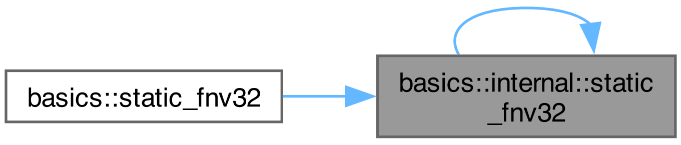
5.2.1.1 static_fnv32()

```
template<size_t LENGTH>
uint32_t basics::internal::static_fnv32 (
             const char * chars)  [constexpr]
```

Here is the call graph for this function:



Here is the caller graph for this function:



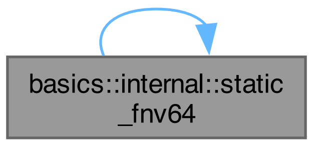
5.2.1.2 static_fnv32< 1 >()

```
template<>
uint32_t basics::internal::static_fnv32< 1 > (
             const char * )  [constexpr]
```

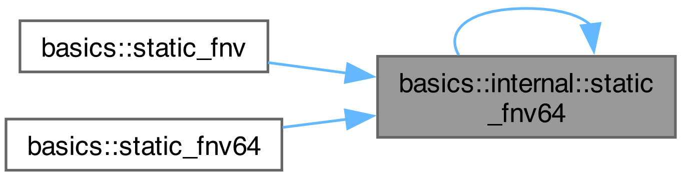
5.2.1.3 static_fnv64()

```
template<size_t LENGTH>
uint64_t basics::internal::static_fnv64 (
            const char * chars)  [constexpr]
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.1.4 static_fnv64< 1 >()

```
template<>
uint64_t basics::internal::static_fnv64< 1 > (
            const char * )  [constexpr]
```

5.2.2 Variable Documentation

5.2.2.1 fnv_basis_32

```
uint32_t basics::internal::fnv_basis_32 = 0x811c9dc5u  [constexpr]
```

5.2.2.2 fnv_basis_64

```
uint64_t basics::internal::fnv_basis_64 = 0xcbf29ce484222325u  [constexpr]
```

5.2.2.3 fnv_prime_32

```
uint32_t basics::internal::fnv_prime_32 = 0x01000193u  [constexpr]
```

5.2.2.4 fnv_prime_64

```
uint64_t basics::internal::fnv_prime_64 = 0x00000100000001b3u  [constexpr]
```

5.3 Ragot Namespace Reference

Classes

- class Assets

  Manages the asset paths for the application.
- class Camera

  Represents a camera in a 3D space, managing its properties and transformations.
- struct Camera_Transform

  Represents the transformation of a camera in 3D space.
- class Component

  Base class for components in the Ragot engine.
- struct coordinates_t

  Represents 2D coordinates.
- class Driver_ST7789

  Driver for the ST7789 LCD panel.
- class DriverEK79007

  Driver for the EK79007 LCD panel.
- class DriverLCD

  Base class for LCD drivers.
- class ExtrudeMesh

  Represents a 3D mesh created by extruding a 2D shape along a specified height. This class inherits from the Mesh class and provides methods to generate vertices and faces for the extruded mesh. It also includes methods for culling faces based on the camera's view direction and logging mesh information.
- struct face_t

  Represents a face in a 3D mesh.
- class Fragment_Shader

  Class for managing an OpenGL fragment shader.
- class FrameBuffer

  Class to manage a frame buffer for rendering graphics.
- class Logger

  Singleton logger class for the Ragot engine.
- class Mesh

  Represents a 3D mesh in the Ragot engine.
- struct mesh_info_t

  Represents information about a mesh.
- class MeshSerializer

  Singleton class to serialize Mesh objects to OBJ file format.
- class Node

  Represents a node in a scene graph for 3D rendering.

- class PSRAMAllocator

  Custom memory allocator for PSRAM.
- class Rasterizer

  Class for rasterizing polygons in a frame buffer.
- class Renderer

  Class for rendering scenes in the Ragot engine.
- class RevolutionMesh

  Class for generating revolution meshes.
- class Scene

  Class for managing a 3D scene.
- class Shader

  Class for managing an OpenGL shader.
- class Shader_Program

  Class for managing an OpenGL shader program.
- class Sync_Queue

  A thread-safe queue implementation.
- class Thread_Pool

  A thread pool for managing concurrent tasks.
- class Transform

  A class representing a 3D transformation with position, rotation, and scale.
- struct transform_t

  Represents a transformation in 3D space.
- class Vertex_Shader

  Class for managing an OpenGL vertex shader.
- struct vertex_t

  Represents a vertex in 3D space.
- class Window

  Class for managing an SDL window with OpenGL context.

Typedefs

- using RGB565 = uint16_t
- using RGB888 = uint32_t
- using RGBA8888 = uint32_t
- using RGB8 = uint8_t

  Color Index.
- using Matrix4x4 = glm::mat4

Enumerations

- enum render_flag_t : uint8_t { RENDER_NONE , RENDER_REVOLUTION , RENDER_EXTRUDE , RENDER_MAX }

  Flags for rendering types.
- enum Buffer : uint8_t { CURRENT_BUFFER = (1 << 0) , NEXT_BUFFER = (1 << 1) , MAX_BUFFER = (1 << 2) }

  Enum to represent the different buffers in a frame buffer.

Functions

- static bool panel_refresh_callback (esp_lcd_panel_handle_t panel, esp_lcd_dpi_panel_event_data_t ∗edata, void ∗user_ctx)
- static bool panel_refresh_callback (esp_lcd_panel_io_handle_t panel, esp_lcd_panel_io_event_data_↩ t ∗edata, void ∗user_ctx)
- template<typename T, uint16_t F1, typename U, uint16_t F2>
  bool operator== (const PSRAMAllocator< T, F1 > &, const PSRAMAllocator< U, F2 > &)

  Equality operator for PSRAMAllocator.
- template<typename T, uint16_t F1, typename U, uint16_t F2>
  bool operator!= (const PSRAMAllocator< T, F1 > &a, const PSRAMAllocator< U, F2 > &b)

  Inequality operator for PSRAMAllocator.
- template<typename Inside, typename Intersect>
  static std::vector< glm::fvec4 > clipAgainstPlane (const std::vector< glm::fvec4 > &in, Inside inside, Intersect intersect)

Variables

- Assets & assets = Assets::instance()
- constexpr float PI = 3.141592653f

  Mathematical constant PI.
- static const char ∗ TAG = "DriverEK79007"
- Logger & logger = Logger::instance()
- MeshSerializer & serializer = MeshSerializer::instance()
- template<class COLOR_BUFFER_TYPE>
  int Rasterizer< COLOR_BUFFER_TYPE >::offset_cache0 [1024]
- template<class COLOR_BUFFER_TYPE>
  int Rasterizer< COLOR_BUFFER_TYPE >::offset_cache1 [1024]
- template<class COLOR_BUFFER_TYPE>
  int Rasterizer< COLOR_BUFFER_TYPE >::z_cache0 [1024]
- template<class COLOR_BUFFER_TYPE>
  int Rasterizer< COLOR_BUFFER_TYPE >::z_cache1 [1024]
- static const char ∗ RENDERER_TAG = "Renderer"
- static const char ∗ MAIN_TAG = "Main"
- Thread_Pool & thread_pool = Thread_Pool::instance ()

5.3.1 Typedef Documentation

5.3.1.1 Matrix4x4

```
using Ragot::Matrix4x4 = glm::mat4
```

5.3.1.2 RGB565

```
using Ragot::RGB565 = uint16_t
```

5.3.1.3 RGB8

```
using Ragot::RGB8 = uint8_t
```

Color Index.

5.3.1.4 RGB888

```
using Ragot::RGB888 = uint32_t
```

5.3.1.5 RGBA8888

```
using Ragot::RGBA8888 = uint32_t
```

5.3.2 Enumeration Type Documentation

5.3.2.1 Buffer

```
enum Ragot::Buffer : uint8_t
```

Enum to represent the different buffers in a frame buffer.

This enum defines the constants for the current buffer, next buffer, and maximum buffer.

Enumerator

CURRENT_BUFFER	
NEXT_BUFFER	
MAX_BUFFER	

5.3.2.2 render_flag_t

```
enum Ragot::render_flag_t : uint8_t
```

Flags for rendering types.

This enumeration defines different rendering flags that can be used to specify how a mesh should be rendered.

Enumerator

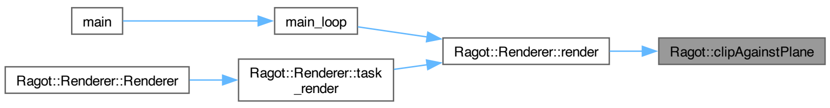
RENDER_NONE	
RENDER_REVOLUTION	
RENDER_EXTRUDE	
RENDER_MAX	

5.3.3 Function Documentation

5.3.3.1 clipAgainstPlane()

```
template<typename Inside, typename Intersect>
static std::vector< glm::fvec4 > Ragot::clipAgainstPlane (
             const std::vector< glm::fvec4 > & in,
             Inside inside,
             Intersect intersect)  [static]
```

Here is the caller graph for this function:



5.3.3.2 operator"!=()

```
template<typename T, uint16_t F1, typename U, uint16_t F2>
bool Ragot::operator!= (
             const PSRAMAllocator< T, F1 > & a,
             const PSRAMAllocator< U, F2 > & b)
```

Inequality operator for PSRAMAllocator.

This operator checks if two PSRAMAllocators are not equal based on their flags.

Template Parameters

T	The type of the first allocator.
F1	The flag of the first allocator.
U	The type of the second allocator.
F2	The flag of the second allocator.

Parameters

a	The first PSRAMAllocator.
b	The second PSRAMAllocator.

Returns

true if the flags are not equal, false otherwise.

5.3.3.3 operator==()

```
template<typename T, uint16_t F1, typename U, uint16_t F2>
bool Ragot::operator== (
             const PSRAMAllocator< T, F1 > & ,
             const PSRAMAllocator< U, F2 > & )
```

Equality operator for PSRAMAllocator.

This operator checks if two PSRAMAllocators are equal based on their flags.

Template Parameters

T	The type of the first allocator.
F1	The flag of the first allocator.
U	The type of the second allocator.
F2	The flag of the second allocator.

Parameters

a	The first PSRAMAllocator.
b	The second PSRAMAllocator.

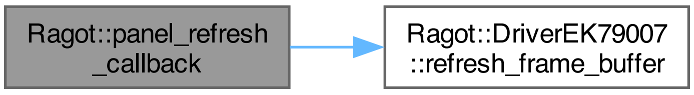
Returns

true if the flags are equal, false otherwise.

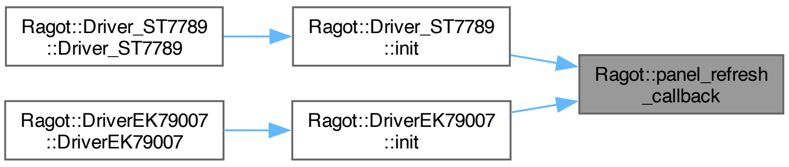
5.3.3.4 panel_refresh_callback() `[1/2]`

```
static bool Ragot::panel_refresh_callback (
            esp_lcd_panel_handle_t panel,
            esp_lcd_dpi_panel_event_data_t * edata,
            void * user_ctx)  [static]
```

Here is the call graph for this function:


Ragot::panel_refresh _callback
Ragot::DriverEK79007 ::refresh_frame_buffer

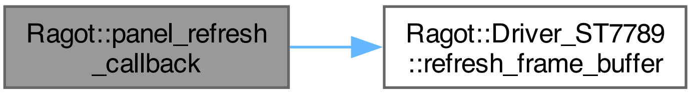
Here is the caller graph for this function:


Ragot::Driver_ST7789 ::Driver_ST7789
Ragot::Driver_ST7789 ::init
Ragot::DriverEK79007 ::DriverEK79007
Ragot::DriverEK79007 ::init
Ragot::panel_refresh _callback

5.3.3.5 panel_refresh_callback() [2/2]

```
static bool Ragot::panel_refresh_callback (
             esp_lcd_panel_io_handle_t panel,
             esp_lcd_panel_io_event_data_t * edata,
             void * user_ctx)  [static]
```

Here is the call graph for this function:


Ragot::panel_refresh _callback
Ragot::Driver_ST7789 ::refresh_frame_buffer

5.3.4 Variable Documentation

5.3.4.1 assets

```
Assets & Ragot::assets = Assets::instance()
```

5.3.4.2 logger

```
Logger & Ragot::logger = Logger::instance()
```

5.3.4.3 MAIN_TAG

```
const char* Ragot::MAIN_TAG = "Main"  [static]
```

5.3.4.4 PI

```
float Ragot::PI = 3.141592653f  [constexpr]
```

Mathematical constant PI.

This constant represents the value of (pi), which is approximately 3.141592653.

5.3.4.5 Rasterizer< COLOR_BUFFER_TYPE >::offset_cache0

```
template<class COLOR_BUFFER_TYPE>
int Ragot::Rasterizer< COLOR_BUFFER_TYPE >::offset_cache0[1024]
```

5.3.4.6 Rasterizer< COLOR_BUFFER_TYPE >::offset_cache1

```
template<class COLOR_BUFFER_TYPE>
int Ragot::Rasterizer< COLOR_BUFFER_TYPE >::offset_cache1[1024]
```

5.3.4.7 Rasterizer< COLOR_BUFFER_TYPE >::z_cache0

```
template<class COLOR_BUFFER_TYPE>
int Ragot::Rasterizer< COLOR_BUFFER_TYPE >::z_cache0[1024]
```

5.3.4.8 Rasterizer< COLOR_BUFFER_TYPE >::z_cache1

```
template<class COLOR_BUFFER_TYPE>
int Ragot::Rasterizer< COLOR_BUFFER_TYPE >::z_cache1[1024]
```

5.3.4.9 RENDERER_TAG

```
const char* Ragot::RENDERER_TAG = "Renderer"  [static]
```

5.3.4.10 serializer

```
MeshSerializer & Ragot::serializer = MeshSerializer::instance()
```

5.3.4.11 TAG

```
const char* Ragot::TAG = "DriverEK79007"  [static]
```

5.3.4.12 thread_pool

```
Thread_Pool & Ragot::thread_pool = Thread_Pool::instance ()
```

Chapter 6

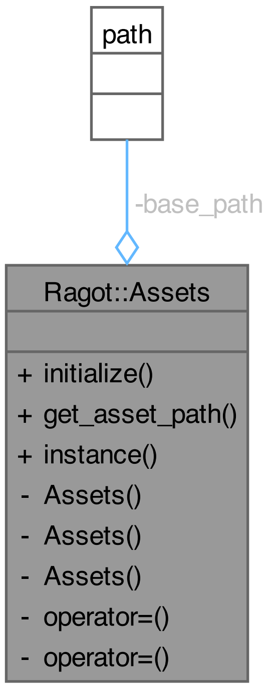
Class Documentation

6.1 Ragot::Assets Class Reference

Manages the asset paths for the application.

```
#include <Assets.hpp>
```

Collaboration diagram for Ragot::Assets:



Public Member Functions

- void initialize (const string &executable_file_path)

  Initializes the base path for assets based on the executable file path.
- path get_asset_path (const string &asset_name)

  Gets the full path of an asset by its name.

Static Public Member Functions

- static Assets & instance ()

  Returns the singleton instance of the Assets class.

Private Member Functions

- Assets ()=default

  Construct a new Assets object.
- Assets (const Assets &)=delete

  Destroy the Assets object.
- Assets (const Assets &&)=delete

  Move constructor is deleted to prevent moving the Assets instance.
- Assets & operator= (const Assets &)=delete

  Assignment operator is deleted to prevent copying the Assets instance.
- Assets & operator= (const Assets &&)=delete

  Move assignment operator is deleted to prevent moving the Assets instance.

Private Attributes

- path base_path = "./"

6.1.1 Detailed Description

Manages the asset paths for the application.

The Assets class provides a singleton instance to manage asset paths, allowing for easy retrieval of asset files. It initializes the base path based on the executable file path, and provides a method to get the full path of an asset by its name.

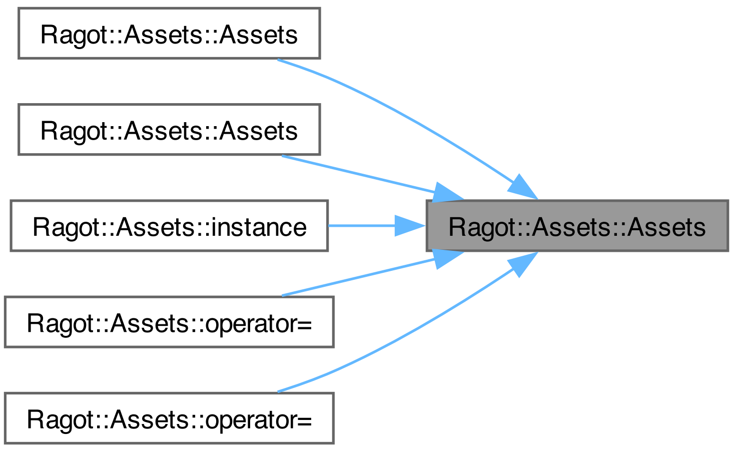
6.1.2 Constructor & Destructor Documentation

6.1.2.1 Assets() [1/3]

```
Ragot::Assets::Assets ()  [private], [default]
```

Construct a new Assets object.

Here is the caller graph for this function:

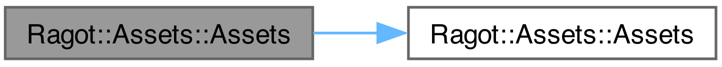


6.1.2.2 Assets() [2/3]

```
Ragot::Assets::Assets (
             const Assets & )  [private], [delete]
```

Destroy the Assets object.

Here is the call graph for this function:

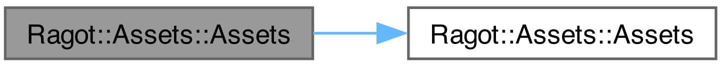


6.1.2.3 Assets() [3/3]

```
Ragot::Assets::Assets (
             const Assets && )  [private], [delete]
```

Move constructor is deleted to prevent moving the Assets instance.

Here is the call graph for this function:



6.1.3 Member Function Documentation

6.1.3.1 get_asset_path()

```
path Ragot::Assets::get_asset_path (
             const string & asset_name)  [inline]
```

Gets the full path of an asset by its name.

This method constructs the full path to an asset file by appending the asset name to the base path.

Parameters

asset_name	The name of the asset file.

Returns

path The full path to the asset file.

6.1.3.2 initialize()

```
void Ragot::Assets::initialize (
             const string & executable_file_path)  [inline]
```

Initializes the base path for assets based on the executable file path.

This method sets the base path to the "assets" directory relative to the executable's location. In debug mode, it uses a relative path to the assets directory. In release mode, it sets the base path to a specific directory.

Parameters

executable_file_path	The path of the executable file.

6.1.3.3 instance()

```
static Assets & Ragot::Assets::instance ()  [inline], [static]
```

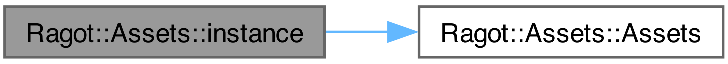
Returns the singleton instance of the Assets class.

This method ensures that only one instance of the Assets class exists throughout the application.

Returns

Assets& Reference to the singleton instance of Assets.

Here is the call graph for this function:

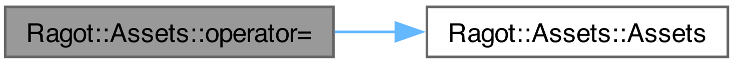


6.1.3.4 operator=() [1/2]

```
Assets & Ragot::Assets::operator= (
              const Assets && )  [private], [delete]
```

Move assignment operator is deleted to prevent moving the Assets instance.

Here is the call graph for this function:

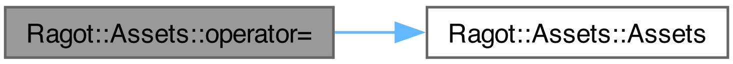


6.1.3.5 operator=() [2/2]

```
Assets & Ragot::Assets::operator= (
              const Assets & )  [private], [delete]
```

Assignment operator is deleted to prevent copying the Assets instance.

Here is the call graph for this function:



6.1.4 Member Data Documentation

6.1.4.1 base_path

```
path Ragot::Assets::base_path = "./"  [private]
```

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

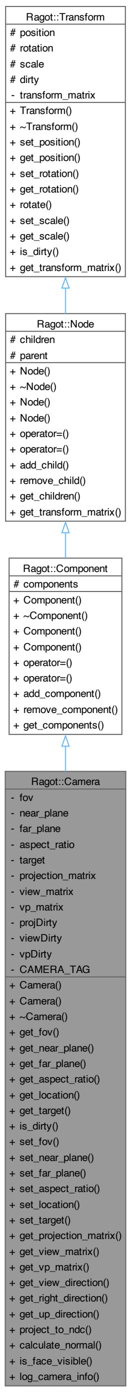
- main/Assets.hpp

6.2 Ragot::Camera Class Reference

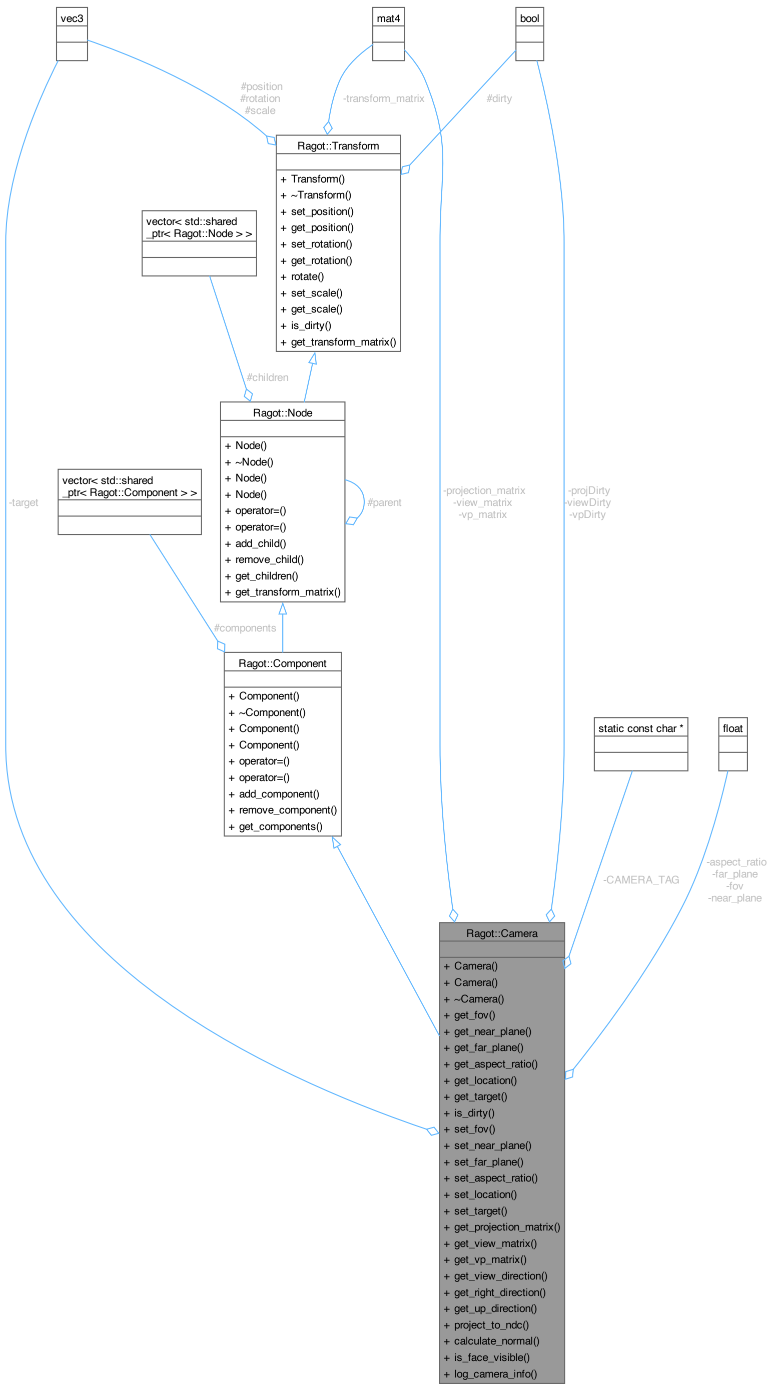
Represents a camera in a 3D space, managing its properties and transformations.

```
#include <Camera.hpp>
```

Inheritance diagram for Ragot::Camera:



Collaboration diagram for Ragot::Camera:



Public Types

- using Matrix4x4 = glm::mat4

Public Member Functions

- Camera ()=delete

Delete default constructor to prevent instantiation without parameters.
- Camera (float aspect_ratio=1.f, float near_plane=1.f, float far_plane=100.f, float fov_deg=60.f)

  Constructs a Camera object with specified parameters.
- ∼Camera ()=default

  Default destructor for Camera class.
- float get_fov () const

  Returns the camera's field of view in degrees.
- float get_near_plane () const

  Returns the camera's near clipping plane distance.
- float get_far_plane () const

  Returns the camera's far clipping plane distance.
- float get_aspect_ratio () const

  Returns the camera's aspect ratio (width/height).
- glm::vec3 get_location () const

  Returns the camera's position in world space.
- glm::vec3 get_target () const

  Returns the camera's target position in world space.
- bool is_dirty () const

  Returns the camera's position in world space.
- void set_fov (float deg)

  Set the fov object.
- void set_near_plane (float np)

  Set the near clipping plane distance.
- void set_far_plane (float fp)

  Set the far clipping plane distance.
- void set_aspect_ratio (float ar)

  Set the aspect ratio of the camera.
- void set_location (const glm::vec3 &p)

  Set the camera's position in world space.
- void set_target (const glm::vec3 &t)

  Set the camera's target position in world space.
- const Matrix4x4 & get_projection_matrix () const

  Get the projection matrix object.
- const Matrix4x4 & get_view_matrix () const

  Get the view matrix object.
- const Matrix4x4 & get_vp_matrix () const

  Get the vp matrix object.
- glm::vec3 get_view_direction () const

  Get the view direction object.
- glm::vec3 get_right_direction () const

  Get the right direction object.
- glm::vec3 get_up_direction () const

  Get the up direction object.
- glm::vec3 project_to_ndc (const glm::vec4 &worldPos) const
- vertex_t calculate_normal (const vertex_t &v1, const vertex_t &v2, const vertex_t &v3)

  Calculate the normal vector of a face defined by three vertices.
- bool is_face_visible (const vertex_t &v1, const vertex_t &v2, const vertex_t &v3)

  Check if a face defined by three vertices is visible from the camera's perspective.
- void log_camera_info () const

  Logs the camera's properties for debugging purposes.

Public Member Functions inherited from Ragot::Component

- Component ()=default

  Default constructor for the Component class.
- virtual ∼Component ()=default

  Default virtual destructor for the Component class.
- Component (const Component &)=delete

  Deleted copy constructor for the Component class.
- Component (const Component &&)=delete

  Deleted move constructor for the Component class.
- Component & operator= (const Component &)=delete

  Deleted assignment operator for the Component class.
- Component & operator= (const Component &&)=delete

  Deleted move assignment operator for the Component class.
- void add_component (std::shared_ptr< Component > component)

  Adds a component to the collection.
- void remove_component (std::shared_ptr< Component > component)

  Removes a component from the collection.
- const std::vector< std::shared_ptr< Component > > get_components () const

  Gets the collection of components.

Public Member Functions inherited from Ragot::Node

- Node ()=default

  Default constructor for Node. Initializes an empty node with no parent and no children.
- virtual ∼Node ()=default

  Default destructor for Node. Cleans up the node and its children.
- Node (const Node &)=delete

  Deleted copy constructor for Node. Prevents copying of Node instances.
- Node (const Node &&)=delete

  Deleted move constructor for Node. Prevents moving of Node instances.
- Node & operator= (const Node &)=delete

  Deleted assignment operator for Node. Prevents assignment of Node instances.
- Node & operator= (const Node &&)=delete

  Deleted move assignment operator for Node. Prevents moving of Node instances.
- void add_child (std::shared_ptr< Node > child)

  Get the parent node.
- void remove_child (std::shared_ptr< Node > child)

  Remove a child node.
- const std::vector< std::shared_ptr< Node > > & get_children () const

  Get the parent node.
- mat4 get_transform_matrix () override

  Get the transform matrix object.

Public Member Functions inherited from Ragot::Transform

- Transform ()

  Default constructor for the Transform class.
- virtual ∼Transform ()=default

  Virtual destructor for the Transform class.
- void set_position (const vec3 &pos)

  Sets the position of the object.
- vec3 get_position () const

  Gets the current position of the object.
- void set_rotation (const vec3 &rot)

  Moves the object by a specified vector.
- vec3 get_rotation () const

  Gets the current rotation of the object.
- void rotate (const float angle, const vec3 &axis)

  Rotates the object by a specified angle around a given axis.
- void set_scale (const vec3 &scale)

  Sets the scale of the object.
- vec3 get_scale () const

  Sets the scale of the object uniformly.
- bool is_dirty () const

  Checks if the transformation matrix is dirty (needs recalculation).

Private Attributes

- float fov

  vertical field of view in degrees
- float near_plane

  near clipping plane distance
- float far_plane

  far clipping plane distance
- float aspect_ratio

  aspect ratio (width/height)
- glm::vec3 target

  world space look-at target
- Matrix4x4 projection_matrix

  cached projection matrix
- Matrix4x4 view_matrix

  cached view matrix
- Matrix4x4 vp_matrix

  cached view-projection matrix
- bool projDirty = true

  flag to indicate if projection matrix is dirty
- bool viewDirty = true

  flag to indicate if view matrix is dirty
- bool vpDirty = true

  flag to indicate if view-projection matrix is dirty

Static Private Attributes

- static const char ∗ CAMERA_TAG = "Camera"

  Tag for logging camera-related messages.

Additional Inherited Members

Protected Attributes inherited from Ragot::Component

- std::vector< std::shared_ptr< Component > > components

  Collection of components managed by this Component instance.

Protected Attributes inherited from Ragot::Node

- std::vector< std::shared_ptr< Node > > children

  List of child nodes.
- Node ∗ parent = nullptr

  Pointer to the parent node.

Protected Attributes inherited from Ragot::Transform

- vec3 position

  The position of the object in 3D space.
- vec3 rotation

  The rotation of the object in degrees around each axis (x, y, z).
- vec3 scale

  The scale of the object in 3D space, default is (1, 1, 1).
- bool dirty = true

  Flag indicating whether the transformation matrix needs to be recalculated.

6.2.1 Detailed Description

Represents a camera in a 3D space, managing its properties and transformations.

The Camera class provides functionality to set and get camera properties such as field of view, near and far planes, aspect ratio, and target position. It also computes projection, view, and combined view-projection matrices, and provides methods for projecting world coordinates to normalized device coordinates (NDC).

6.2.2 Member Typedef Documentation

6.2.2.1 Matrix4x4

```
using Ragot::Camera::Matrix4x4 = glm::mat4
```

6.2.3 Constructor & Destructor Documentation

6.2.3.1 Camera() [1/2]

```
Ragot::Camera::Camera ()  [delete]
```

Delete default constructor to prevent instantiation without parameters.

6.2.3.2 Camera() [2/2]

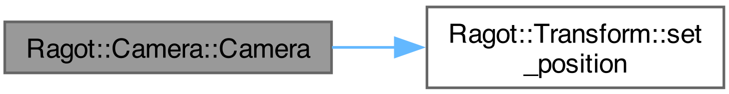
```
Ragot::Camera::Camera (
             float aspect_ratio = 1.f,
             float near_plane = 1.f,
             float far_plane = 100.f,
             float fov_deg = 60.f)  [inline]
```

Constructs a Camera object with specified parameters.

Parameters

aspect_ratio	Aspect ratio of the camera (default is 1.0).
near_plane	Distance to the near clipping plane (default is 1.0).
far_plane	Distance to the far clipping plane (default is 100.0).
fov_deg	Vertical field of view in degrees (default is 60.0).

Here is the call graph for this function:



6.2.3.3 ∼Camera()

```
Ragot::Camera::∼Camera ()  [default]
```

Default destructor for Camera class.

6.2.4 Member Function Documentation

6.2.4.1 calculate_normal()

```
vertex_t Ragot::Camera::calculate_normal (
             const vertex_t & v1,
             const vertex_t & v2,
             const vertex_t & v3)
```

Calculate the normal vector of a face defined by three vertices.

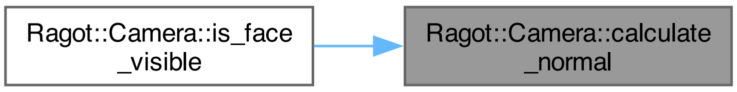
Parameters

v1	First vertex of the face.
v2	Second vertex of the face.
v3	Third vertex of the face.

Returns

vertex_t

Here is the caller graph for this function:


Ragot::Camera::is_face_visible
Ragot::Camera::calculate_normal

6.2.4.2 get_aspect_ratio()

```
float Ragot::Camera::get_aspect_ratio () const  [inline]
```

Returns the camera's aspect ratio (width/height).

Returns

float The aspect ratio of the camera.

6.2.4.3 get_far_plane()

```
float Ragot::Camera::get_far_plane () const  [inline]
```

Returns the camera's far clipping plane distance.

Returns

float The distance to the far clipping plane.

6.2.4.4 get_fov()

```
float Ragot::Camera::get_fov () const  [inline]
```

Returns the camera's field of view in degrees.

Returns

float The field of view in degrees.

6.2.4.5 get_location()

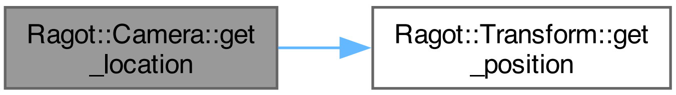
```
glm::vec3 Ragot::Camera::get_location () const  [inline]
```

Returns the camera's position in world space.

Returns

glm::vec3 The position of the camera.

Here is the call graph for this function:



6.2.4.6 get_near_plane()

```
float Ragot::Camera::get_near_plane () const  [inline]
```

Returns the camera's near clipping plane distance.

Returns

float The distance to the near clipping plane.

6.2.4.7 get_projection_matrix()

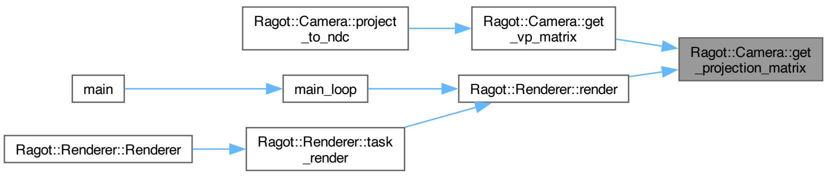
```
const Matrix4x4 & Ragot::Camera::get_projection_matrix () const  [inline]
```

Get the projection matrix object.

Returns

const Matrix4x4&

Here is the caller graph for this function:


Ragot::Camera::project _to_ndc
Ragot::Camera::get _vp_matrix
Ragot::Camera::get _projection_matrix
main
main_loop
Ragot::Renderer::render
Ragot::Renderer::Renderer
Ragot::Renderer::task _render

6.2.4.8 get_right_direction()

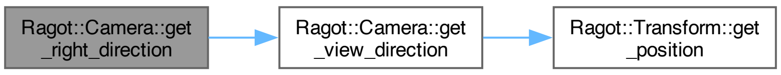
```
glm::vec3 Ragot::Camera::get_right_direction () const  [inline]
```

Get the right direction object.

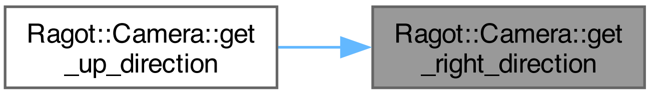
Returns

glm::vec3

Here is the call graph for this function:


Ragot::Camera::get_right_direction
Ragot::Camera::get_view_direction
Ragot::Transform::get_position

Here is the caller graph for this function:


Ragot::Camera::get_up_direction
Ragot::Camera::get_right_direction

6.2.4.9 get_target()

```
glm::vec3 Ragot::Camera::get_target () const  [inline]
```

Returns the camera's target position in world space.

Returns

glm::vec3 The target position of the camera.

6.2.4.10 get_up_direction()

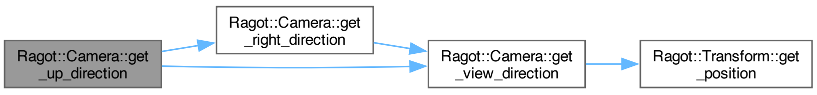
```
glm::vec3 Ragot::Camera::get_up_direction () const  [inline]
```

Get the up direction object.

Returns

glm::vec3

Here is the call graph for this function:


Ragot::Camera::get_up_direction
Ragot::Camera::get_right_direction
Ragot::Camera::get_view_direction
Ragot::Transform::get_position

6.2.4.11 get_view_direction()

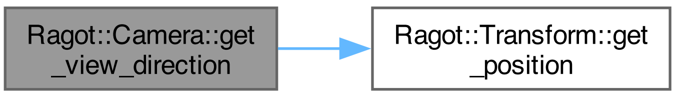
```
glm::vec3 Ragot::Camera::get_view_direction () const  [inline]
```

Get the view direction object.

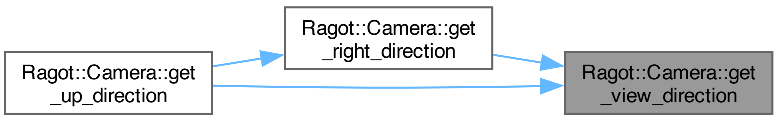
Returns

glm::vec3

Here is the call graph for this function:


Ragot::Camera::get_view_direction
Ragot::Transform::get_position

Here is the caller graph for this function:


Ragot::Camera::get_up_direction
Ragot::Camera::get_right_direction
Ragot::Camera::get_view_direction

6.2.4.12 get_view_matrix()

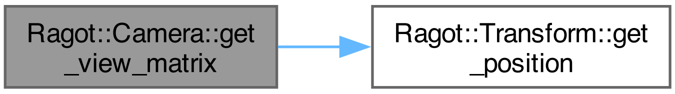
```
const Matrix4x4 & Ragot::Camera::get_view_matrix () const  [inline]
```

Get the view matrix object.

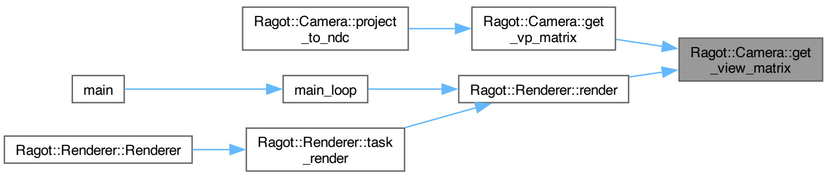
Returns

const Matrix4x4&

Here is the call graph for this function:


Ragot::Camera::get_view_matrix
Ragot::Transform::get_position

Here is the caller graph for this function:


Ragot::Camera::project_to_ndc
Ragot::Camera::get_vp_matrix
Ragot::Camera::get_view_matrix
main
main_loop
Ragot::Renderer::render
Ragot::Renderer::Renderer
Ragot::Renderer::task_render

6.2.4.13 get_vp_matrix()

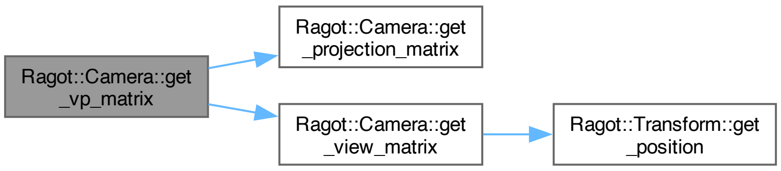
```
const Matrix4x4 & Ragot::Camera::get_vp_matrix () const  [inline]
```

Get the vp matrix object.

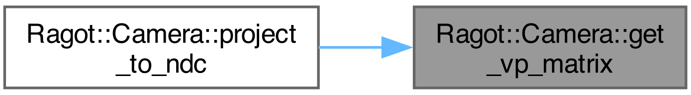
Returns

const Matrix4x4&

Here is the call graph for this function:


Ragot::Camera::get_vp_matrix
Ragot::Camera::get_projection_matrix
Ragot::Camera::get_view_matrix
Ragot::Transform::get_position

Here is the caller graph for this function:



6.2.4.14 is_dirty()

```
bool Ragot::Camera::is_dirty () const  [inline]
```

Returns the camera's position in world space.

Returns

glm::vec3 The position of the camera.

6.2.4.15 is_face_visible()

```
bool Ragot::Camera::is_face_visible (
             const vertex_t & v1,
             const vertex_t & v2,
             const vertex_t & v3)
```

Check if a face defined by three vertices is visible from the camera's perspective.

A face is considered visible if its normal vector points towards the camera.

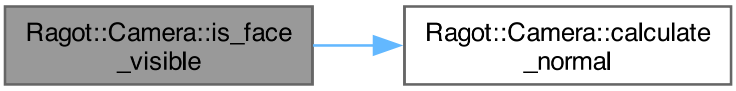
Parameters

v1	First vertex of the face.
v2	Second vertex of the face.
v3	Third vertex of the face.

Returns

true if the face is visible, false otherwise.

Here is the call graph for this function:


Ragot::Camera::is_face_visible
Ragot::Camera::calculate_normal

6.2.4.16 log_camera_info()

```
void Ragot::Camera::log_camera_info () const
```

Logs the camera's properties for debugging purposes.

This method logs the camera's position, target, field of view, near and far planes, and aspect ratio.

6.2.4.17 project_to_ndc()

```
glm::vec3 Ragot::Camera::project_to_ndc (
             const glm::vec4 & worldPos) const  [inline]
```

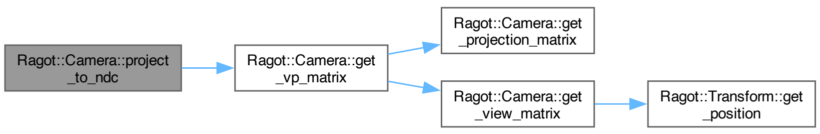
Parameters

worldPos	

Returns

glm::vec3

Here is the call graph for this function:


Ragot::Camera::project_to_ndc
Ragot::Camera::get_vp_matrix
Ragot::Camera::get_projection_matrix
Ragot::Camera::get_view_matrix
Ragot::Transform::get_position

6.2.4.18 set_aspect_ratio()

```
void Ragot::Camera::set_aspect_ratio (
             float ar)  [inline]
```

Set the aspect ratio of the camera.

Parameters

ar	Aspect ratio (width/height).

6.2.4.19 set_far_plane()

```
void Ragot::Camera::set_far_plane (
             float fp)  [inline]
```

Set the far clipping plane distance.

Parameters

fp	Distance to the far clipping plane.

6.2.4.20 set_fov()

```
void Ragot::Camera::set_fov (
             float deg)  [inline]
```

Set the fov object.

Parameters

deg	Field of view in degrees.

6.2.4.21 set_location()

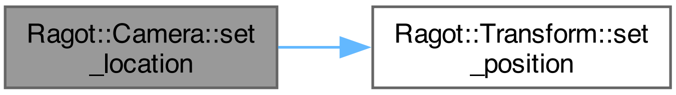
```
void Ragot::Camera::set_location (
             const glm::vec3 & p)  [inline]
```

Set the camera's position in world space.

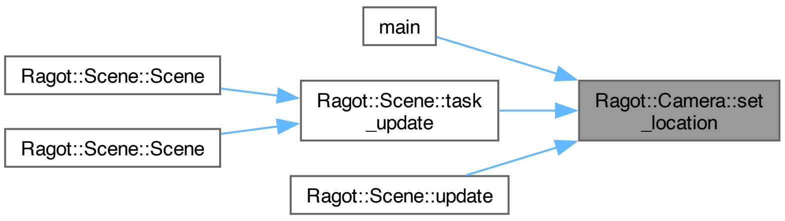
Parameters

p	Position of the camera.

Here is the call graph for this function:



Here is the caller graph for this function:



6.2.4.22 set_near_plane()

```
void Ragot::Camera::set_near_plane (
                float np)   [inline]
```

Set the near clipping plane distance.

Parameters

np	Distance to the near clipping plane.

6.2.4.23 set_target()

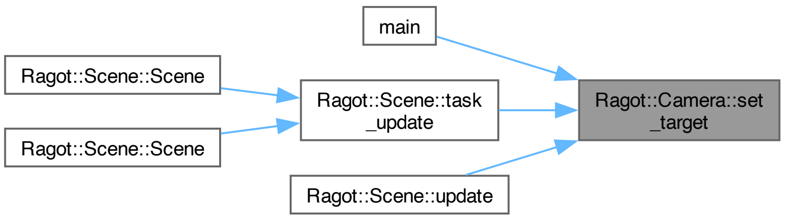
```
void Ragot::Camera::set_target (
                const glm::vec3 & t)   [inline]
```

Set the camera's target position in world space.

Parameters

t	Target position of the camera.

Here is the caller graph for this function:



6.2.5 Member Data Documentation

6.2.5.1 aspect_ratio

```
float Ragot::Camera::aspect_ratio  [private]
```

aspect ratio (width/height)

6.2.5.2 CAMERA_TAG

```
const char * Ragot::Camera::CAMERA_TAG = "Camera"  [static], [private]
```

Tag for logging camera-related messages.

6.2.5.3 far_plane

```
float Ragot::Camera::far_plane  [private]
```

far clipping plane distance

6.2.5.4 fov

```
float Ragot::Camera::fov  [private]
```

vertical field of view in degrees

6.2.5.5 near_plane

```
float Ragot::Camera::near_plane  [private]
```

near clipping plane distance

6.2.5.6 projDirty

```
bool Ragot::Camera::projDirty = true  [mutable], [private]
```

flag to indicate if projection matrix is dirty

6.2.5.7 projection_matrix

```
Matrix4x4 Ragot::Camera::projection_matrix  [mutable], [private]
```

cached projection matrix

6.2.5.8 target

```
glm::vec3 Ragot::Camera::target  [private]
```

world space look-at target

6.2.5.9 view_matrix

```
Matrix4x4 Ragot::Camera::view_matrix  [mutable], [private]
```

cached view matrix

6.2.5.10 viewDirty

```
bool Ragot::Camera::viewDirty = true  [mutable], [private]
```

flag to indicate if view matrix is dirty

6.2.5.11 vp_matrix

```
Matrix4x4 Ragot::Camera::vp_matrix  [mutable], [private]
```

cached view-projection matrix

6.2.5.12 vpDirty

```
bool Ragot::Camera::vpDirty = true  [mutable], [private]
```

flag to indicate if view-projection matrix is dirty

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

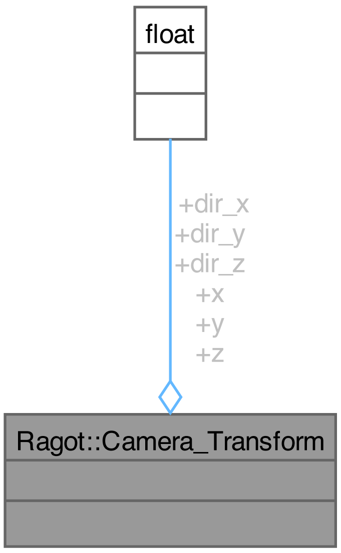
- main/Camera.hpp
- main/Camera.cpp

6.3 Ragot::Camera_Transform Struct Reference

Represents the transformation of a camera in 3D space.

```
#include <CommonTypes.hpp>
```

Collaboration diagram for Ragot::Camera_Transform:



Public Attributes

- float x
- float y
- float z
- float dir_x
- float dir_y
- float dir_z

6.3.1 Detailed Description

Represents the transformation of a camera in 3D space.

This structure holds the position and direction of the camera, allowing for transformations in a 3D environment.

6.3.2 Member Data Documentation

6.3.2.1 dir_x

```
float Ragot::Camera_Transform::dir_x
```

6.3.2.2 dir_y

```
float Ragot::Camera_Transform::dir_y
```

6.3.2.3 dir_z

```
float Ragot::Camera_Transform::dir_z
```

6.3.2.4 x

```
float Ragot::Camera_Transform::x
```

6.3.2.5 y

```
float Ragot::Camera_Transform::y
```

6.3.2.6 z

```
float Ragot::Camera_Transform::z
```

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

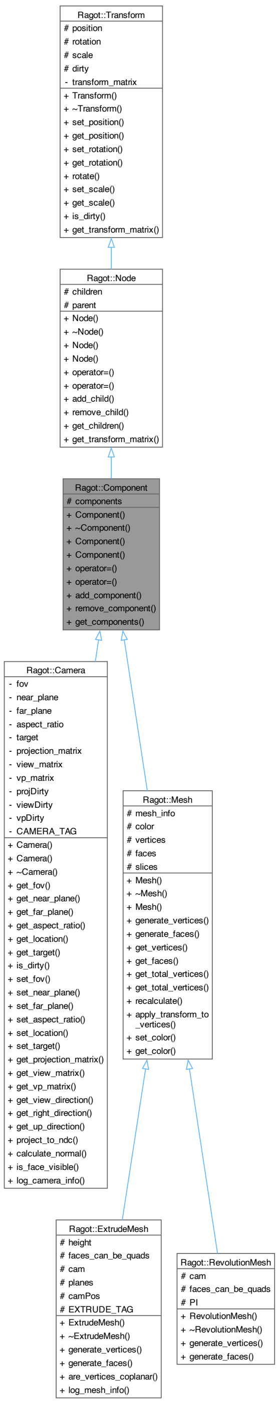
- main/CommonTypes.hpp

6.4 Ragot::Component Class Reference

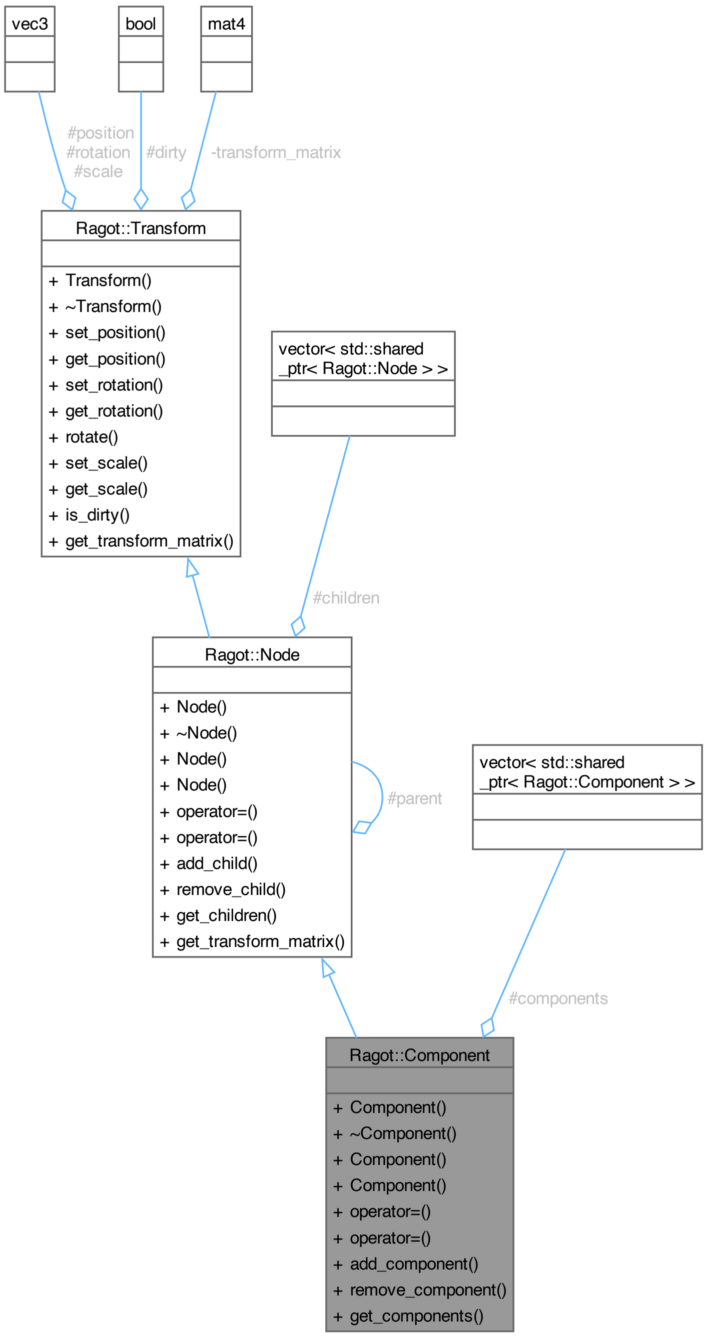
Base class for components in the Ragot engine.

```
#include <Components.hpp>
```

Inheritance diagram for Ragot::Component:



Collaboration diagram for Ragot::Component:



Public Member Functions

- Component ()=default

  Default constructor for the Component class.
- virtual ~Component ()=default

  Default virtual destructor for the Component class.
- Component (const Component &)=delete

  Deleted copy constructor for the Component class.
- Component (const Component &&)=delete

  Deleted move constructor for the Component class.
- Component & operator= (const Component &)=delete

  Deleted assignment operator for the Component class.
- Component & operator= (const Component &&)=delete

  Deleted move assignment operator for the Component class.
- void add_component (std::shared_ptr< Component > component)

  Adds a component to the collection.
- void remove_component (std::shared_ptr< Component > component)

  Removes a component from the collection.
- const std::vector< std::shared_ptr< Component > > get_components () const

  Gets the collection of components.

Public Member Functions inherited from Ragot::Node

- Node ()=default

  Default constructor for Node. Initializes an empty node with no parent and no children.
- virtual ∼Node ()=default

  Default destructor for Node. Cleans up the node and its children.
- Node (const Node &)=delete

  Deleted copy constructor for Node. Prevents copying of Node instances.
- Node (const Node &&)=delete

  Deleted move constructor for Node. Prevents moving of Node instances.
- Node & operator= (const Node &)=delete

  Deleted assignment operator for Node. Prevents assignment of Node instances.
- Node & operator= (const Node &&)=delete

  Deleted move assignment operator for Node. Prevents moving of Node instances.
- void add_child (std::shared_ptr< Node > child)

  Get the parent node.
- void remove_child (std::shared_ptr< Node > child)

  Remove a child node.
- const std::vector< std::shared_ptr< Node > > & get_children () const

  Get the parent node.
- mat4 get_transform_matrix () override

  Get the transform matrix object.

Public Member Functions inherited from Ragot::Transform

- Transform ()

  Default constructor for the Transform class.
- virtual ∼Transform ()=default

  Virtual destructor for the Transform class.
- void set_position (const vec3 &pos)

  Sets the position of the object.
- vec3 get_position () const

  Gets the current position of the object.
- void set_rotation (const vec3 &rot)

  Moves the object by a specified vector.

- vec3 get_rotation () const

  Gets the current rotation of the object.
- void rotate (const float angle, const vec3 &axis)

  Rotates the object by a specified angle around a given axis.
- void set_scale (const vec3 &scale)

  Sets the scale of the object.
- vec3 get_scale () const

  Sets the scale of the object uniformly.
- bool is_dirty () const

  Checks if the transformation matrix is dirty (needs recalculation).

Protected Attributes

- std::vector< std::shared_ptr< Component > > components

  Collection of components managed by this Component instance.

Protected Attributes inherited from Ragot::Node

- std::vector< std::shared_ptr< Node > > children

  List of child nodes.
- Node ∗ parent = nullptr

  Pointer to the parent node.

Protected Attributes inherited from Ragot::Transform

- vec3 position

  The position of the object in 3D space.
- vec3 rotation

  The rotation of the object in degrees around each axis (x, y, z).
- vec3 scale

  The scale of the object in 3D space, default is (1, 1, 1).
- bool dirty = true

  Flag indicating whether the transformation matrix needs to be recalculated.

6.4.1 Detailed Description

Base class for components in the Ragot engine.

The Component class serves as a base class for all components in the Ragot engine. It allows for the management of a collection of components, providing methods to add and remove components, and access the list of components.

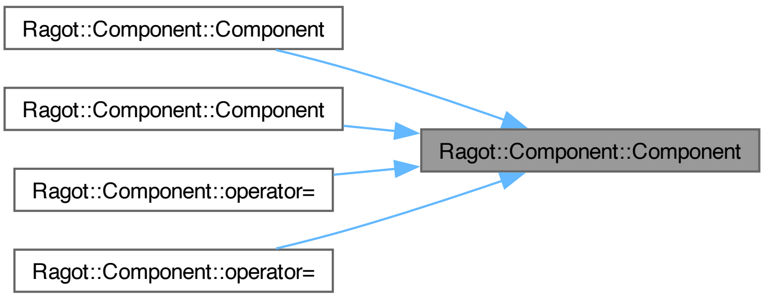
6.4.2 Constructor & Destructor Documentation

6.4.2.1 Component() `[1/3]`

```
Ragot::Component::Component ()  [default]
```

Default constructor for the Component class.

Initializes an empty component with no parent. Here is the caller graph for this function:


Ragot::Component::Component
Ragot::Component::Component
Ragot::Component::operator=
Ragot::Component::operator=
Ragot::Component::Component

6.4.2.2 ∼Component()

```
virtual Ragot::Component::∼Component ()  [virtual], [default]
```

Default virtual destructor for the Component class.

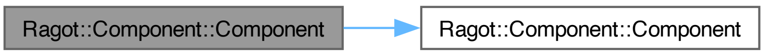
Cleans up the component and its resources.

6.4.2.3 Component() `[2/3]`

```
Ragot::Component::Component (
            const Component & )  [delete]
```

Deleted copy constructor for the Component class.

Here is the call graph for this function:

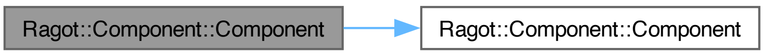


6.4.2.4 Component() [3/3]

```
Ragot::Component::Component (
            const Component && )  [delete]
```

Deleted move constructor for the Component class.

Prevents moving a Component instance. Here is the call graph for this function:



6.4.3 Member Function Documentation

6.4.3.1 add_component()

```
void Ragot::Component::add_component (
            std::shared_ptr< Component > component)  [inline]
```

Adds a component to the collection.

This method adds a shared pointer to a component to the collection and sets its parent to this Component instance.

Parameters

component	Shared pointer to the component to be added.

6.4.3.2 get_components()

```
const std::vector< std::shared_ptr< Component > > Ragot::Component::get_components () const
[inline]
```

Gets the collection of components.

This method returns a constant reference to the vector of components managed by this Component instance.

Returns

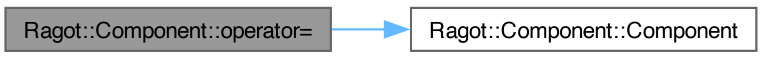
const std::vector<std::shared_ptr<Component>>& Reference to the vector of components.

6.4.3.3 operator=() [1/2]

```
Component & Ragot::Component::operator= (
             const Component && )  [delete]
```

Deleted move assignment operator for the Component class.

Prevents moving a Component instance. Here is the call graph for this function:

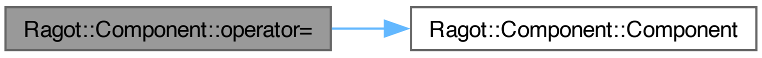


6.4.3.4 operator=() [2/2]

```
Component & Ragot::Component::operator= (
             const Component & )  [delete]
```

Deleted assignment operator for the Component class.

Prevents assignment of a Component instance. Here is the call graph for this function:



6.4.3.5 remove_component()

```
void Ragot::Component::remove_component (
             std::shared_ptr< Component > component)  [inline]
```

Removes a component from the collection.

This method removes a shared pointer to a component from the collection and sets its parent to nullptr.

Parameters

component	Shared pointer to the component to be removed.

6.4.4 Member Data Documentation

6.4.4.1 components

```
std::vector< std::shared_ptr < Component > > Ragot::Component::components  [protected]
```

Collection of components managed by this Component instance.

This vector holds shared pointers to the components that are part of this Component instance.

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

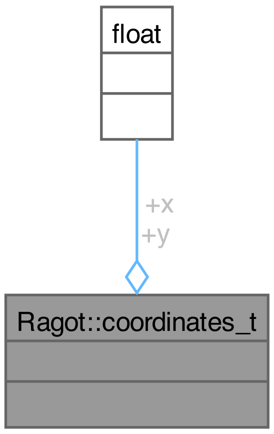
- main/Components.hpp

6.5 Ragot::coordinates_t Struct Reference

Represents 2D coordinates.

```
#include <CommonTypes.hpp>
```

Collaboration diagram for Ragot::coordinates_t:



Public Attributes

- float x
- float y

6.5.1 Detailed Description

Represents 2D coordinates.

This structure holds the x and y coordinates of a point in 2D space, typically used for mesh vertices.

6.5.2 Member Data Documentation

6.5.2.1 x

```
float Ragot::coordinates_t::x
```

6.5.2.2 y

```
float Ragot::coordinates_t::y
```

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

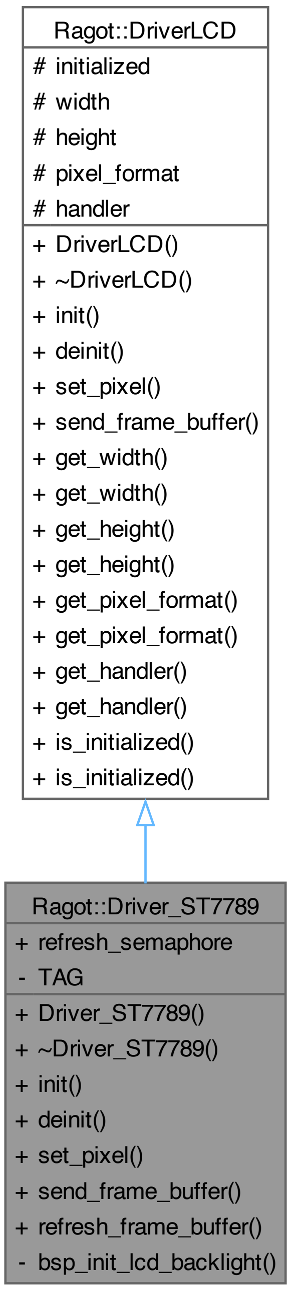
- main/CommonTypes.hpp

6.6 Ragot::Driver_ST7789 Class Reference

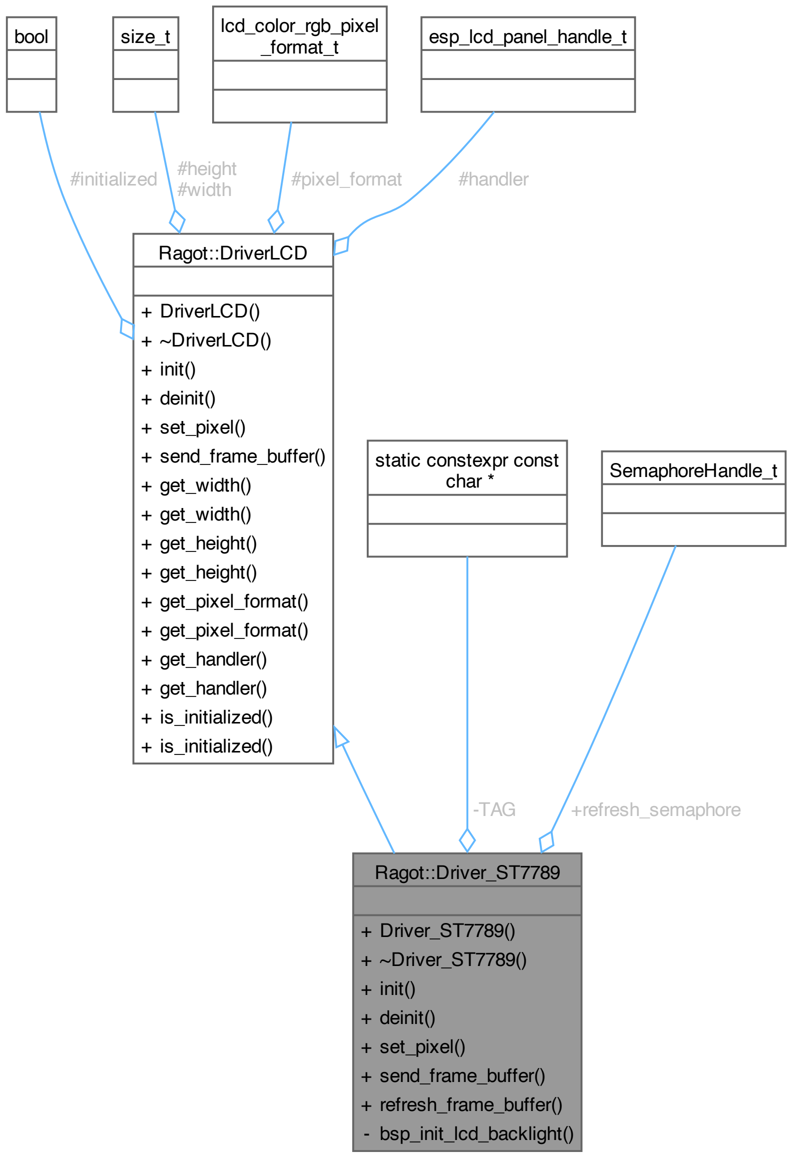
Driver for the ST7789 LCD panel.

```
#include <Driver_ST7789.hpp>
```

Inheritance diagram for Ragot::Driver_ST7789:



Collaboration diagram for Ragot::Driver_ST7789:



Public Member Functions

- Driver_ST7789 ()

    Default constructor for the Driver_ST7789 class.
- virtual ~Driver_ST7789 ()=default

    Default virtual destructor for the Driver_ST7789 class.
- esp_err_t init (gpio_num_t reset_pin, gpio_num_t bk_pin) override

    Initializes the ST7789 LCD panel driver.
- esp_err_t deinit () override

    Deinitializes the ST7789 LCD panel driver.
- esp_err_t set_pixel (uint32_t x, uint32_t y, uint32_t color) override

Sets a pixel at the specified coordinates to the given color.
- esp_err_t send_frame_buffer (const void ∗frame_buffer) override

  Sends a frame buffer to the ST7789 LCD panel.
- IRAM_ATTR bool refresh_frame_buffer (void ∗user_ctx)

  Refreshes the frame buffer for the ST7789 LCD panel.

Public Member Functions inherited from Ragot::DriverLCD

- DriverLCD ()=default

  Default constructor for the DriverLCD class.
- virtual ∼DriverLCD ()=default

  Default virtual destructor for the DriverLCD class.
- const size_t get_width () const

  Gets the width of the LCD panel.
- size_t get_width ()

  Gets the height of the LCD panel.
- const size_t get_height () const

  Gets the height of the LCD panel.
- size_t get_height ()

  Gets the height of the LCD panel.
- const lcd_color_rgb_pixel_format_t get_pixel_format () const

  Get the pixel format object.
- lcd_color_rgb_pixel_format_t get_pixel_format ()

  Get the pixel format object.
- const esp_lcd_panel_handle_t get_handler () const

  Get the handler object.
- esp_lcd_panel_handle_t get_handler ()

  Get the handler object.
- const bool is_initialized () const

  Checks if the LCD driver is initialized.
- bool is_initialized ()

  Checks if the LCD driver is initialized.

Public Attributes

- SemaphoreHandle_t refresh_semaphore

  Refresh semaphore for synchronizing frame buffer updates.

Private Member Functions

- void bsp_init_lcd_backlight (gpio_num_t bk_pin)

  Initializes the LCD backlight GPIO pin.

Static Private Attributes

- static constexpr const char ∗ TAG = "[Driver_ST7789]..."

  Tag for logging messages related to the ST7789 driver.

Additional Inherited Members

Protected Attributes inherited from Ragot::DriverLCD

- bool initialized = false

  Flag indicating if the LCD driver is initialized.
- size_t width

  Width of the LCD panel in pixels.
- size_t height

  Height of the LCD panel in pixels.
- lcd_color_rgb_pixel_format_t pixel_format

  Pixel format of the LCD panel.
- esp_lcd_panel_handle_t handler

  Handle to the LCD panel.

6.6.1 Detailed Description

Driver for the ST7789 LCD panel.

This class provides methods to initialize, deinitialize, and send frame buffers to the ST7789 LCD panel. It inherits from the DriverLCD class and implements the necessary methods for LCD operations.

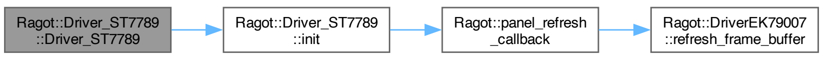
6.6.2 Constructor & Destructor Documentation

6.6.2.1 Driver_ST7789()

```
Ragot::Driver_ST7789::Driver_ST7789 ()
```

Default constructor for the Driver_ST7789 class.

Initializes the LCD driver with default values. Here is the call graph for this function:



6.6.2.2 ∼Driver_ST7789()

```
virtual Ragot::Driver_ST7789::∼Driver_ST7789 ()  [virtual], [default]
```

Default virtual destructor for the Driver_ST7789 class.

Cleans up resources used by the LCD driver.

6.6.3 Member Function Documentation

6.6.3.1 bsp_init_lcd_backlight()

```
void Ragot::Driver_ST7789::bsp_init_lcd_backlight (
            gpio_num_t bk_pin)  [private]
```

Initializes the LCD backlight GPIO pin.

This method sets up the specified GPIO pin for controlling the LCD backlight.

Parameters

bk_pin	GPIO pin for backlight control.

6.6.3.2 deinit()

```
esp_err_t Ragot::Driver_ST7789::deinit ()  [override], [virtual]
```

Deinitializes the ST7789 LCD panel driver.

This method cleans up resources used by the driver, including deleting the panel handler and freeing the refresh semaphore.

Returns

esp_err_t ESP_OK on success, or an error code on failure.

Implements Ragot::DriverLCD.

6.6.3.3 init()

```
esp_err_t Ragot::Driver_ST7789::init (
            gpio_num_t reset_pin,
            gpio_num_t bk_pin)  [override], [virtual]
```

Initializes the ST7789 LCD panel driver.

This method sets up the LCD panel with the specified reset and backlight GPIO pins. It configures the panel's pixel format, width, height, and other parameters.

Parameters

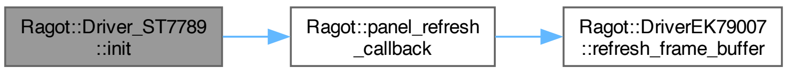
reset_pin	GPIO pin for panel reset.
bk_pin	GPIO pin for backlight control.

Returns

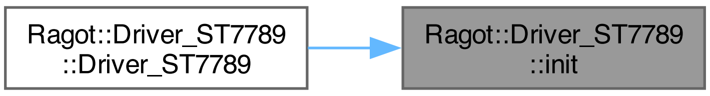
esp_err_t ESP_OK on success, or an error code on failure.

Implements Ragot::DriverLCD.

Here is the call graph for this function:


Ragot::Driver_ST7789 ::init
Ragot::panel_refresh _callback
Ragot::DriverEK79007 ::refresh_frame_buffer

Here is the caller graph for this function:



6.6.3.4 refresh_frame_buffer()

```
IRAM_ATTR bool Ragot::Driver_ST7789::refresh_frame_buffer (
            void * user_ctx)
```

Refreshes the frame buffer for the ST7789 LCD panel.

Note

This method is called from the ISR context and should be used to signal that the frame buffer has been updated.

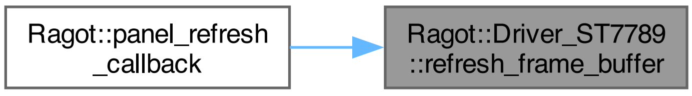
Parameters

user_ctx	

Returns

bool True if the frame buffer was refreshed successfully, false otherwise.

Here is the caller graph for this function:



6.6.3.5 send_frame_buffer()

```
esp_err_t Ragot::Driver_ST7789::send_frame_buffer (
            const void * frame_buffer)  [override], [virtual]
```

Sends a frame buffer to the ST7789 LCD panel.

This method sends the provided frame buffer to the panel for display. It waits for the refresh semaphore to be available before sending the frame buffer.

Parameters

frame_buffer	Pointer to the frame buffer data.

Returns

esp_err_t ESP_OK on success, or an error code on failure.

Implements Ragot::DriverLCD.

6.6.3.6 set_pixel()

```
esp_err_t Ragot::Driver_ST7789::set_pixel (
            uint32_t x,
            uint32_t y,
            uint32_t color) [inline], [override], [virtual]
```

Sets a pixel at the specified coordinates to the given color.

This method is not implemented for the ST7789 driver and will return ESP_FAIL.

Parameters

x	X coordinate of the pixel.
y	Y coordinate of the pixel.
color	Color value for the pixel.

Returns

esp_err_t ESP_FAIL as this method is not implemented.

Implements Ragot::DriverLCD.

6.6.4 Member Data Documentation

6.6.4.1 refresh_semaphore

```
SemaphoreHandle_t Ragot::Driver_ST7789::refresh_semaphore
```

Refresh semaphore for synchronizing frame buffer updates.

6.6.4.2 TAG

```
const char* Ragot::Driver_ST7789::TAG = "[Driver_ST7789]..." [static], [constexpr], [private]
```

Tag for logging messages related to the ST7789 driver.

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

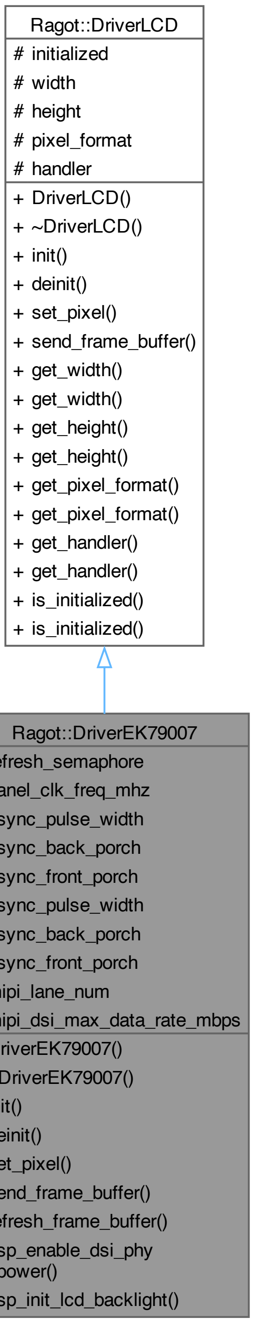
- main/Driver_ST7789.hpp
- main/Driver_ST7789.cpp

6.7 Ragot::DriverEK79007 Class Reference

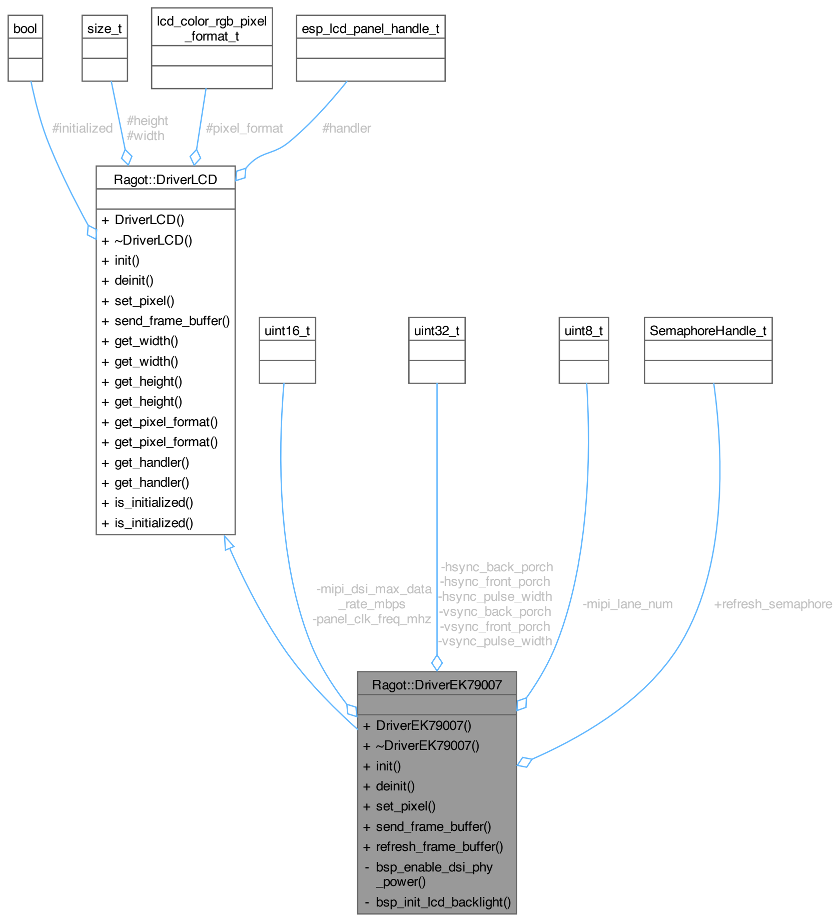
Driver for the EK79007 LCD panel.

```
#include <driver_ek79007.hpp>
```

Inheritance diagram for Ragot::DriverEK79007:



Collaboration diagram for Ragot::DriverEK79007:



Public Member Functions

- DriverEK79007 ()

  Default constructor for the DriverEK79007 class.
- ∼DriverEK79007 () override

  Destructor for the DriverEK79007 class.
- esp_err_t init (gpio_num_t reset_pin, gpio_num_t bk_pin) override

  Initializes the EK79007 LCD panel driver.
- esp_err_t deinit () override

  Deinitializes the EK79007 LCD panel driver.
- esp_err_t set_pixel (uint32_t x, uint32_t y, uint32_t color) override

  Sets a pixel at the specified coordinates to the given color.
- esp_err_t send_frame_buffer (const void ∗frame_buffer) override

  Sends a frame buffer to the EK79007 LCD panel.
- IRAM_ATTR bool refresh_frame_buffer (esp_lcd_panel_handle_t panel, esp_lcd_dpi_panel_event_data_←t ∗edata, void ∗user_ctx)

  Refreshes the frame buffer for the EK79007 LCD panel.

Public Member Functions inherited from Ragot::DriverLCD

- DriverLCD ()=default

  Default constructor for the DriverLCD class.
- virtual ∼DriverLCD ()=default

  Default virtual destructor for the DriverLCD class.
- const size_t get_width () const

  Gets the width of the LCD panel.
- size_t get_width ()

  Gets the height of the LCD panel.
- const size_t get_height () const

  Gets the height of the LCD panel.
- size_t get_height ()

  Gets the height of the LCD panel.
- const lcd_color_rgb_pixel_format_t get_pixel_format () const

  Get the pixel format object.
- lcd_color_rgb_pixel_format_t get_pixel_format ()

  Get the pixel format object.
- const esp_lcd_panel_handle_t get_handler () const

  Get the handler object.
- esp_lcd_panel_handle_t get_handler ()

  Get the handler object.
- const bool is_initialized () const

  Checks if the LCD driver is initialized.
- bool is_initialized ()

  Checks if the LCD driver is initialized.

Public Attributes

- SemaphoreHandle_t refresh_semaphore

  Refresh semaphore for synchronizing frame buffer updates.

Private Member Functions

- void bsp_enable_dsi_phy_power ()

  Enables the MIPI DSI PHY power.
- void bsp_init_lcd_backlight (gpio_num_t bk_pin)

  Initializes the LCD backlight.

Private Attributes

- uint16_t panel_clk_freq_mhz

  Horizontal pixel clock frequency in MHz.
- uint32_t hsync_pulse_width

  Horizontal sync width, in pixel clock.
- uint32_t hsync_back_porch

  Horizontal back porch, number of pixel clock between hsync and start of line active data.
- uint32_t hsync_front_porch

  Horizontal front porch, number of pixel clock between the end of active data and the next hsync.

- uint32_t vsync_pulse_width

  Vertical sync width, in number of lines.
- uint32_t vsync_back_porch

  Vertical back porch, number of invalid lines between vsync and start of frame.
- uint32_t vsync_front_porch

  Vertical front porch, number of invalid lines between the end of frame and the next vsync.
- uint8_t mipi_lane_num

  Number of MIPI DSI lanes used for the panel.
- uint16_t mipi_dsi_max_data_rate_mbps

  Maximum data rate of MIPI DSI in Mbps.

Additional Inherited Members

Protected Attributes inherited from Ragot::DriverLCD

- bool initialized = false

  Flag indicating if the LCD driver is initialized.
- size_t width

  Width of the LCD panel in pixels.
- size_t height

  Height of the LCD panel in pixels.
- lcd_color_rgb_pixel_format_t pixel_format

  Pixel format of the LCD panel.
- esp_lcd_panel_handle_t handler

  Handle to the LCD panel.

6.7.1 Detailed Description

Driver for the EK79007 LCD panel.

This class provides methods to initialize, deinitialize, and send frame buffers to the EK79007 LCD panel. It inherits from the DriverLCD class and implements the necessary methods for LCD operations.

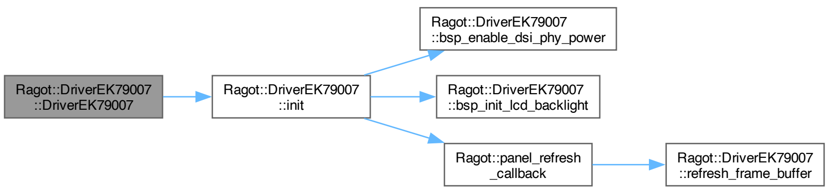
6.7.2 Constructor & Destructor Documentation

6.7.2.1 DriverEK79007()

```
Ragot::DriverEK79007::DriverEK79007 ()
```

Default constructor for the DriverEK79007 class.

Here is the call graph for this function:

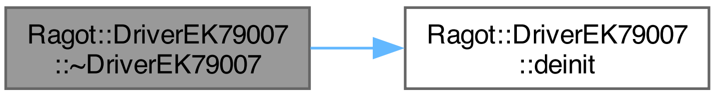

Ragot::DriverEK79007 ::DriverEK79007
Ragot::DriverEK79007 ::init
Ragot::DriverEK79007 ::bsp_enable_dsi_phy_power
Ragot::DriverEK79007 ::bsp_init_lcd_backlight
Ragot::panel_refresh _callback
Ragot::DriverEK79007 ::refresh_frame_buffer

6.7.2.2 ∼DriverEK79007()

```
Ragot::DriverEK79007::∼DriverEK79007 ()  [override]
```

Destructor for the DriverEK79007 class.

Cleans up resources used by the driver. Here is the call graph for this function:



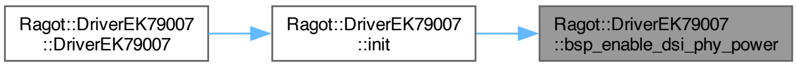
6.7.3 Member Function Documentation

6.7.3.1 bsp_enable_dsi_phy_power()

```
void Ragot::DriverEK79007::bsp_enable_dsi_phy_power ()  [private]
```

Enables the MIPI DSI PHY power.

This method powers on the MIPI DSI PHY by acquiring the appropriate LDO channel. Here is the caller graph for this function:



6.7.3.2 bsp_init_lcd_backlight()

```
void Ragot::DriverEK79007::bsp_init_lcd_backlight (
             gpio_num_t bk_pin)  [private]
```

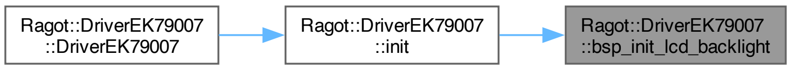
Initializes the LCD backlight.

This method configures the GPIO pin for the backlight and sets it to high to turn on the backlight.

Parameters

bk_pin	GPIO pin number for the backlight control.

Here is the caller graph for this function:


Ragot::DriverEK79007 ::DriverEK79007
Ragot::DriverEK79007 ::init
Ragot::DriverEK79007 ::bsp_init_lcd_backlight

6.7.3.3 deinit()

```
esp_err_t Ragot::DriverEK79007::deinit ()  [override], [virtual]
```

Deinitializes the EK79007 LCD panel driver.

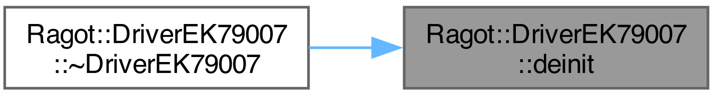
This method cleans up resources used by the driver, including deleting the panel handler and freeing the refresh semaphore.

Returns

esp_err_t ESP_OK on success, or an error code on failure.

Implements Ragot::DriverLCD.

Here is the caller graph for this function:


Ragot::DriverEK79007 ::~DriverEK79007
Ragot::DriverEK79007 ::deinit

6.7.3.4 init()

```
esp_err_t Ragot::DriverEK79007::init (
            gpio_num_t reset_pin,
            gpio_num_t bk_pin)  [override], [virtual]
```

Initializes the EK79007 LCD panel driver.

This method sets up the panel with the specified reset and backlight GPIO pins, configures the panel parameters, and registers the refresh callback.

Parameters

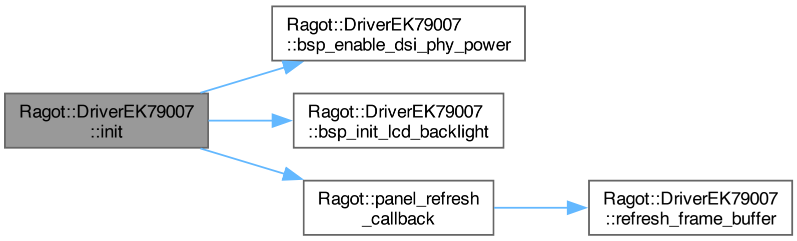
reset_pin	GPIO pin for panel reset.
bk_pin	GPIO pin for backlight control.

Returns

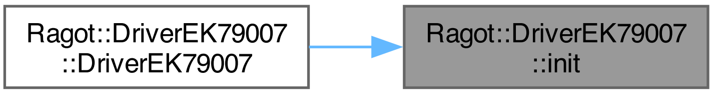
esp_err_t ESP_OK on success, or an error code on failure.

Implements Ragot::DriverLCD.

Here is the call graph for this function:



Here is the caller graph for this function:



6.7.3.5 refresh_frame_buffer()

```
IRAM_ATTR bool Ragot::DriverEK79007::refresh_frame_buffer (
             esp_lcd_panel_handle_t panel,
             esp_lcd_dpi_panel_event_data_t * edata,
             void * user_ctx)
```

Refreshes the frame buffer for the EK79007 LCD panel.

This method is called when the panel refresh is done. It handles the actual drawing of the frame buffer to the panel and releases the refresh semaphore.

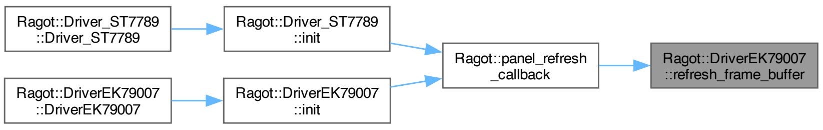
Parameters

panel	Pointer to the LCD panel handle.
edata	Pointer to the event data for the DPI panel.
user_ctx	User context pointer, which is this driver instance.

Returns

true if successful, false otherwise.

Here is the caller graph for this function:


Ragot::Driver_ST7789 ::Driver_ST7789
Ragot::Driver_ST7789 ::init
Ragot::DriverEK79007 ::DriverEK79007
Ragot::DriverEK79007 ::init
Ragot::panel_refresh _callback
Ragot::DriverEK79007 ::refresh_frame_buffer

6.7.3.6 send_frame_buffer()

```
esp_err_t Ragot::DriverEK79007::send_frame_buffer (
            const void * frame_buffer)  [override], [virtual]
```

Sends a frame buffer to the EK79007 LCD panel.

This method sends the provided frame buffer to the panel for display. It waits for the refresh semaphore to be available before sending the frame buffer.

Parameters

frame_buffer	Pointer to the frame buffer data.

Returns

esp_err_t ESP_OK on success, or an error code on failure.

Implements Ragot::DriverLCD.

6.7.3.7 set_pixel()

```
esp_err_t Ragot::DriverEK79007::set_pixel (
            uint32_t x,
            uint32_t y,
            uint32_t color)  [inline], [override], [virtual]
```

Sets a pixel at the specified coordinates to the given color.

This method is not implemented for the EK79007 driver and will return ESP_FAIL.

Parameters

x	X coordinate of the pixel.
y	Y coordinate of the pixel.
color	Color value to set the pixel to.

Returns

esp_err_t ESP_FAIL as this method is not implemented.

Implements Ragot::DriverLCD.

6.7.4 Member Data Documentation

6.7.4.1 hsync_back_porch

```
uint32_t Ragot::DriverEK79007::hsync_back_porch  [private]
```

Horizontal back porch, number of pixel clock between hsync and start of line active data.

6.7.4.2 hsync_front_porch

```
uint32_t Ragot::DriverEK79007::hsync_front_porch  [private]
```

Horizontal front porch, number of pixel clock between the end of active data and the next hsync.

6.7.4.3 hsync_pulse_width

```
uint32_t Ragot::DriverEK79007::hsync_pulse_width  [private]
```

Horizontal sync width, in pixel clock.

6.7.4.4 mipi_dsi_max_data_rate_mbps

```
uint16_t Ragot::DriverEK79007::mipi_dsi_max_data_rate_mbps  [private]
```

Maximum data rate of MIPI DSI in Mbps.

6.7.4.5 mipi_lane_num

```
uint8_t Ragot::DriverEK79007::mipi_lane_num  [private]
```

Number of MIPI DSI lanes used for the panel.

6.7.4.6 panel_clk_freq_mhz

```
uint16_t Ragot::DriverEK79007::panel_clk_freq_mhz [private]
```

Horizontal pixel clock frequency in MHz.

6.7.4.7 refresh_semaphore

```
SemaphoreHandle_t Ragot::DriverEK79007::refresh_semaphore
```

Refresh semaphore for synchronizing frame buffer updates.

This semaphore is used to ensure that the frame buffer is updated only when it is safe to do so.

6.7.4.8 vsync_back_porch

```
uint32_t Ragot::DriverEK79007::vsync_back_porch [private]
```

Vertical back porch, number of invalid lines between vsync and start of frame.

6.7.4.9 vsync_front_porch

```
uint32_t Ragot::DriverEK79007::vsync_front_porch [private]
```

Vertical front porch, number of invalid lines between the end of frame and the next vsync.

6.7.4.10 vsync_pulse_width

```
uint32_t Ragot::DriverEK79007::vsync_pulse_width [private]
```

Vertical sync width, in number of lines.

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

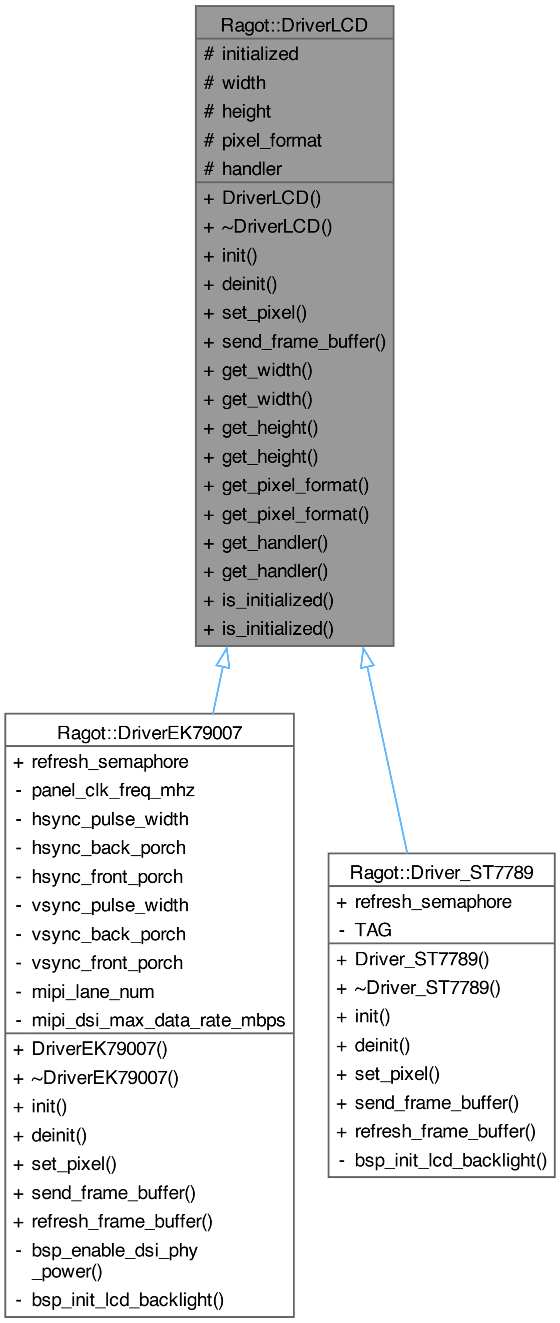
- main/driver_ek79007.hpp
- main/driver_ek79007.cpp

6.8 Ragot::DriverLCD Class Reference

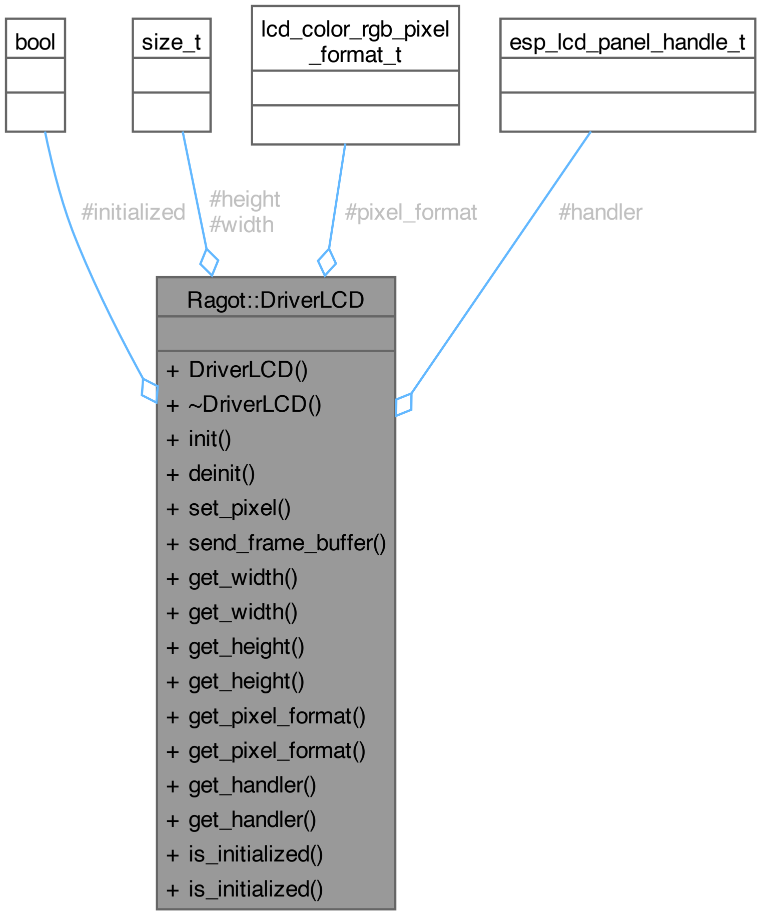
Base class for LCD drivers.

```
#include <driver_lcd.hpp>
```

Inheritance diagram for Ragot::DriverLCD:



Collaboration diagram for Ragot::DriverLCD:



Public Member Functions

- DriverLCD ()=default

  Default constructor for the DriverLCD class.
- virtual ∼DriverLCD ()=default

  Default virtual destructor for the DriverLCD class.
- virtual esp_err_t init (gpio_num_t reset_pin, gpio_num_t bk_pin)=0

  Initializes the LCD driver with the specified reset and backlight GPIO pins.
- virtual esp_err_t deinit ()=0

  Deinitializes the LCD driver.
- virtual esp_err_t set_pixel (uint32_t x, uint32_t y, uint32_t color)=0

  Sets a pixel at the specified coordinates to the given color.
- virtual esp_err_t send_frame_buffer (const void ∗frame_buffer)=0

  Sends a frame buffer to the LCD panel.
- const size_t get_width () const

  Gets the width of the LCD panel.
- size_t get_width ()

Gets the height of the LCD panel.

- const size_t get_height () const

  Gets the height of the LCD panel.
- size_t get_height ()

  Gets the height of the LCD panel.
- const lcd_color_rgb_pixel_format_t get_pixel_format () const

  Get the pixel format object.
- lcd_color_rgb_pixel_format_t get_pixel_format ()

  Get the pixel format object.
- const esp_lcd_panel_handle_t get_handler () const

  Get the handler object.
- esp_lcd_panel_handle_t get_handler ()

  Get the handler object.
- const bool is_initialized () const

  Checks if the LCD driver is initialized.
- bool is_initialized ()

  Checks if the LCD driver is initialized.

Protected Attributes

- bool initialized = false

  Flag indicating if the LCD driver is initialized.
- size_t width

  Width of the LCD panel in pixels.
- size_t height

  Height of the LCD panel in pixels.
- lcd_color_rgb_pixel_format_t pixel_format

  Pixel format of the LCD panel.
- esp_lcd_panel_handle_t handler

  Handle to the LCD panel.

6.8.1 Detailed Description

Base class for LCD drivers.

This class provides an interface for LCD drivers, including methods for initialization, deinitialization, setting pixels, and sending frame buffers. It also provides access to the LCD panel's width, height, pixel format, and handler.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 DriverLCD()

```
Ragot::DriverLCD::DriverLCD ()  [default]
```

Default constructor for the DriverLCD class.

Initializes the LCD driver with default values.

6.8.2.2 ∼DriverLCD()

```
virtual Ragot::DriverLCD::∼DriverLCD ()  [virtual], [default]
```

Default virtual destructor for the DriverLCD class.

Cleans up resources used by the LCD driver.

6.8.3 Member Function Documentation

6.8.3.1 deinit()

```
virtual esp_err_t Ragot::DriverLCD::deinit ()  [pure virtual]
```

Deinitializes the LCD driver.

This method should be implemented by derived classes to clean up resources used by the LCD panel.

Returns

esp_err_t ESP_OK on success, or an error code on failure.

Implemented in Ragot::Driver_ST7789, and Ragot::DriverEK79007.

6.8.3.2 get_handler() [1/2]

```
esp_lcd_panel_handle_t Ragot::DriverLCD::get_handler ()  [inline]
```

Get the handler object.

Returns

esp_lcd_panel_handle_t

6.8.3.3 get_handler() [2/2]

```
const esp_lcd_panel_handle_t Ragot::DriverLCD::get_handler () const  [inline]
```

Get the handler object.

Returns

const esp_lcd_panel_handle_t

6.8.3.4 get_height() [1/2]

```
size_t Ragot::DriverLCD::get_height ()  [inline]
```

Gets the height of the LCD panel.

Returns

size_t Height of the LCD panel in pixels.

6.8.3.5 get_height() [2/2]

```
const size_t Ragot::DriverLCD::get_height () const  [inline]
```

Gets the height of the LCD panel.

Returns

size_t Height of the LCD panel in pixels.

6.8.3.6 get_pixel_format() [1/2]

```
lcd_color_rgb_pixel_format_t Ragot::DriverLCD::get_pixel_format ()  [inline]
```

Get the pixel format object.

Returns

lcd_color_rgb_pixel_format_t

6.8.3.7 get_pixel_format() [2/2]

```
const lcd_color_rgb_pixel_format_t Ragot::DriverLCD::get_pixel_format () const  [inline]
```

Get the pixel format object.

Returns

const lcd_color_rgb_pixel_format_t

6.8.3.8 get_width() [1/2]

```
size_t Ragot::DriverLCD::get_width ()  [inline]
```

Gets the height of the LCD panel.

Returns

size_t Height of the LCD panel in pixels.

6.8.3.9 get_width() [2/2]

```
const size_t Ragot::DriverLCD::get_width () const  [inline]
```

Gets the width of the LCD panel.

Returns

size_t Width of the LCD panel in pixels.

6.8.3.10 init()

```
virtual esp_err_t Ragot::DriverLCD::init (
            gpio_num_t reset_pin,
            gpio_num_t bk_pin)  [pure virtual]
```

Initializes the LCD driver with the specified reset and backlight GPIO pins.

This method should be implemented by derived classes to set up the LCD panel.

Parameters

reset_pin	GPIO pin for panel reset.
bk_pin	GPIO pin for backlight control.

Returns

esp_err_t ESP_OK on success, or an error code on failure.

Implemented in Ragot::Driver_ST7789, and Ragot::DriverEK79007.

6.8.3.11 is_initialized() [1/2]

```
bool Ragot::DriverLCD::is_initialized ()  [inline]
```

Checks if the LCD driver is initialized.

Returns

true

false

6.8.3.12 is_initialized() [2/2]

```
const bool Ragot::DriverLCD::is_initialized () const  [inline]
```

Checks if the LCD driver is initialized.

Returns

true

false

6.8.3.13 send_frame_buffer()

```
virtual esp_err_t Ragot::DriverLCD::send_frame_buffer (
            const void * frame_buffer)  [pure virtual]
```

Sends a frame buffer to the LCD panel.

This method should be implemented by derived classes to send the provided frame buffer to the panel for display.

Parameters

frame_buffer	Pointer to the frame buffer data.

Returns

esp_err_t ESP_OK on success, or an error code on failure.

Implemented in Ragot::Driver_ST7789, and Ragot::DriverEK79007.

6.8.3.14 set_pixel()

```
virtual esp_err_t Ragot::DriverLCD::set_pixel (
            uint32_t x,
            uint32_t y,
            uint32_t color)  [pure virtual]
```

Sets a pixel at the specified coordinates to the given color.

This method should be implemented by derived classes to set a pixel on the LCD panel.

Parameters

x	X coordinate of the pixel.
y	Y coordinate of the pixel.
color	Color value for the pixel.

Returns

esp_err_t ESP_OK on success, or an error code on failure.

Implemented in Ragot::Driver_ST7789, and Ragot::DriverEK79007.

6.8.4 Member Data Documentation

6.8.4.1 handler

```
esp_lcd_panel_handle_t Ragot::DriverLCD::handler  [protected]
```

Handle to the LCD panel.

6.8.4.2 height

```
size_t Ragot::DriverLCD::height  [protected]
```

Height of the LCD panel in pixels.

6.8.4.3 initialized

```
bool Ragot::DriverLCD::initialized = false  [protected]
```

Flag indicating if the LCD driver is initialized.

6.8.4.4 pixel_format

```
lcd_color_rgb_pixel_format_t Ragot::DriverLCD::pixel_format  [protected]
```

Pixel format of the LCD panel.

6.8.4.5 width

```
size_t Ragot::DriverLCD::width  [protected]
```

Width of the LCD panel in pixels.

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

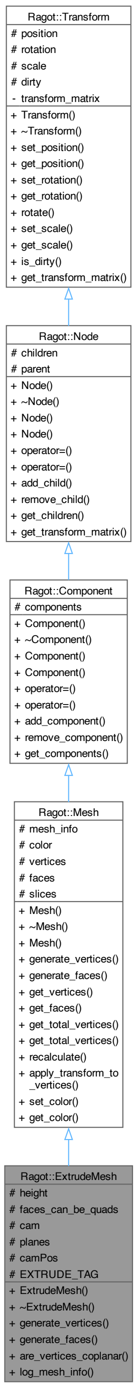
- main/driver_lcd.hpp

6.9 Ragot::ExtrudeMesh Class Reference

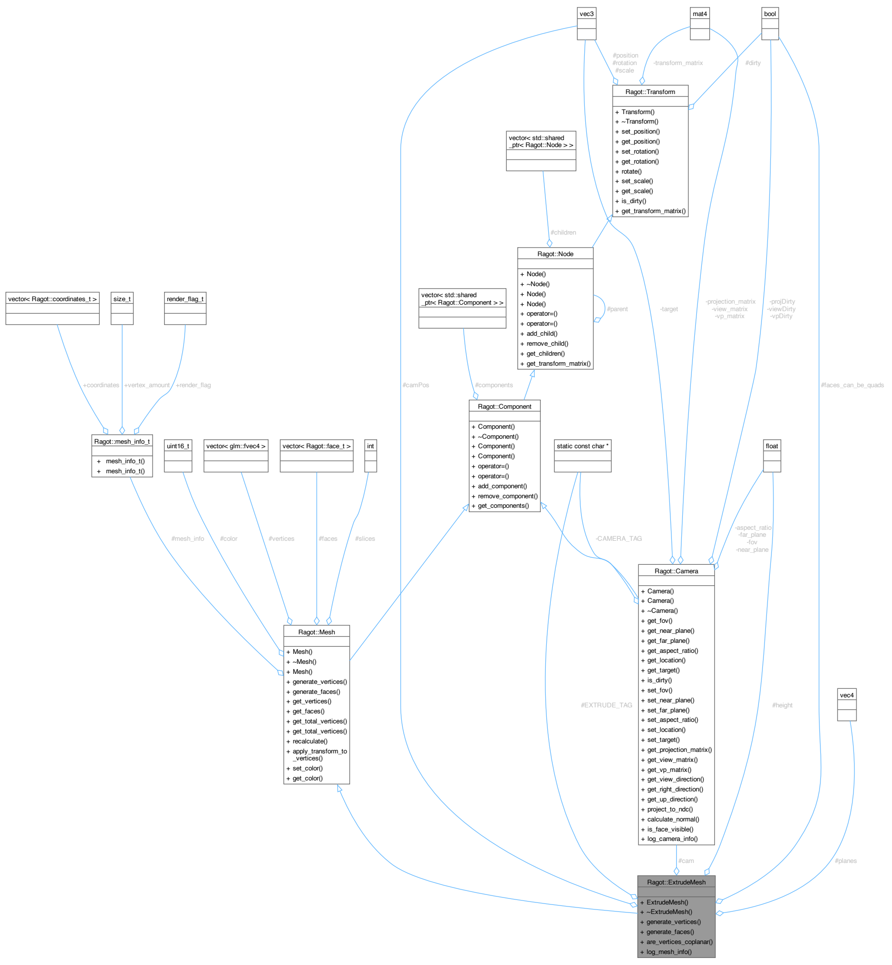
Represents a 3D mesh created by extruding a 2D shape along a specified height. This class inherits from the Mesh class and provides methods to generate vertices and faces for the extruded mesh. It also includes methods for culling faces based on the camera's view direction and logging mesh information.

```
#include <ExtrudeMesh.hpp>
```

Inheritance diagram for Ragot::ExtrudeMesh:



Collaboration diagram for Ragot::ExtrudeMesh:



Public Member Functions

- ExtrudeMesh (mesh_info_t &mesh_info, const Camera &cam)

  Constructs an ExtrudeMesh object with the specified mesh information and camera reference.
- ∼ExtrudeMesh ()=default

  Default destructor for the ExtrudeMesh class.
- void generate_vertices () override

  Deleted copy constructor for the ExtrudeMesh class.
- void generate_faces () override

  Generates the faces of the extruded mesh.
- bool are_vertices_coplanar (const glm::fvec4 &v1, const glm::fvec4 &v2, const glm::fvec4 &v3, const glm↩ ::fvec4 &v4, float tolerance=0.1f)

  Verifies if four vertices are coplanar.
- void log_mesh_info () const

  Logs detailed information about the mesh.

Public Member Functions inherited from Ragot::Mesh

- Mesh ()=delete

  Construct a new Mesh object (deleted constructor).
- virtual ∼Mesh ()=default

  Default virtual destructor for the Mesh class.
- Mesh (mesh_info_t &mesh_info)

  Construct a new Mesh object with mesh information.
- const std::vector< glm::fvec4 > & get_vertices () const

  Get the vertices object.
- const std::vector< face_t > & get_faces () const

  Get the faces object.
- const size_t get_total_vertices () const

  Get the total vertices object.
- size_t get_total_vertices ()

  Get the total vertices object.
- void recalculate ()

  Recalculate the mesh vertices and faces.
- void apply_transform_to_vertices ()

  Apply the current transformation to the vertices of the mesh. This method applies the transformation matrix obtained from the Transform class to each vertex in the mesh. This is useful for updating the mesh vertices after any transformation has been applied, such as translation, rotation, or scaling. It modifies the vertices in place, transforming them according to the current transformation matrix.
- void set_color (uint16_t new_color)

  Set the color of the mesh.
- uint16_t get_color () const

  Get the color of the mesh.

Public Member Functions inherited from Ragot::Component

- Component ()=default

  Default constructor for the Component class.
- virtual ∼Component ()=default

  Default virtual destructor for the Component class.
- Component (const Component &)=delete

  Deleted copy constructor for the Component class.
- Component (const Component &&)=delete

  Deleted move constructor for the Component class.
- Component & operator= (const Component &)=delete

  Deleted assignment operator for the Component class.
- Component & operator= (const Component &&)=delete

  Deleted move assignment operator for the Component class.
- void add_component (std::shared_ptr< Component > component)

  Adds a component to the collection.
- void remove_component (std::shared_ptr< Component > component)

  Removes a component from the collection.
- const std::vector< std::shared_ptr< Component > > get_components () const

  Gets the collection of components.

Public Member Functions inherited from Ragot::Node

- Node ()=default

  Default constructor for Node. Initializes an empty node with no parent and no children.
- virtual ∼Node ()=default

  Default destructor for Node. Cleans up the node and its children.
- Node (const Node &)=delete

  Deleted copy constructor for Node. Prevents copying of Node instances.
- Node (const Node &&)=delete

  Deleted move constructor for Node. Prevents moving of Node instances.
- Node & operator= (const Node &)=delete

  Deleted assignment operator for Node. Prevents assignment of Node instances.
- Node & operator= (const Node &&)=delete

  Deleted move assignment operator for Node. Prevents moving of Node instances.
- void add_child (std::shared_ptr< Node > child)

  Get the parent node.
- void remove_child (std::shared_ptr< Node > child)

  Remove a child node.
- const std::vector< std::shared_ptr< Node > > & get_children () const

  Get the parent node.
- mat4 get_transform_matrix () override

  Get the transform matrix object.

Public Member Functions inherited from Ragot::Transform

- Transform ()

  Default constructor for the Transform class.
- virtual ∼Transform ()=default

  Virtual destructor for the Transform class.
- void set_position (const vec3 &pos)

  Sets the position of the object.
- vec3 get_position () const

  Gets the current position of the object.
- void set_rotation (const vec3 &rot)

  Moves the object by a specified vector.
- vec3 get_rotation () const

  Gets the current rotation of the object.
- void rotate (const float angle, const vec3 &axis)

  Rotates the object by a specified angle around a given axis.
- void set_scale (const vec3 &scale)

  Sets the scale of the object.
- vec3 get_scale () const

  Sets the scale of the object uniformly.
- bool is_dirty () const

  Checks if the transformation matrix is dirty (needs recalculation).

Protected Attributes

- float height = 1.0f

  Height of the extrusion.
- bool faces_can_be_quads = false

  Flag indicating whether the faces can be quads.
- const Camera & cam

  Camera reference for culling faces based on the camera's view direction.
- glm::vec4 planes [4]

  Array of planes used for culling faces.
- glm::vec3 camPos

  Position of the camera in world space.

Protected Attributes inherited from Ragot::Mesh

- mesh_info_t mesh_info

  Information about the mesh, including coordinates and rendering type.
- uint16_t color = 0xFFFF

  Color of the mesh, default is white (0xFFFF).
- std::vector< glm::fvec4 > vertices

  Vector of vertices representing the mesh in 3D space.
- std::vector< face_t > faces

  Vector of faces representing the mesh, each face can be a triangle or a quad.
- int slices = 16

  Number of slices for generating the mesh, default is 16.

Protected Attributes inherited from Ragot::Component

- std::vector< std::shared_ptr< Component > > components

  Collection of components managed by this Component instance.

Protected Attributes inherited from Ragot::Node

- std::vector< std::shared_ptr< Node > > children

  List of child nodes.
- Node $*$ parent = nullptr

  Pointer to the parent node.

Protected Attributes inherited from Ragot::Transform

- vec3 position

  The position of the object in 3D space.
- vec3 rotation

  The rotation of the object in degrees around each axis (x, y, z).
- vec3 scale

  The scale of the object in 3D space, default is (1, 1, 1).
- bool dirty = true

  Flag indicating whether the transformation matrix needs to be recalculated.

Static Protected Attributes

- static const char ∗ EXTRUDE_TAG = "ExtrudeMesh"

  Tag for logging messages related to the ExtrudeMesh class.

6.9.1 Detailed Description

Represents a 3D mesh created by extruding a 2D shape along a specified height. This class inherits from the Mesh class and provides methods to generate vertices and faces for the extruded mesh. It also includes methods for culling faces based on the camera's view direction and logging mesh information.

The ExtrudeMesh class is designed to create a 3D mesh by extruding a 2D shape along a specified height. It uses the GLM library for vector and matrix operations, and includes functionality for face culling based on the camera's view direction. The class also provides a method to log detailed information about the mesh, including its position, rotation, scale, and vertex data.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 ExtrudeMesh()

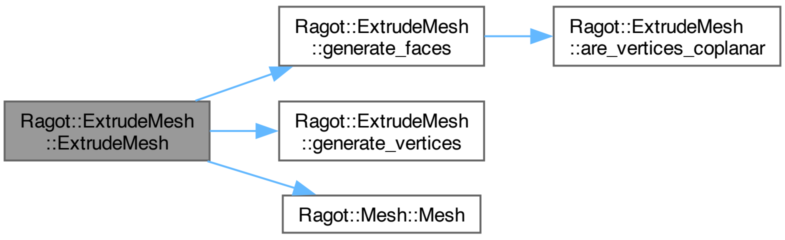
```
Ragot::ExtrudeMesh::ExtrudeMesh (
            mesh_info_t & mesh_info,
            const Camera & cam)  [inline]
```

Constructs an ExtrudeMesh object with the specified mesh information and camera reference.

Parameters

mesh_info	Information about the mesh to be extruded.
cam	Reference to the camera used for culling faces.

Here is the call graph for this function:


Ragot::ExtrudeMesh ::ExtrudeMesh
Ragot::ExtrudeMesh ::generate_faces
Ragot::ExtrudeMesh ::are_vertices_coplanar
Ragot::ExtrudeMesh ::generate_vertices
Ragot::Mesh::Mesh

6.9.2.2 ∼ExtrudeMesh()

```
Ragot::ExtrudeMesh::∼ExtrudeMesh ()  [default]
```

Default destructor for the ExtrudeMesh class.

This destructor is used to clean up resources when the ExtrudeMesh object is destroyed.

6.9.3 Member Function Documentation

6.9.3.1 are_vertices_coplanar()

```
bool Ragot::ExtrudeMesh::are_vertices_coplanar (
             const glm::fvec4 & v1,
             const glm::fvec4 & v2,
             const glm::fvec4 & v3,
             const glm::fvec4 & v4,
             float tolerance = 0.1f)
```

Verifies if four vertices are coplanar.

This method checks if the four vertices v1, v2, v3, and v4 are coplanar within a specified tolerance. It uses the scalar triple product to determine coplanarity.

Parameters

v1	First vertex in homogeneous coordinates.
v2	Second vertex in homogeneous coordinates.
v3	Third vertex in homogeneous coordinates.
v4	Fourth vertex in homogeneous coordinates.
tolerance	Tolerance value for coplanarity check (default is 0.1).

Returns

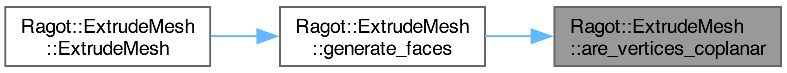
true if the vertices are coplanar, false otherwise.

The method calculates the scalar triple product of the vectors formed by the vertices and checks if it is close to zero within the specified tolerance. The scalar triple product is computed as the dot product of the first vector with the cross product of the other two vectors. This method is useful for determining if a set of vertices can form a valid face in the mesh.

Note

This method assumes that the vertices are provided in homogeneous coordinates (4D vectors).

Here is the caller graph for this function:


Ragot::ExtrudeMesh ::ExtrudeMesh
Ragot::ExtrudeMesh ::generate_faces
Ragot::ExtrudeMesh ::are_vertices_coplanar

6.9.3.2 generate_faces()

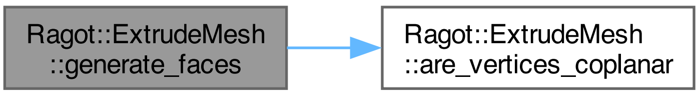
```
void Ragot::ExtrudeMesh::generate_faces ()  [override], [virtual]
```

Generates the faces of the extruded mesh.

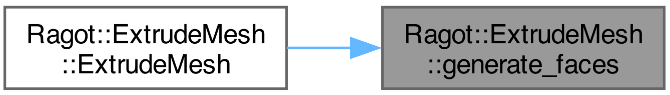
This method creates the faces of the mesh based on the vertices generated by the generate_vertices method. It checks if the vertices are coplanar and creates either quads or triangles accordingly.

Implements Ragot::Mesh.

Here is the call graph for this function:


Ragot::ExtrudeMesh ::generate_faces
Ragot::ExtrudeMesh ::are_vertices_coplanar

Here is the caller graph for this function:


Ragot::ExtrudeMesh ::ExtrudeMesh
Ragot::ExtrudeMesh ::generate_faces

6.9.3.3 generate_vertices()

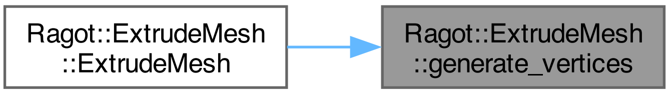
```
void Ragot::ExtrudeMesh::generate_vertices ()  [override], [virtual]
```

Deleted copy constructor for the ExtrudeMesh class.

This constructor is deleted to prevent copying of ExtrudeMesh objects.

Implements Ragot::Mesh.

Here is the caller graph for this function:

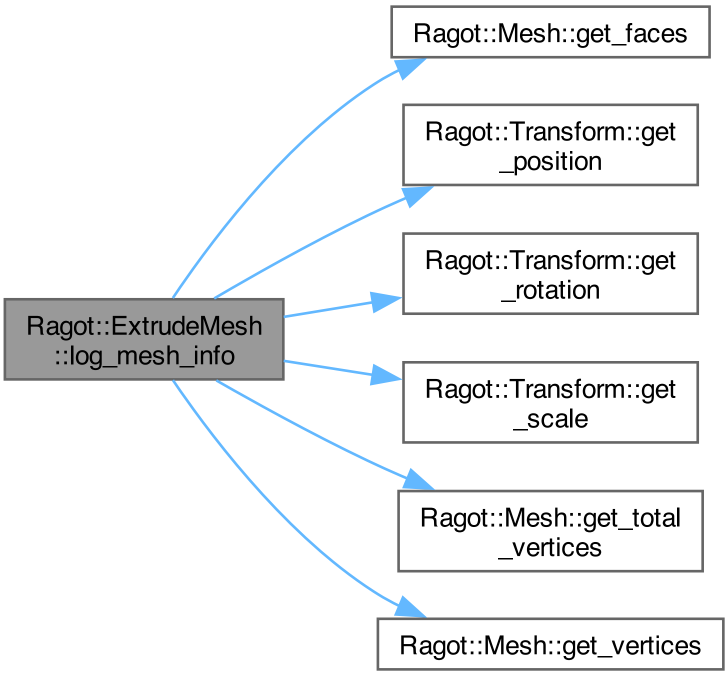

Ragot::ExtrudeMesh ::ExtrudeMesh
Ragot::ExtrudeMesh ::generate_vertices

6.9.3.4 log_mesh_info()

```
void Ragot::ExtrudeMesh::log_mesh_info () const
```

Logs detailed information about the mesh.

Here is the call graph for this function:


Ragot::ExtrudeMesh ::log_mesh_info
Ragot::Mesh::get_faces
Ragot::Transform::get _position
Ragot::Transform::get _rotation
Ragot::Transform::get _scale
Ragot::Mesh::get_total _vertices
Ragot::Mesh::get_vertices

6.9.4 Member Data Documentation

6.9.4.1 cam

```
const Camera& Ragot::ExtrudeMesh::cam [protected]
```

Camera reference for culling faces based on the camera's view direction.

This reference is used to determine which faces of the mesh are visible from the camera's perspective.

6.9.4.2 camPos

```
glm::vec3 Ragot::ExtrudeMesh::camPos [protected]
```

Position of the camera in world space.

This vector represents the position of the camera in the 3D world. It is used to calculate the visibility of faces based on the camera's position.

6.9.4.3 EXTRUDE_TAG

```
const char * Ragot::ExtrudeMesh::EXTRUDE_TAG = "ExtrudeMesh"  [static], [protected]
```

Tag for logging messages related to the ExtrudeMesh class.

6.9.4.4 faces_can_be_quads

```
bool Ragot::ExtrudeMesh::faces_can_be_quads = false  [protected]
```

Flag indicating whether the faces can be quads.

This flag is set to true if the number of vertices is a multiple of 8 or if it is exactly 4. It determines how faces are generated in the mesh.

6.9.4.5 height

```
float Ragot::ExtrudeMesh::height = 1.0f  [protected]
```

Height of the extrusion.

This value determines how far the 2D shape is extruded in the Z direction.

6.9.4.6 planes

```
glm::vec4 Ragot::ExtrudeMesh::planes[4]  [protected]
```

Array of planes used for culling faces.

This array contains the planes that define the view frustum of the camera. It is used to determine which faces are visible and which can be culled.

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

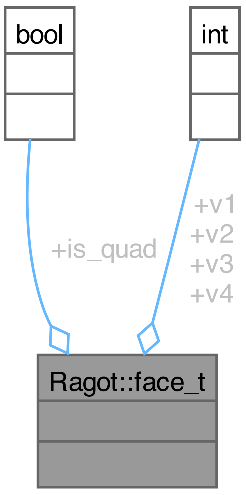
- main/ExtrudeMesh.hpp
- main/ExtrudeMesh.cpp

6.10 Ragot::face_t Struct Reference

Represents a face in a 3D mesh.

`#include <CommonTypes.hpp>`

Collaboration diagram for Ragot::face_t:



Public Attributes

- bool is_quad
- int v1
- int v2
- int v3
- int v4

6.10.1 Detailed Description

Represents a face in a 3D mesh.

This structure can represent either a triangle or a quadrilateral face, depending on the `is_quad` flag.

6.10.2 Member Data Documentation

6.10.2.1 is_quad

```
bool Ragot::face_t::is_quad
```

6.10.2.2 v1

```
int Ragot::face_t::v1
```

6.10.2.3 v2

```
int Ragot::face_t::v2
```

6.10.2.4 v3

```
int Ragot::face_t::v3
```

6.10.2.5 v4

```
int Ragot::face_t::v4
```

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

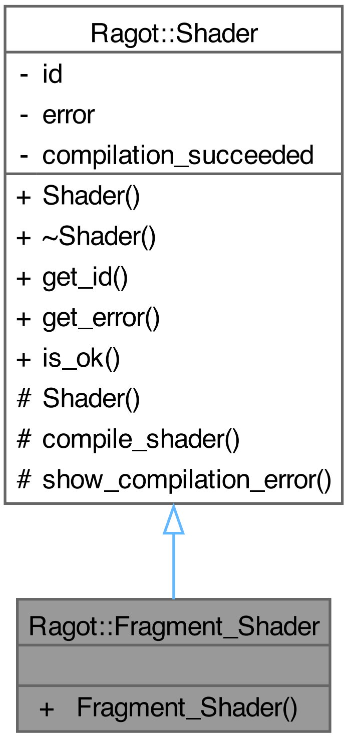
- main/CommonTypes.hpp

6.11 Ragot::Fragment_Shader Class Reference

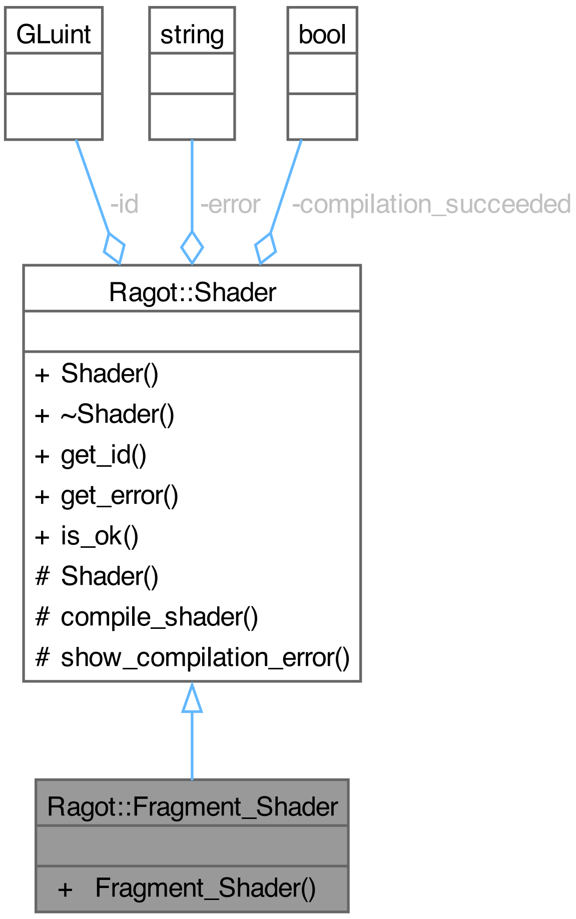
Class for managing an OpenGL fragment shader.

```
#include <Shader_Program.hpp>
```

Inheritance diagram for Ragot::Fragment_Shader:



Collaboration diagram for Ragot::Fragment_Shader:



Public Member Functions

- Fragment_Shader (const vector< string > &source_code)

  Constructor for the Fragment_Shader class.

Public Member Functions inherited from Ragot::Shader

- Shader ()=delete

  Deleted default constructor.
- ∼Shader ()

  Destructor for the Shader class.
- GLuint get_id () const

  Gets the shader ID.
- string ∗ get_error ()

  Gets the compilation error message.
- bool is_ok () const

  Checks if the shader is compiled successfully.

Additional Inherited Members

Protected Member Functions inherited from Ragot::Shader

- Shader (const vector< string > &source_code, GLenum type)

  Constructor for the Shader class.
- GLuint compile_shader ()

  Compiles the shader.
- void show_compilation_error ()

  Displays compilation errors.

6.11.1 Detailed Description

Class for managing an OpenGL fragment shader.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 Fragment_Shader()

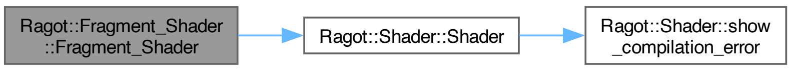
```
Ragot::Fragment_Shader::Fragment_Shader (
            const vector< string > & source_code)  [inline]
```

Constructor for the Fragment_Shader class.

Parameters

source_code	Vector of fragment shader source code.

Here is the call graph for this function:


Ragot::Fragment_Shader ::Fragment_Shader
Ragot::Shader::Shader
Ragot::Shader::show _compilation_error

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

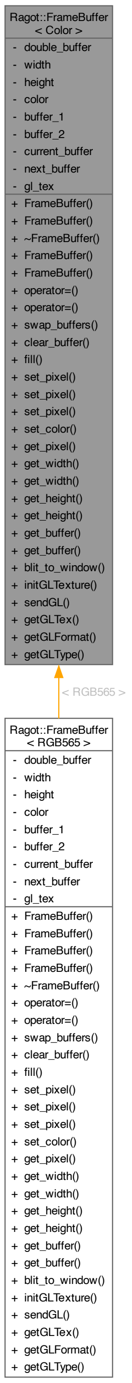
- main/Shader_Program.hpp

6.12 Ragot::FrameBuffer< Color > Class Template Reference

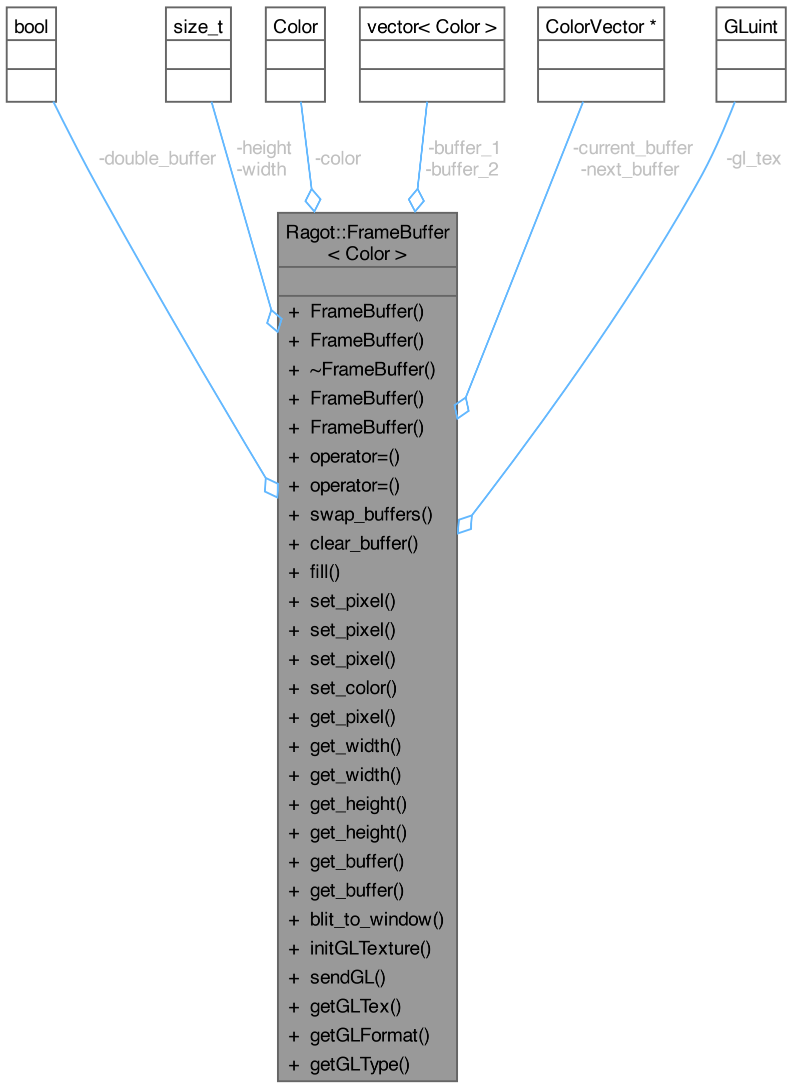
Class to manage a frame buffer for rendering graphics.

`#include <FrameBuffer.hpp>`

Inheritance diagram for Ragot::FrameBuffer< Color >:



Collaboration diagram for Ragot::FrameBuffer< Color >:



Public Types

- using TYPE = Color
- using ColorVector = std::vector < Color >

Public Member Functions

- FrameBuffer (size_t width, size_t height, bool double_buffer)

  Constructor for the FrameBuffer class.
- FrameBuffer ()=delete

  Default constructor for the FrameBuffer class (Deleted).
- ∼FrameBuffer ()=default

Default destructor for the FrameBuffer class.
- FrameBuffer (const FrameBuffer &)=delete

  Construct a new Frame Buffer object (Deleted).
- FrameBuffer (const FrameBuffer &&)=delete

  Construct a new Frame Buffer object (Deleted).
- FrameBuffer & operator= (const FrameBuffer &)=delete

  Assignment operator for the FrameBuffer class (Deleted).
- FrameBuffer & operator= (const FrameBuffer &&)=delete

  Assignment operator for the FrameBuffer class (Deleted).
- void swap_buffers ()

  Swaps the current buffer with the next buffer.
- void clear_buffer (Buffer buffer_to_clear=NEXT_BUFFER)

  Clears the specified buffer by filling it with the default color.
- void fill (Color color=0, Buffer buffer_to_fill=NEXT_BUFFER)

  Fills the specified buffer with the given color.
- void set_pixel (size_t x, size_t y, Color color)

  Sets a pixel at the specified coordinates to the given color.
- void set_pixel (size_t offset, Color color)

  Sets a pixel at the specified offset in the buffer to the given color.
- void set_pixel (size_t offset)

  Sets a pixel at the specified offset in the buffer to the default color.
- void set_color (Color color)

  Sets the default color for the frame buffer.
- Color get_pixel (size_t x, size_t y) const

  Gets the color of a pixel at the specified coordinates.
- size_t get_width ()

  Gets the color of a pixel at the specified offset in the buffer.
- size_t get_width () const

  Gets the width of the frame buffer.
- size_t get_height ()

  Gets the height of the frame buffer.
- size_t get_height () const

  Gets the height of the frame buffer.
- const Color ∗ get_buffer () const

  Gets the current buffer being used.
- Color ∗ get_buffer ()

  Get the buffer object.
- void blit_to_window () const

  Blits the current buffer to the window. This method copies the contents of the current buffer to the next buffer, effectively preparing the next frame for rendering.
- void initGLTexture ()

  Initializes the OpenGL texture for the frame buffer.
- void sendGL () const

  Clears the OpenGL texture associated with the frame buffer.
- GLuint getGLTex () const

  Gets the OpenGL texture ID for the frame buffer.

Static Public Member Functions

- static GLenum getGLFormat ()

  Sets the OpenGL texture for the frame buffer.
- static GLenum getGLType ()

  Gets the OpenGL type for the frame buffer.

Private Attributes

- bool double_buffer

    Flag to indicate if double buffering is enabled.
- size_t width

    Width of the frame buffer in pixels.
- size_t height

    Height of the frame buffer in pixels.
- Color color

    Default color for filling the buffer.
- ColorVector buffer_1

    First buffer for single or double buffering.
- ColorVector buffer_2

    Second buffer for double buffering (if enabled)
- ColorVector ∗ current_buffer

    Pointer to the current buffer being used.
- ColorVector ∗ next_buffer

    Pointer to the next buffer to be used (for double buffering)
- GLuint gl_tex = 0

    OpenGL texture ID for the frame buffer.

6.12.1 Detailed Description

template<typename Color>
class Ragot::FrameBuffer< Color >

Class to manage a frame buffer for rendering graphics.

This class provides methods to create a frame buffer, swap buffers, clear the buffer, fill it with a color, set and get pixels, and manage OpenGL textures. It supports both single and double buffering modes.

6.12.2 Member Typedef Documentation

6.12.2.1 ColorVector

```
template<typename Color>
using Ragot::FrameBuffer< Color >::ColorVector = std::vector < Color >
```

6.12.2.2 TYPE

```
template<typename Color>
using Ragot::FrameBuffer< Color >::TYPE = Color
```

6.12.3 Constructor & Destructor Documentation

6.12.3.1 FrameBuffer() [1/4]

```
template<typename Color>
Ragot::FrameBuffer< Color >::FrameBuffer (
             size_t width,
             size_t height,
             bool double_buffer)
```

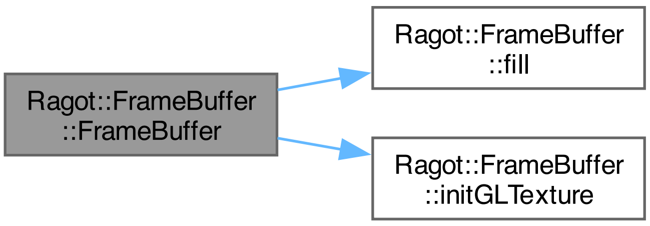
Constructor for the FrameBuffer class.

Initializes the frame buffer with the specified width, height, and double buffering option. Allocates memory for the buffers and fills them with the default color.

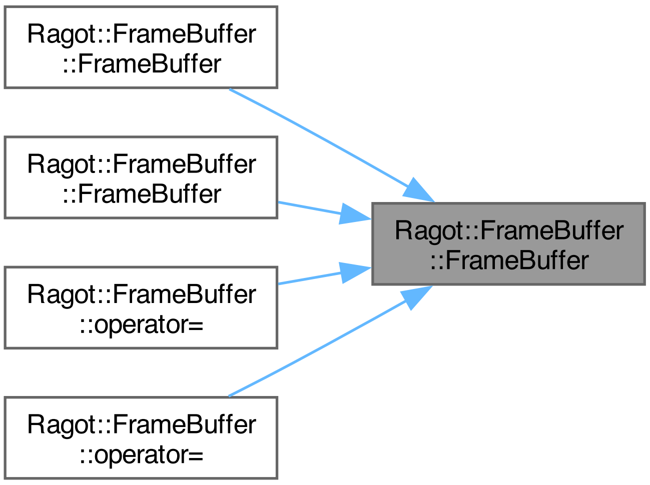
Parameters

width	Width of the frame buffer in pixels.
height	Height of the frame buffer in pixels.
double_buffer	Flag to indicate if double buffering is enabled.

Here is the call graph for this function:


Ragot::FrameBuffer ::FrameBuffer
Ragot::FrameBuffer ::fill
Ragot::FrameBuffer ::initGLTexture

Here is the caller graph for this function:


Ragot::FrameBuffer ::FrameBuffer
Ragot::FrameBuffer ::FrameBuffer
Ragot::FrameBuffer ::operator=
Ragot::FrameBuffer ::operator=
Ragot::FrameBuffer ::FrameBuffer

6.12.3.2 FrameBuffer() [2/4]

```
template<typename Color>
Ragot::FrameBuffer< Color >::FrameBuffer ()  [delete]
```

Default constructor for the FrameBuffer class (Deleted).

6.12.3.3 ∼FrameBuffer()

```
template<typename Color>
Ragot::FrameBuffer< Color >::∼FrameBuffer ()  [default]
```

Default destructor for the FrameBuffer class.

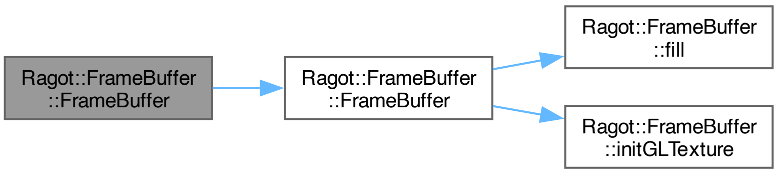
Cleans up resources used by the frame buffer.

6.12.3.4 FrameBuffer() [3/4]

```
template<typename Color>
Ragot::FrameBuffer< Color >::FrameBuffer (
            const FrameBuffer< Color > & )  [delete]
```

Construct a new Frame Buffer object (Deleted).

Here is the call graph for this function:

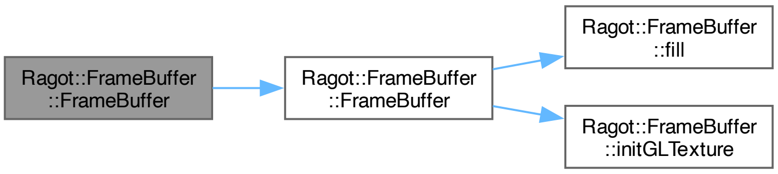

Ragot::FrameBuffer ::FrameBuffer
Ragot::FrameBuffer ::FrameBuffer
Ragot::FrameBuffer ::fill
Ragot::FrameBuffer ::initGLTexture

6.12.3.5 FrameBuffer() [4/4]

```
template<typename Color>
Ragot::FrameBuffer< Color >::FrameBuffer (
            const FrameBuffer< Color > && )  [delete]
```

Construct a new Frame Buffer object (Deleted).

Here is the call graph for this function:


Ragot::FrameBuffer ::FrameBuffer
Ragot::FrameBuffer ::FrameBuffer
Ragot::FrameBuffer ::fill
Ragot::FrameBuffer ::initGLTexture

6.12.4 Member Function Documentation

6.12.4.1 blit_to_window()

```
template<typename Color>
void Ragot::FrameBuffer< Color >::blit_to_window () const  [inline]
```

Blits the current buffer to the window. This method copies the contents of the current buffer to the next buffer, effectively preparing the next frame for rendering.

6.12.4.2 clear_buffer()

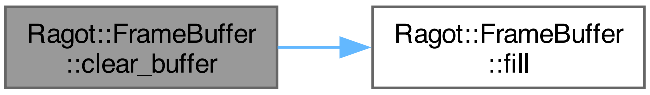
```
template<typename Color>
void Ragot::FrameBuffer< Color >::clear_buffer (
            Buffer buffer_to_clear = NEXT_BUFFER)
```

Clears the specified buffer by filling it with the default color.

Parameters

buffer_to_clear	The buffer to clear (CURRENT_BUFFER, NEXT_BUFFER, or MAX_BUFFER).

Here is the call graph for this function:



6.12.4.3 fill()

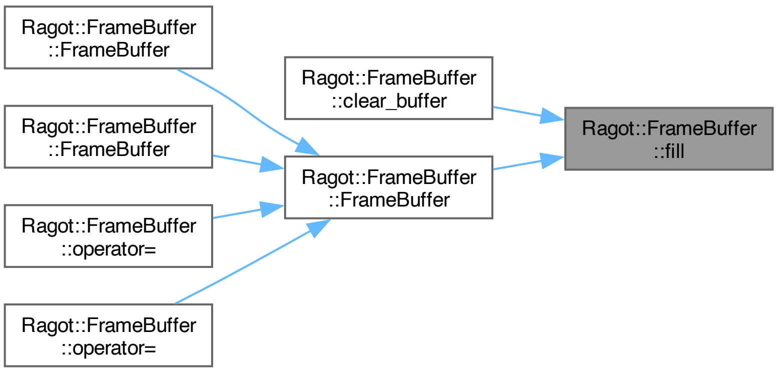
```
template<typename Color>
void Ragot::FrameBuffer< Color >::fill (
            Color color = 0,
            Buffer buffer_to_fill = NEXT_BUFFER)
```

Fills the specified buffer with the given color.

Parameters

color	The color to fill the buffer with (default is 0).
buffer_to↩ _fill	The buffer to fill (CURRENT_BUFFER, NEXT_BUFFER, or MAX_BUFFER).

Here is the caller graph for this function:


Ragot::FrameBuffer ::FrameBuffer
Ragot::FrameBuffer ::FrameBuffer
Ragot::FrameBuffer ::operator=
Ragot::FrameBuffer ::operator=
Ragot::FrameBuffer ::clear_buffer
Ragot::FrameBuffer ::FrameBuffer
Ragot::FrameBuffer ::fill

6.12.4.4 get_buffer() `[1/2]`

```
template<typename Color>
Color * Ragot::FrameBuffer< Color >::get_buffer ()  [inline]
```

Get the buffer object.

Returns

Color∗

6.12.4.5 get_buffer() `[2/2]`

```
template<typename Color>
const Color * Ragot::FrameBuffer< Color >::get_buffer () const  [inline]
```

Gets the current buffer being used.

Returns

const Color∗ Pointer to the current buffer data.

6.12.4.6 get_height() `[1/2]`

```
template<typename Color>
size_t Ragot::FrameBuffer< Color >::get_height ()  [inline]
```

Gets the height of the frame buffer.

Returns

size_t The height of the frame buffer in pixels.

6.12.4.7 get_height() [2/2]

```
template<typename Color>
size_t Ragot::FrameBuffer< Color >::get_height () const  [inline]
```

Gets the height of the frame buffer.

Returns

size_t The height of the frame buffer in pixels.

6.12.4.8 get_pixel()

```
template<typename Color>
Color Ragot::FrameBuffer< Color >::get_pixel (
            size_t x,
            size_t y) const
```

Gets the color of a pixel at the specified coordinates.

Parameters

x	X coordinate of the pixel.
y	Y coordinate of the pixel.

Returns

Color The color of the pixel at the specified coordinates.

6.12.4.9 get_width() [1/2]

```
template<typename Color>
size_t Ragot::FrameBuffer< Color >::get_width ()  [inline]
```

Gets the color of a pixel at the specified offset in the buffer.

Parameters

offset	Offset in the buffer (calculated as y * width + x).

Returns

Color The color of the pixel at the specified offset.

6.12.4.10 get_width() [2/2]

```
template<typename Color>
size_t Ragot::FrameBuffer< Color >::get_width () const  [inline]
```

Gets the width of the frame buffer.

Returns

size_t The width of the frame buffer in pixels.

6.12.4.11 getGLFormat()

```
template<typename Color>
GLenum Ragot::FrameBuffer< Color >::getGLFormat ()  [static]
```

Sets the OpenGL texture for the frame buffer.

This method binds the OpenGL texture to the current context.

6.12.4.12 getGLTex()

```
template<typename Color>
GLuint Ragot::FrameBuffer< Color >::getGLTex () const  [inline]
```

Gets the OpenGL texture ID for the frame buffer.

Returns

GLuint The OpenGL texture ID.

6.12.4.13 getGLType()

```
template<typename Color>
GLenum Ragot::FrameBuffer< Color >::getGLType ()  [static]
```

Gets the OpenGL type for the frame buffer.

This method returns the OpenGL type corresponding to the color format used in the frame buffer.

Returns

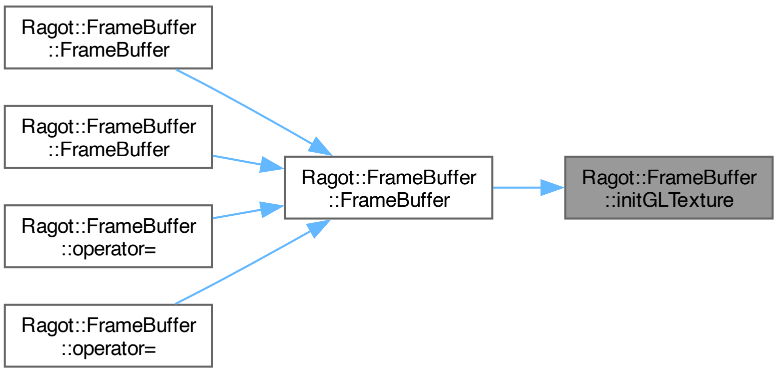
GLenum The OpenGL type for the frame buffer.

6.12.4.14 initGLTexture()

```
template<typename Color>
void Ragot::FrameBuffer< Color >::initGLTexture ()
```

Initializes the OpenGL texture for the frame buffer.

This method creates an OpenGL texture and binds it to the frame buffer. Here is the caller graph for this function:


Ragot::FrameBuffer ::FrameBuffer
Ragot::FrameBuffer ::FrameBuffer
Ragot::FrameBuffer ::operator=
Ragot::FrameBuffer ::operator=
Ragot::FrameBuffer ::FrameBuffer
Ragot::FrameBuffer ::initGLTexture

6.12.4.15 operator=() [1/2]

```
template<typename Color>
FrameBuffer & Ragot::FrameBuffer< Color >::operator= (
             const FrameBuffer< Color > && )  [delete]
```

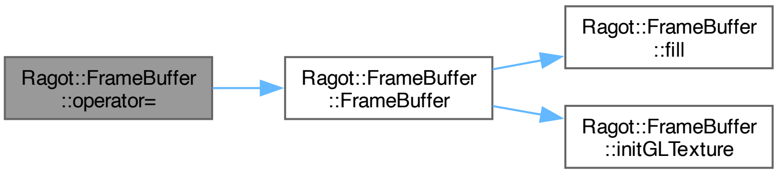
Assignment operator for the FrameBuffer class (Deleted).

Prevents assignment of FrameBuffer objects.

Returns

FrameBuffer& Reference to the current object.

Here is the call graph for this function:


Ragot::FrameBuffer ::operator=
Ragot::FrameBuffer ::FrameBuffer
Ragot::FrameBuffer ::fill
Ragot::FrameBuffer ::initGLTexture

6.12.4.16 operator=() [2/2]

```
template<typename Color>
FrameBuffer & Ragot::FrameBuffer< Color >::operator= (
            const FrameBuffer< Color > & )  [delete]
```

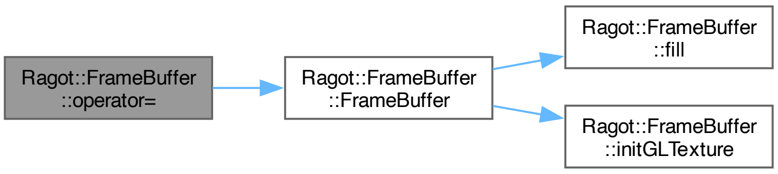
Assignment operator for the FrameBuffer class (Deleted).

Prevents assignment of FrameBuffer objects.

Returns

FrameBuffer& Reference to the current object.

Here is the call graph for this function:


Ragot::FrameBuffer ::operator=
Ragot::FrameBuffer ::FrameBuffer
Ragot::FrameBuffer ::fill
Ragot::FrameBuffer ::initGLTexture

6.12.4.17 sendGL()

```
template<typename Color>
void Ragot::FrameBuffer< Color >::sendGL () const
```

Clears the OpenGL texture associated with the frame buffer.

This method deletes the OpenGL texture to free up resources.

6.12.4.18 set_color()

```
template<typename Color>
void Ragot::FrameBuffer< Color >::set_color (
            Color color)
```

Sets the default color for the frame buffer.

Parameters

color	The color to set as the default.

6.12.4.19 set_pixel() [1/3]

```
template<typename Color>
void Ragot::FrameBuffer< Color >::set_pixel (
            size_t offset)
```

Sets a pixel at the specified offset in the buffer to the default color.

Parameters

offset	Offset in the buffer (calculated as y ∗ width + x).

6.12.4.20 set_pixel() [2/3]

```
template<typename Color>
void Ragot::FrameBuffer< Color >::set_pixel (
            size_t offset,
            Color color)
```

Sets a pixel at the specified offset in the buffer to the given color.

Parameters

offset	Offset in the buffer (calculated as y ∗ width + x).
color	Color value for the pixel.

6.12.4.21 set_pixel() [3/3]

```
template<typename Color>
void Ragot::FrameBuffer< Color >::set_pixel (
            size_t x,
            size_t y,
            Color color)
```

Sets a pixel at the specified coordinates to the given color.

Parameters

x	X coordinate of the pixel.
y	Y coordinate of the pixel.
color	Color value for the pixel.

6.12.4.22 swap_buffers()

```
template<typename Color>
void Ragot::FrameBuffer< Color >::swap_buffers ()
```

Swaps the current buffer with the next buffer.

This method is used in double buffering to switch between the buffers for rendering.

6.12.5 Member Data Documentation

6.12.5.1 buffer_1

```
template<typename Color>
ColorVector Ragot::FrameBuffer< Color >::buffer_1  [private]
```

First buffer for single or double buffering.

6.12.5.2 buffer_2

```
template<typename Color>
ColorVector Ragot::FrameBuffer< Color >::buffer_2  [private]
```

Second buffer for double buffering (if enabled)

6.12.5.3 color

```
template<typename Color>
Color Ragot::FrameBuffer< Color >::color  [private]
```

Default color for filling the buffer.

6.12.5.4 current_buffer

```
template<typename Color>
ColorVector* Ragot::FrameBuffer< Color >::current_buffer  [private]
```

Pointer to the current buffer being used.

6.12.5.5 double_buffer

```
template<typename Color>
bool Ragot::FrameBuffer< Color >::double_buffer  [private]
```

Flag to indicate if double buffering is enabled.

6.12.5.6 gl_tex

```
template<typename Color>
GLuint Ragot::FrameBuffer< Color >::gl_tex = 0  [private]
```

OpenGL texture ID for the frame buffer.

This variable holds the OpenGL texture ID used for rendering the frame buffer.

6.12.5.7 height

```
template<typename Color>
size_t Ragot::FrameBuffer< Color >::height  [private]
```

Height of the frame buffer in pixels.

6.12.5.8 next_buffer

```
template<typename Color>
ColorVector* Ragot::FrameBuffer< Color >::next_buffer  [private]
```

Pointer to the next buffer to be used (for double buffering)

6.12.5.9 width

```
template<typename Color>
size_t Ragot::FrameBuffer< Color >::width  [private]
```

Width of the frame buffer in pixels.

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

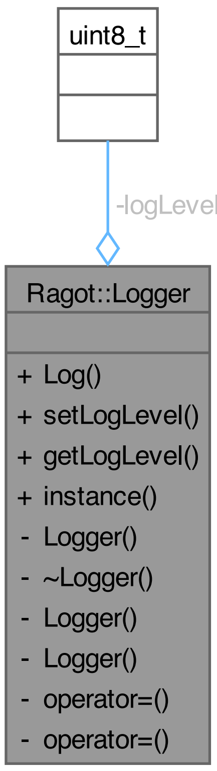
- main/FrameBuffer.hpp
- main/FrameBuffer.cpp

6.13 Ragot::Logger Class Reference

Singleton logger class for the Ragot engine.

`#include <Logger.hpp>`

Collaboration diagram for Ragot::Logger:



Public Member Functions

- template<typename... Args>
  void Log (const char ∗TAG, uint8_t level, const char ∗fmt, Args... args)

  Logs a message with the specified tag and level.
- void setLogLevel (uint8_t level)

  Sets the log level for the logger.
- uint8_t getLogLevel () const

  Gets the current log level of the logger.

Static Public Member Functions

- static Logger & instance ()

  Gets the singleton instance of the Logger class.

Private Member Functions

- Logger ()=default

  Default constructor for the Logger class.
- ∼Logger ()=default

  Default destructor for the Logger class.
- Logger (const Logger &)=delete

  Construct a new Logger object (Deleted).
- Logger (const Logger &&)=delete

  Construct a new Logger object (Deleted).
- Logger & operator= (const Logger &)=delete

  Assignment operator for the Logger class (Deleted).
- Logger & operator= (const Logger &&)=delete

  Assignment operator for the Logger class (Deleted).

Private Attributes

- uint8_t logLevel = 0

  Current log level for the logger.

6.13.1 Detailed Description

Singleton logger class for the Ragot engine.

This class provides a singleton logger that allows logging messages with different severity levels (INFO, WARNING, ERROR). It supports formatted logging using printf-style format strings and can be used across different platforms.

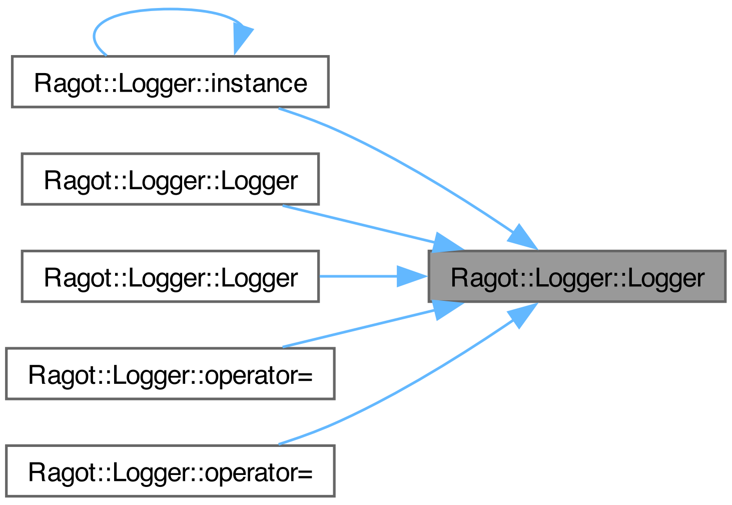
6.13.2 Constructor & Destructor Documentation

6.13.2.1 Logger() [1/3]

```
Ragot::Logger::Logger ()  [private], [default]
```

Default constructor for the Logger class.

This constructor is private to enforce the singleton pattern. Here is the caller graph for this function:



6.13.2.2 ∼Logger()

```
Ragot::Logger::∼Logger ()  [private], [default]
```

Default destructor for the Logger class.

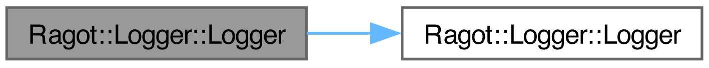
This destructor is defaulted and does not perform any special cleanup.

6.13.2.3 Logger() [2/3]

```
Ragot::Logger::Logger (
            const Logger & )  [private], [delete]
```

Construct a new Logger object (Deleted).

Here is the call graph for this function:

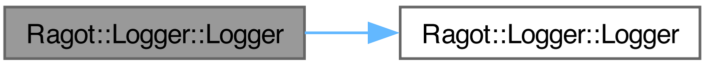


6.13.2.4 Logger() [3/3]

```
Ragot::Logger::Logger (
             const Logger && )  [private], [delete]
```

Construct a new Logger object (Deleted).

This constructor is deleted to prevent moving the Logger instance. Here is the call graph for this function:



6.13.3 Member Function Documentation

6.13.3.1 getLogLevel()

```
uint8_t Ragot::Logger::getLogLevel () const  [inline]
```

Gets the current log level of the logger.

This method returns the current log level, which determines the severity of messages that will be logged.

Returns

uint8_t The current log level (0 = INFO, 1 = WARNING, 2 = ERROR).

6.13.3.2 instance()

```
static Logger & Ragot::Logger::instance ()  [inline], [static]
```

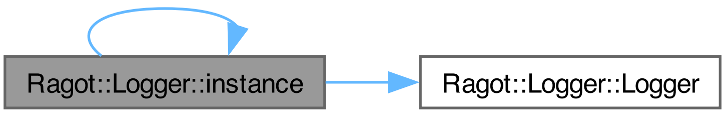
Gets the singleton instance of the Logger class.

This method returns a reference to the singleton Logger instance.

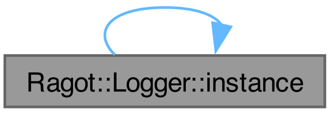
Returns

Logger& Reference to the singleton Logger instance.

Here is the call graph for this function:



Here is the caller graph for this function:



6.13.3.3 Log()

```
template<typename... Args>
void Ragot::Logger::Log (
             const char * TAG,
             uint8_t level,
             const char * fmt,
             Args... args) [inline]
```

Logs a message with the specified tag and level.

This method logs a message with the given tag and severity level. The message can be formatted using printf-style format strings.

Parameters

TAG	The tag for the log message.
level	The severity level of the log message (0 = INFO, 1 = WARNING, 2 = ERROR).
fmt	The format string for the log message.
args	The arguments to format the log message.

6.13.3.4 operator=() [1/2]

```
Logger & Ragot::Logger::operator= (
             const Logger && )  [private], [delete]
```

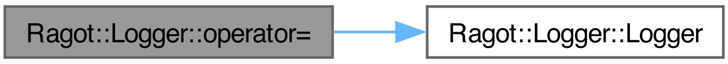
Assignment operator for the Logger class (Deleted).

This operator is deleted to prevent moving Logger instances.

Returns

Logger& Reference to the current object.

Here is the call graph for this function:



6.13.3.5 operator=() [2/2]

```
Logger & Ragot::Logger::operator= (
             const Logger & )  [private], [delete]
```

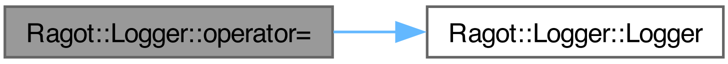
Assignment operator for the Logger class (Deleted).

This operator is deleted to prevent assignment of Logger instances.

Returns

Logger& Reference to the current object.

Here is the call graph for this function:



6.13.3.6 setLogLevel()

```
void Ragot::Logger::setLogLevel (
             uint8_t level)  [inline]
```

Sets the log level for the logger.

This method sets the log level for the logger, which determines the severity of messages that will be logged. It also configures the ESP-IDF logging system if running on an ESP platform.

Parameters

level	The new log level to set (0 = INFO, 1 = WARNING, 2 = ERROR).

6.13.4 Member Data Documentation

6.13.4.1 logLevel

```
uint8_t Ragot::Logger::logLevel = 0  [private]
```

Current log level for the logger.

This variable stores the current log level, which determines the severity of messages that will be logged. 0 = INFO, 1 = WARNING, 2 = ERROR.

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

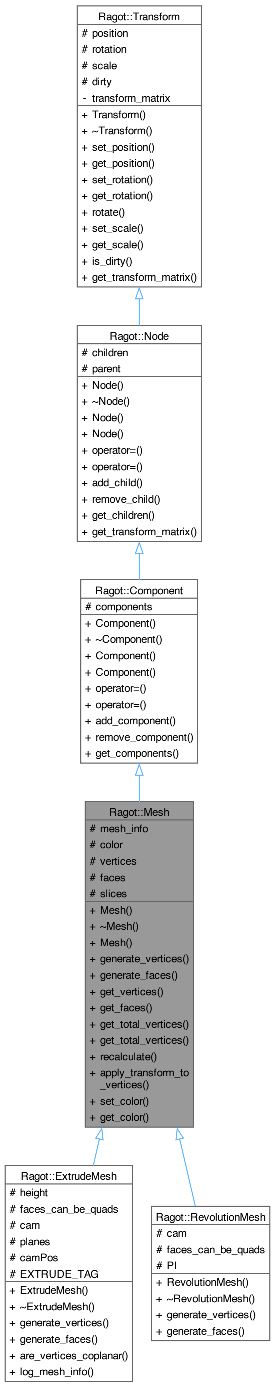
- main/Logger.hpp

6.14 Ragot::Mesh Class Reference

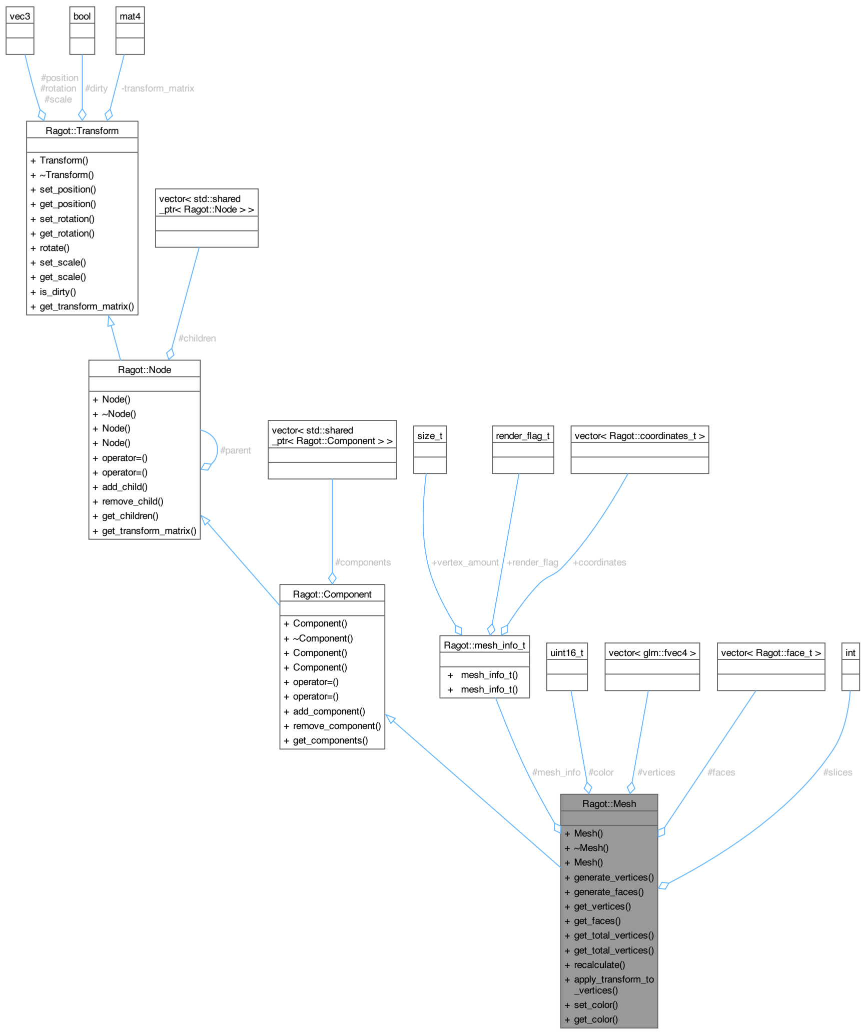
Represents a 3D mesh in the Ragot engine.

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

Inheritance diagram for Ragot::Mesh:



Collaboration diagram for Ragot::Mesh:



Public Member Functions

- Mesh ()=delete

  Construct a new Mesh object (deleted constructor).
- virtual ∼Mesh ()=default

  Default virtual destructor for the Mesh class.
- Mesh (mesh_info_t &mesh_info)

  Construct a new Mesh object with mesh information.
- virtual void generate_vertices ()=0

  Generate vertices for the mesh.
- virtual void generate_faces ()=0

  Generate faces for the mesh.
- const std::vector< glm::fvec4 > & get_vertices () const

  Get the vertices object.
- const std::vector< face_t > & get_faces () const

  Get the faces object.

- const size_t get_total_vertices () const

  Get the total vertices object.
- size_t get_total_vertices ()

  Get the total vertices object.
- void recalculate ()

  Recalculate the mesh vertices and faces.
- void apply_transform_to_vertices ()

  Apply the current transformation to the vertices of the mesh. This method applies the transformation matrix obtained from the Transform class to each vertex in the mesh. This is useful for updating the mesh vertices after any transformation has been applied, such as translation, rotation, or scaling. It modifies the vertices in place, transforming them according to the current transformation matrix.
- void set_color (uint16_t new_color)

  Set the color of the mesh.
- uint16_t get_color () const

  Get the color of the mesh.

Public Member Functions inherited from Ragot::Component

- Component ()=default

  Default constructor for the Component class.
- virtual ∼Component ()=default

  Default virtual destructor for the Component class.
- Component (const Component &)=delete

  Deleted copy constructor for the Component class.
- Component (const Component &&)=delete

  Deleted move constructor for the Component class.
- Component & operator= (const Component &)=delete

  Deleted assignment operator for the Component class.
- Component & operator= (const Component &&)=delete

  Deleted move assignment operator for the Component class.
- void add_component (std::shared_ptr< Component > component)

  Adds a component to the collection.
- void remove_component (std::shared_ptr< Component > component)

  Removes a component from the collection.
- const std::vector< std::shared_ptr< Component > > get_components () const

  Gets the collection of components.

Public Member Functions inherited from Ragot::Node

- Node ()=default

  Default constructor for Node. Initializes an empty node with no parent and no children.
- virtual ∼Node ()=default

  Default destructor for Node. Cleans up the node and its children.
- Node (const Node &)=delete

  Deleted copy constructor for Node. Prevents copying of Node instances.
- Node (const Node &&)=delete

  Deleted move constructor for Node. Prevents moving of Node instances.
- Node & operator= (const Node &)=delete

  Deleted assignment operator for Node. Prevents assignment of Node instances.
- Node & operator= (const Node &&)=delete

Deleted move assignment operator for Node. Prevents moving of Node instances.

- void add_child (std::shared_ptr< Node > child)

  Get the parent node.
- void remove_child (std::shared_ptr< Node > child)

  Remove a child node.
- const std::vector< std::shared_ptr< Node > > & get_children () const

  Get the parent node.
- mat4 get_transform_matrix () override

  Get the transform matrix object.

Public Member Functions inherited from Ragot::Transform

- Transform ()

  Default constructor for the Transform class.
- virtual ∼Transform ()=default

  Virtual destructor for the Transform class.
- void set_position (const vec3 &pos)

  Sets the position of the object.
- vec3 get_position () const

  Gets the current position of the object.
- void set_rotation (const vec3 &rot)

  Moves the object by a specified vector.
- vec3 get_rotation () const

  Gets the current rotation of the object.
- void rotate (const float angle, const vec3 &axis)

  Rotates the object by a specified angle around a given axis.
- void set_scale (const vec3 &scale)

  Sets the scale of the object.
- vec3 get_scale () const

  Sets the scale of the object uniformly.
- bool is_dirty () const

  Checks if the transformation matrix is dirty (needs recalculation).

Protected Attributes

- mesh_info_t mesh_info

  Information about the mesh, including coordinates and rendering type.
- uint16_t color = 0xFFFF

  Color of the mesh, default is white (0xFFFF).
- std::vector< glm::fvec4 > vertices

  Vector of vertices representing the mesh in 3D space.
- std::vector< face_t > faces

  Vector of faces representing the mesh, each face can be a triangle or a quad.
- int slices = 16

  Number of slices for generating the mesh, default is 16.

Protected Attributes inherited from Ragot::Component

- std::vector< std::shared_ptr< Component > > components

  Collection of components managed by this Component instance.

Protected Attributes inherited from Ragot::Node

- std::vector< std::shared_ptr< Node > > children

  List of child nodes.
- Node ∗ parent = nullptr

  Pointer to the parent node.

Protected Attributes inherited from Ragot::Transform

- vec3 position

  The position of the object in 3D space.
- vec3 rotation

  The rotation of the object in degrees around each axis (x, y, z).
- vec3 scale

  The scale of the object in 3D space, default is (1, 1, 1).
- bool dirty = true

  Flag indicating whether the transformation matrix needs to be recalculated.

6.14.1 Detailed Description

Represents a 3D mesh in the Ragot engine.

The Mesh class is a base class for creating 3D meshes with vertices and faces. It provides methods to generate vertices and faces, apply transformations, and manage mesh information. The class also includes methods for setting and getting the color of the mesh.

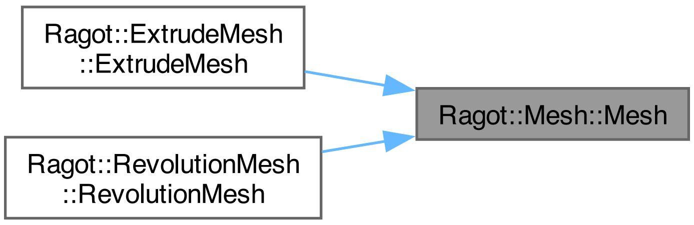
6.14.2 Constructor & Destructor Documentation

6.14.2.1 Mesh() [1/2]

```
Ragot::Mesh::Mesh ()  [delete]
```

Construct a new Mesh object (deleted constructor).

Here is the caller graph for this function:


Ragot::ExtrudeMesh ::ExtrudeMesh
Ragot::RevolutionMesh ::RevolutionMesh
Ragot::Mesh::Mesh

6.14.2.2 ∼Mesh()

```
virtual Ragot::Mesh::∼Mesh () [virtual], [default]
```

Default virtual destructor for the Mesh class.

Cleans up the mesh and its resources.

6.14.2.3 Mesh() [2/2]

```
Ragot::Mesh::Mesh (
            mesh_info_t & mesh_info)
```

Construct a new Mesh object with mesh information.

Initializes the mesh with the provided mesh information.

Parameters

mesh_info	Information about the mesh, including coordinates and rendering type.

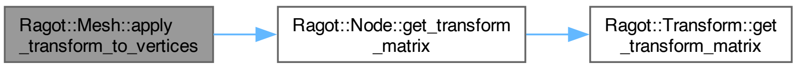
6.14.3 Member Function Documentation

6.14.3.1 apply_transform_to_vertices()

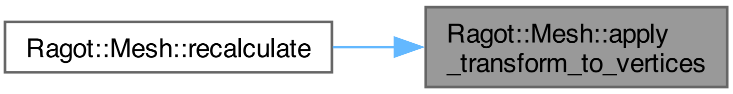
```
void Ragot::Mesh::apply_transform_to_vertices () [inline]
```

Apply the current transformation to the vertices of the mesh. This method applies the transformation matrix obtained from the Transform class to each vertex in the mesh. This is useful for updating the mesh vertices after any transformation has been applied, such as translation, rotation, or scaling. It modifies the vertices in place, transforming them according to the current transformation matrix.

Here is the call graph for this function:


Ragot::Mesh::apply_transform_to_vertices
Ragot::Node::get_transform_matrix
Ragot::Transform::get_transform_matrix

Here is the caller graph for this function:



6.14.3.2 generate_faces()

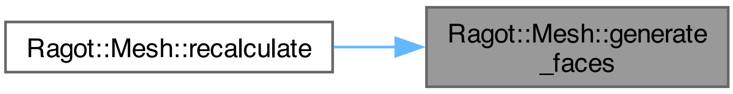
```
virtual void Ragot::Mesh::generate_faces ()  [pure virtual]
```

Generate faces for the mesh.

This method is pure virtual and must be implemented by derived classes. It is responsible for generating the faces of the mesh based on the vertices generated by generate_vertices.

Implemented in Ragot::ExtrudeMesh, and Ragot::RevolutionMesh.

Here is the caller graph for this function:



6.14.3.3 generate_vertices()

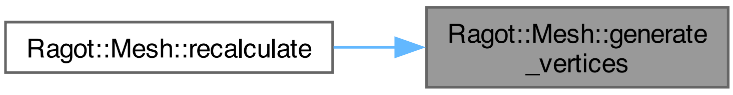
```
virtual void Ragot::Mesh::generate_vertices ()  [pure virtual]
```

Generate vertices for the mesh.

This method is pure virtual and must be implemented by derived classes. It is responsible for generating the vertices of the mesh based on the mesh information.

Implemented in Ragot::ExtrudeMesh, and Ragot::RevolutionMesh.

Here is the caller graph for this function:



6.14.3.4 get_color()

```
uint16_t Ragot::Mesh::get_color () const  [inline]
```

Get the color of the mesh.

This method returns the current color of the mesh.

Returns

uint16_t The color of the mesh.

6.14.3.5 get_faces()

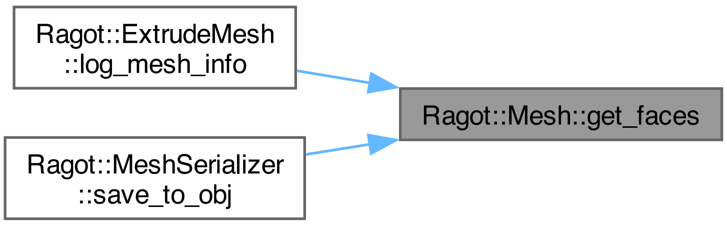
```
const std::vector< face_t > & Ragot::Mesh::get_faces () const  [inline]
```

Get the faces object.

Returns

const std::vector < face_t >&

Here is the caller graph for this function:


Ragot::ExtrudeMesh ::log_mesh_info
Ragot::MeshSerializer ::save_to_obj
Ragot::Mesh::get_faces

6.14.3.6 get_total_vertices() [1/2]

```
size_t Ragot::Mesh::get_total_vertices ()  [inline]
```

Get the total vertices object.

Returns

size_t

6.14.3.7 get_total_vertices() [2/2]

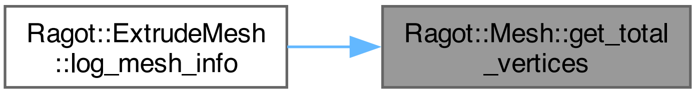
```
const size_t Ragot::Mesh::get_total_vertices () const  [inline]
```

Get the total vertices object.

Returns

const size_t

Here is the caller graph for this function:


Ragot::ExtrudeMesh ::log_mesh_info
Ragot::Mesh::get_total _vertices

6.14.3.8 get_vertices()

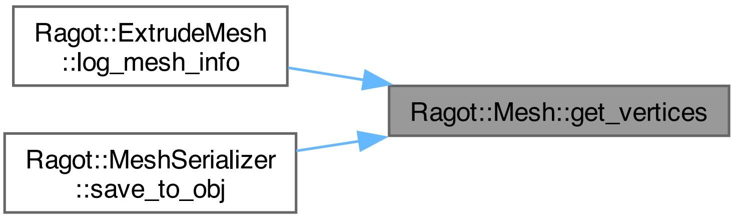
```
const std::vector< glm::fvec4 > & Ragot::Mesh::get_vertices () const  [inline]
```

Get the vertices object.

Returns

const std::vector $<$ glm::fvec4 $>$&

Here is the caller graph for this function:

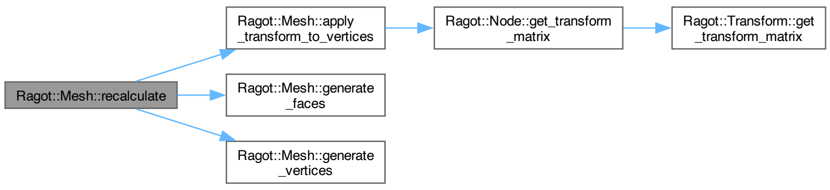

Ragot::ExtrudeMesh ::log_mesh_info
Ragot::MeshSerializer ::save_to_obj
Ragot::Mesh::get_vertices

6.14.3.9 recalculate()

```
void Ragot::Mesh::recalculate ()  [inline]
```

Recalculate the mesh vertices and faces.

Here is the call graph for this function:


Ragot::Mesh::recalculate
Ragot::Mesh::apply_transform_to_vertices
Ragot::Mesh::generate_faces
Ragot::Mesh::generate_vertices
Ragot::Node::get_transform_matrix
Ragot::Transform::get_transform_matrix

6.14.3.10 set_color()

```
void Ragot::Mesh::set_color (
             uint16_t new_color)  [inline]
```

Set the color of the mesh.

This method sets the color of the mesh to the specified new color.

Parameters

new_color	The new color to set for the mesh.

6.14.4 Member Data Documentation

6.14.4.1 color

```
uint16_t Ragot::Mesh::color = 0xFFFF  [protected]
```

Color of the mesh, default is white (0xFFFF).

6.14.4.2 faces

```
std::vector< face_t > Ragot::Mesh::faces  [protected]
```

Vector of faces representing the mesh, each face can be a triangle or a quad.

6.14.4.3 mesh_info

```
mesh_info_t Ragot::Mesh::mesh_info [protected]
```

Information about the mesh, including coordinates and rendering type.

6.14.4.4 slices

```
int Ragot::Mesh::slices = 16 [protected]
```

Number of slices for generating the mesh, default is 16.

6.14.4.5 vertices

```
std::vector< glm::fvec4 > Ragot::Mesh::vertices [protected]
```

Vector of vertices representing the mesh in 3D space.

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

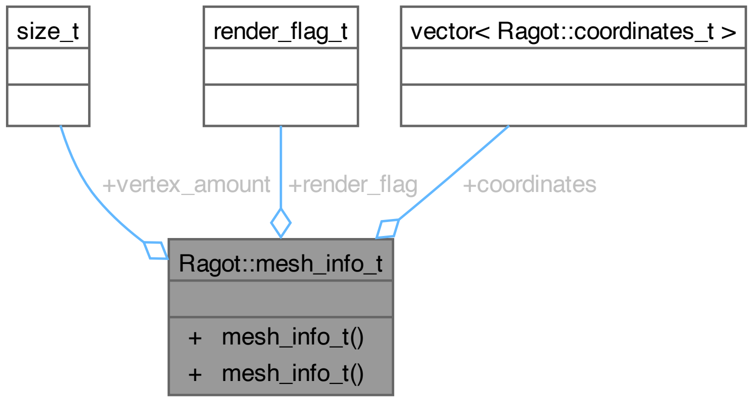
- main/Mesh.hpp
- main/Mesh.cpp

6.15 Ragot::mesh_info_t Struct Reference

Represents information about a mesh.

```
#include <CommonTypes.hpp>
```

Collaboration diagram for Ragot::mesh_info_t:



Public Member Functions

- mesh_info_t ()=default
- mesh_info_t (std::vector< coordinates_t > &coords, render_flag_t flag)

Public Attributes

- size_t vertex_amount = 0
- render_flag_t render_flag = RENDER_NONE
- std::vector< coordinates_t > coordinates

6.15.1 Detailed Description

Represents information about a mesh.

This structure holds the number of vertices, rendering flags, and a vector of coordinates for a mesh.

6.15.2 Constructor & Destructor Documentation

6.15.2.1 mesh_info_t() [1/2]

```
Ragot::mesh_info_t::mesh_info_t ()  [default]
```

6.15.2.2 mesh_info_t() [2/2]

```
Ragot::mesh_info_t::mesh_info_t (
            std::vector< coordinates_t > & coords,
            render_flag_t flag)  [inline]
```

6.15.3 Member Data Documentation

6.15.3.1 coordinates

```
std::vector< coordinates_t > Ragot::mesh_info_t::coordinates
```

6.15.3.2 render_flag

```
render_flag_t Ragot::mesh_info_t::render_flag = RENDER_NONE
```

6.15.3.3 vertex_amount

```
size_t Ragot::mesh_info_t::vertex_amount = 0
```

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

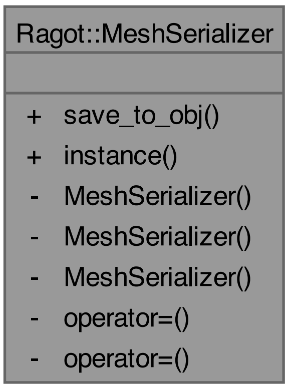
- main/CommonTypes.hpp

6.16 Ragot::MeshSerializer Class Reference

Singleton class to serialize Mesh objects to OBJ file format.

```
#include <MeshSerializer.hpp>
```

Collaboration diagram for Ragot::MeshSerializer:



Public Member Functions

- bool save_to_obj (const Mesh &mesh, const std::filesystem::path &path)

  Saves a Mesh object to an OBJ file.

Static Public Member Functions

- static MeshSerializer & instance ()

  Gets the singleton instance of the MeshSerializer class.

Private Member Functions

- MeshSerializer ()=default

  Private constructor to prevent instantiation from outside the class.
- MeshSerializer (const MeshSerializer &)=delete

  Construct a new Mesh Serializer object (deleted).
- MeshSerializer (const MeshSerializer &&)=delete

  Construct a new Mesh Serializer object (deleted).
- MeshSerializer & operator= (const MeshSerializer &)=delete

  Assignment operator for the MeshSerializer class (deleted).
- MeshSerializer & operator= (const MeshSerializer &&)=delete

  Assignment operator for the MeshSerializer class (deleted).

6.16.1 Detailed Description

Singleton class to serialize Mesh objects to OBJ file format.

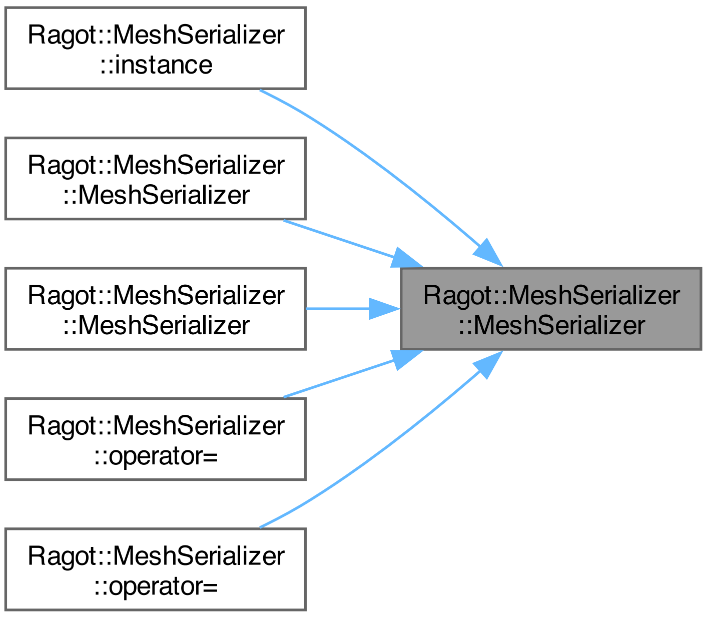
6.16.2 Constructor & Destructor Documentation

6.16.2.1 MeshSerializer() [1/3]

```
Ragot::MeshSerializer::MeshSerializer ()  [private], [default]
```

Private constructor to prevent instantiation from outside the class.

This constructor is private to enforce the singleton pattern, ensuring that only one instance of MeshSerializer exists. Here is the caller graph for this function:

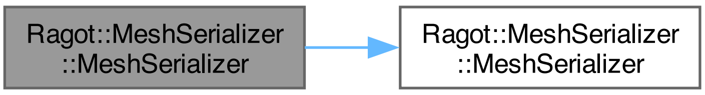

Ragot::MeshSerializer ::instance
Ragot::MeshSerializer ::MeshSerializer
Ragot::MeshSerializer ::MeshSerializer
Ragot::MeshSerializer ::operator=
Ragot::MeshSerializer ::operator=
Ragot::MeshSerializer ::MeshSerializer

6.16.2.2 MeshSerializer() [2/3]

```
Ragot::MeshSerializer::MeshSerializer (
            const MeshSerializer & )  [private], [delete]
```

Construct a new Mesh Serializer object (deleted).

Here is the call graph for this function:

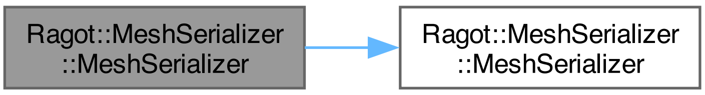

Ragot::MeshSerializer ::MeshSerializer
Ragot::MeshSerializer ::MeshSerializer

6.16.2.3 MeshSerializer() `[3/3]`

```
Ragot::MeshSerializer::MeshSerializer (
            const MeshSerializer && )  [private], [delete]
```

Construct a new Mesh Serializer object (deleted).

This constructor is deleted to prevent moving the MeshSerializer instance. Here is the call graph for this function:


Ragot::MeshSerializer ::MeshSerializer
Ragot::MeshSerializer ::MeshSerializer

6.16.3 Member Function Documentation

6.16.3.1 instance()

```
static MeshSerializer & Ragot::MeshSerializer::instance ()  [inline], [static]
```

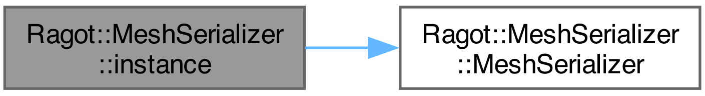
Gets the singleton instance of the MeshSerializer class.

This method ensures that only one instance of MeshSerializer exists throughout the application.

Returns

MeshSerializer& Reference to the singleton instance of MeshSerializer.

Here is the call graph for this function:


Ragot::MeshSerializer ::instance
Ragot::MeshSerializer ::MeshSerializer

6.16.3.2 operator=() [1/2]

```
MeshSerializer & Ragot::MeshSerializer::operator= (
             const MeshSerializer && )  [private], [delete]
```

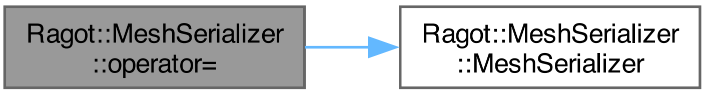
Assignment operator for the MeshSerializer class (deleted).

This operator is deleted to prevent moving MeshSerializer instances.

Returns

MeshSerializer& Reference to the current object.

Here is the call graph for this function:


Ragot::MeshSerializer ::operator=
Ragot::MeshSerializer ::MeshSerializer

6.16.3.3 operator=() [2/2]

```
MeshSerializer & Ragot::MeshSerializer::operator= (
             const MeshSerializer & )  [private], [delete]
```

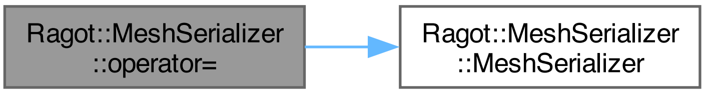
Assignment operator for the MeshSerializer class (deleted).

This operator is deleted to prevent assignment of MeshSerializer instances.

Returns

MeshSerializer& Reference to the current object.

Here is the call graph for this function:


Ragot::MeshSerializer ::operator=
Ragot::MeshSerializer ::MeshSerializer

6.16.3.4 save_to_obj()

```
bool Ragot::MeshSerializer::save_to_obj (
             const Mesh & mesh,
             const std::filesystem::path & path)
```

Saves a Mesh object to an OBJ file.

This method serializes the vertices and faces of the Mesh object and writes them to the specified OBJ file path.

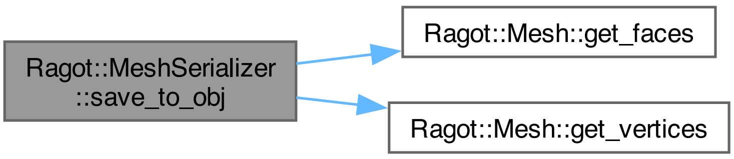
Parameters

mesh	The Mesh object to serialize.
path	The filesystem path where the OBJ file will be saved.

Returns

true if the serialization was successful, false otherwise.

Here is the call graph for this function:


Ragot::MeshSerializer ::save_to_obj
Ragot::Mesh::get_faces
Ragot::Mesh::get_vertices

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

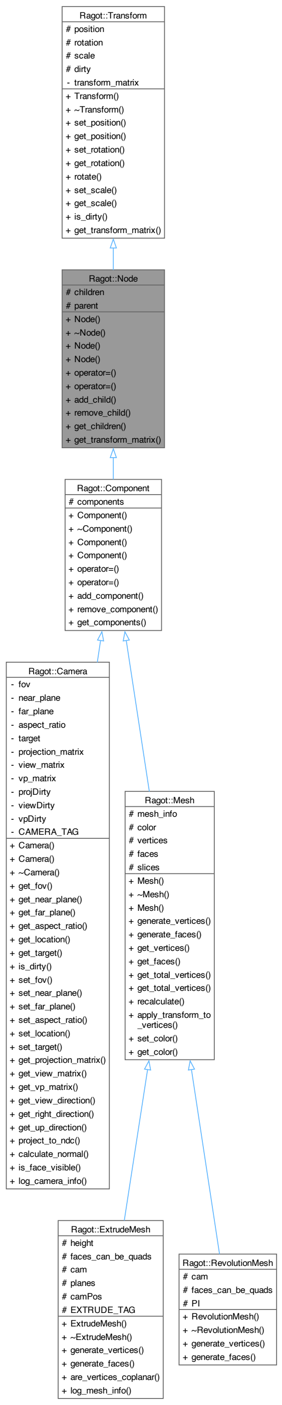
- main/MeshSerializer.hpp
- main/MeshSerializer.cpp

6.17 Ragot::Node Class Reference

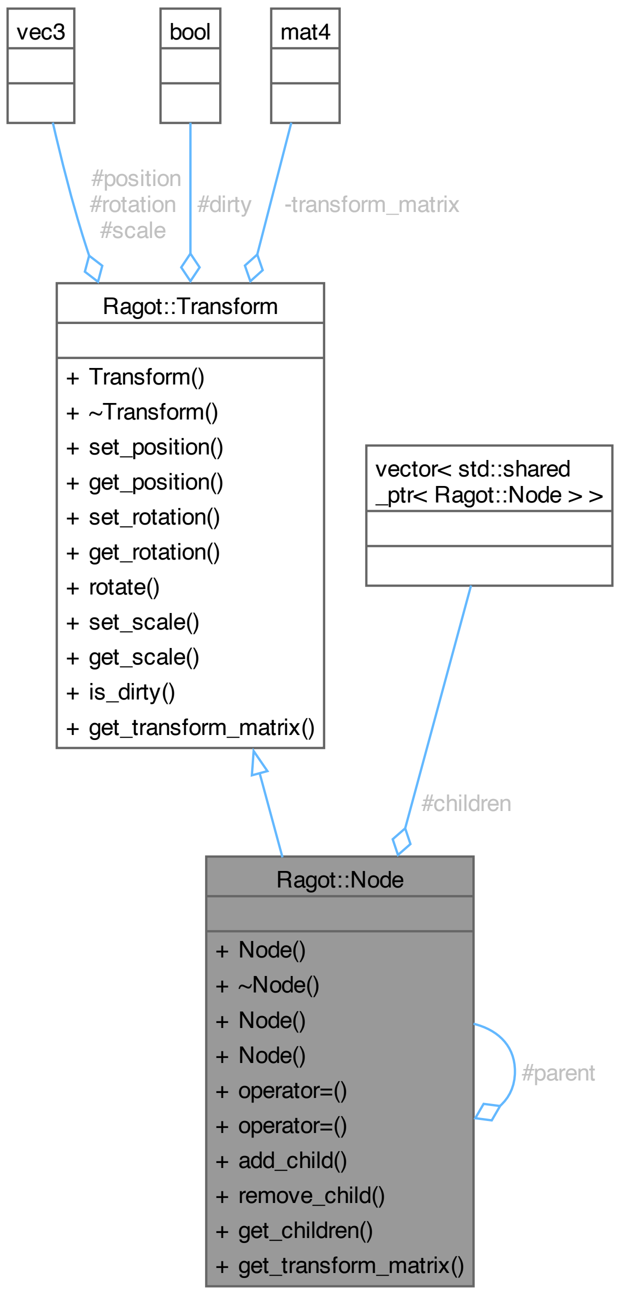
Represents a node in a scene graph for 3D rendering.

```
#include <Node.hpp>
```

Inheritance diagram for Ragot::Node:



Collaboration diagram for Ragot::Node:



Public Member Functions

- Node ()=default

  Default constructor for Node. Initializes an empty node with no parent and no children.
- virtual ~Node ()=default

  Default destructor for Node. Cleans up the node and its children.
- Node (const Node &)=delete

Deleted copy constructor for Node. Prevents copying of Node instances.

- Node (const Node &&)=delete

  Deleted move constructor for Node. Prevents moving of Node instances.
- Node & operator= (const Node &)=delete

  Deleted assignment operator for Node. Prevents assignment of Node instances.
- Node & operator= (const Node &&)=delete

  Deleted move assignment operator for Node. Prevents moving of Node instances.
- void add_child (std::shared_ptr< Node > child)

  Get the parent node.
- void remove_child (std::shared_ptr< Node > child)

  Remove a child node.
- const std::vector< std::shared_ptr< Node > > & get_children () const

  Get the parent node.
- mat4 get_transform_matrix () override

  Get the transform matrix object.

Public Member Functions inherited from Ragot::Transform

- Transform ()

  Default constructor for the Transform class.
- virtual ∼Transform ()=default

  Virtual destructor for the Transform class.
- void set_position (const vec3 &pos)

  Sets the position of the object.
- vec3 get_position () const

  Gets the current position of the object.
- void set_rotation (const vec3 &rot)

  Moves the object by a specified vector.
- vec3 get_rotation () const

  Gets the current rotation of the object.
- void rotate (const float angle, const vec3 &axis)

  Rotates the object by a specified angle around a given axis.
- void set_scale (const vec3 &scale)

  Sets the scale of the object.
- vec3 get_scale () const

  Sets the scale of the object uniformly.
- bool is_dirty () const

  Checks if the transformation matrix is dirty (needs recalculation).

Protected Attributes

- std::vector< std::shared_ptr< Node > > children

  List of child nodes.
- Node ∗ parent = nullptr

  Pointer to the parent node.

Protected Attributes inherited from Ragot::Transform

- vec3 position

    The position of the object in 3D space.
- vec3 rotation

    The rotation of the object in degrees around each axis (x, y, z).
- vec3 scale

    The scale of the object in 3D space, default is (1, 1, 1).
- bool dirty = true

    Flag indicating whether the transformation matrix needs to be recalculated.

6.17.1 Detailed Description

Represents a node in a scene graph for 3D rendering.

The Node class extends the Transform class to include child nodes, allowing for hierarchical transformations and management of child nodes.

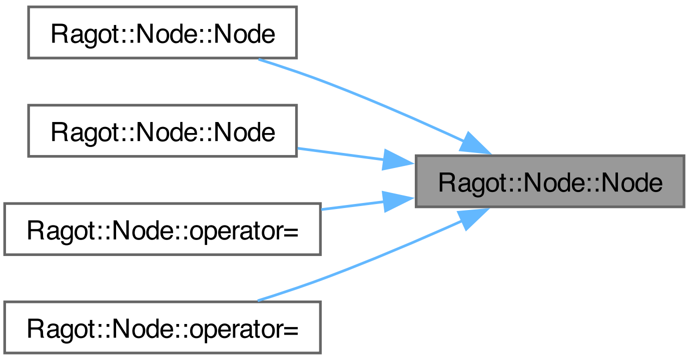
6.17.2 Constructor & Destructor Documentation

6.17.2.1 Node() [1/3]

```
Ragot::Node::Node ()  [default]
```

Default constructor for Node. Initializes an empty node with no parent and no children.

Here is the caller graph for this function:



6.17.2.2 ∼Node()

```
virtual Ragot::Node::∼Node ()  [virtual], [default]
```

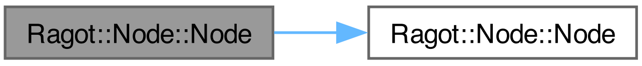
Default destructor for Node. Cleans up the node and its children.

6.17.2.3 Node() [2/3]

```
Ragot::Node::Node (
             const Node & )  [delete]
```

Deleted copy constructor for Node. Prevents copying of Node instances.

Here is the call graph for this function:

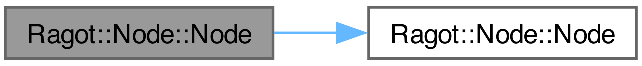


6.17.2.4 Node() [3/3]

```
Ragot::Node::Node (
             const Node && )  [delete]
```

Deleted move constructor for Node. Prevents moving of Node instances.

Here is the call graph for this function:



6.17.3 Member Function Documentation

6.17.3.1 add_child()

```
void Ragot::Node::add_child (
             std::shared_ptr< Node > child)  [inline]
```

Get the parent node.

Returns

Node∗ Pointer to the parent node, or nullptr if no parent exists.

6.17.3.2 get_children()

```
const std::vector< std::shared_ptr< Node > > & Ragot::Node::get_children () const  [inline]
```

Get the parent node.

Returns

Node∗ Pointer to the parent node, or nullptr if no parent exists.

6.17.3.3 get_transform_matrix()

```
mat4 Ragot::Node::get_transform_matrix ()  [inline], [override], [virtual]
```

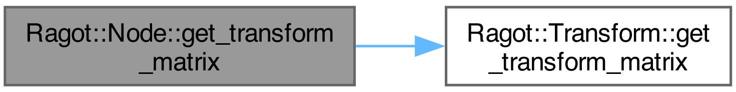
Get the transform matrix object.

Returns

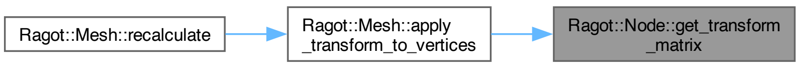
mat4 The transformation matrix of the node, including its parent's transformation.

Reimplemented from Ragot::Transform.

Here is the call graph for this function:


Ragot::Node::get_transform _matrix
Ragot::Transform::get _transform_matrix

Here is the caller graph for this function:


Ragot::Mesh::recalculate
Ragot::Mesh::apply _transform_to_vertices
Ragot::Node::get_transform _matrix

6.17.3.4 operator=() [1/2]

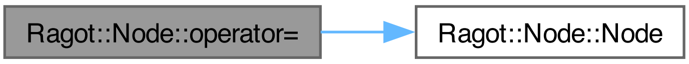
```
Node & Ragot::Node::operator= (
             const Node && )   [delete]
```

Deleted move assignment operator for Node. Prevents moving of Node instances.

Returns

Node& Reference to the current object.

Here is the call graph for this function:

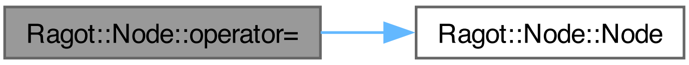


6.17.3.5 operator=() [2/2]

```
Node & Ragot::Node::operator= (
             const Node & )   [delete]
```

Deleted assignment operator for Node. Prevents assignment of Node instances.

Here is the call graph for this function:



6.17.3.6 remove_child()

```
void Ragot::Node::remove_child (
             std::shared_ptr< Node > child)   [inline]
```

Remove a child node.

This method removes the specified child node from the list of children. If the child exists, it is removed and its parent pointer is set to nullptr.

Parameters

child	The child node to remove.

6.17.4 Member Data Documentation

6.17.4.1 children

```
std::vector< std::shared_ptr < Node > > Ragot::Node::children  [protected]
```

List of child nodes.

6.17.4.2 parent

```
Node* Ragot::Node::parent = nullptr  [protected]
```

Pointer to the parent node.

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

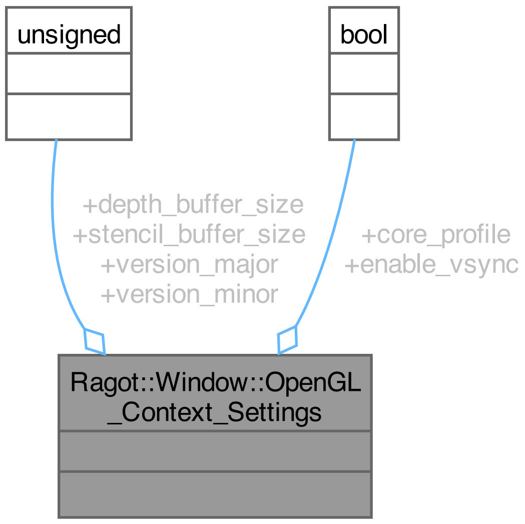
- main/Node.hpp

6.18 Ragot::Window::OpenGL_Context_Settings Struct Reference

Struct for OpenGL context settings.

```
#include <Window.hpp>
```

Collaboration diagram for Ragot::Window::OpenGL_Context_Settings:



Public Attributes

- unsigned version_major = 3

    Major version of OpenGL.
- unsigned version_minor = 3

    Minor version of OpenGL.
- bool core_profile = true

    Core profile flag.
- unsigned depth_buffer_size = 24

    Depth buffer size.
- unsigned stencil_buffer_size = 0

    Stencil buffer size.
- bool enable_vsync = true

    V-Sync enable flag.

6.18.1 Detailed Description

Struct for OpenGL context settings.

6.18.2 Member Data Documentation

6.18.2.1 core_profile

```
bool Ragot::Window::OpenGL_Context_Settings::core_profile = true
```

Core profile flag.

6.18.2.2 depth_buffer_size

```
unsigned Ragot::Window::OpenGL_Context_Settings::depth_buffer_size = 24
```

Depth buffer size.

6.18.2.3 enable_vsync

```
bool Ragot::Window::OpenGL_Context_Settings::enable_vsync = true
```

V-Sync enable flag.

6.18.2.4 stencil_buffer_size

```
unsigned Ragot::Window::OpenGL_Context_Settings::stencil_buffer_size = 0
```

Stencil buffer size.

6.18.2.5 version_major

```
unsigned Ragot::Window::OpenGL_Context_Settings::version_major = 3
```

Major version of OpenGL.

6.18.2.6 version_minor

```
unsigned Ragot::Window::OpenGL_Context_Settings::version_minor = 3
```

Minor version of OpenGL.

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

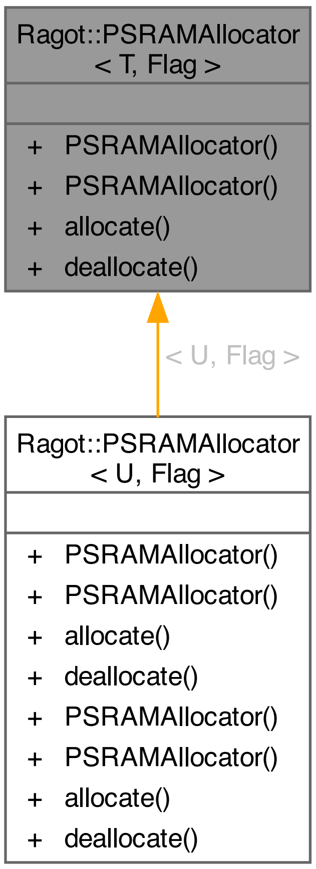
- main/Window.hpp

6.19 Ragot::PSRAMAllocator< T, Flag > Class Template Reference

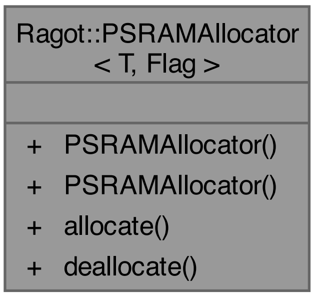
Custom memory allocator for PSRAM.

```
#include <RamAllocator.hpp>
```

Inheritance diagram for Ragot::PSRAMAllocator< T, Flag >:



Collaboration diagram for Ragot::PSRAMAllocator< T, Flag >:



Classes

- struct rebind

  Rebinds the allocator to a different type. This struct allows the PSRAMAllocator to be used with different types while maintaining the same allocation flags.

Public Types

- using value_type = T
- using pointer = T∗
- using size_type = std::size_t

Public Member Functions

- PSRAMAllocator () noexcept

  Default constructor for PSRAMAllocator.
- template<typename U, uint16_t F2>
  PSRAMAllocator (const PSRAMAllocator< U, F2 > &) noexcept

  Copy constructor for PSRAMAllocator.
- T ∗ allocate (size_type n)

  Allocates memory for n objects of type T in PSRAM.
- void deallocate (T ∗p, size_type) noexcept

  Deallocates memory for n objects of type T in PSRAM.

6.19.1 Detailed Description

template<typename T, uint16_t Flag>
class Ragot::PSRAMAllocator< T, Flag >

Custom memory allocator for PSRAM.

This class provides a custom memory allocator that uses PSRAM with specific flags. It can be used with standard containers like std::vector to manage memory in embedded systems.

6.19.2 Member Typedef Documentation

6.19.2.1 pointer

```
template<typename T, uint16_t Flag>
using Ragot::PSRAMAllocator< T, Flag >::pointer = T*
```

6.19.2.2 size_type

```
template<typename T, uint16_t Flag>
using Ragot::PSRAMAllocator< T, Flag >::size_type = std::size_t
```

6.19.2.3 value_type

```
template<typename T, uint16_t Flag>
using Ragot::PSRAMAllocator< T, Flag >::value_type = T
```

6.19.3 Constructor & Destructor Documentation

6.19.3.1 PSRAMAllocator() [1/2]

```
template<typename T, uint16_t Flag>
Ragot::PSRAMAllocator< T, Flag >::PSRAMAllocator ()  [inline], [noexcept]
```

Default constructor for PSRAMAllocator.

This constructor initializes the PSRAMAllocator without any specific parameters.

6.19.3.2 PSRAMAllocator() [2/2]

```
template<typename T, uint16_t Flag>
template<typename U, uint16_t F2>
Ragot::PSRAMAllocator< T, Flag >::PSRAMAllocator (
             const PSRAMAllocator< U, F2 > & )  [inline], [noexcept]
```

Copy constructor for PSRAMAllocator.

This constructor allows copying of the PSRAMAllocator, but it does not perform any specific actions. It is designed to be used with standard containers that require copyable allocators.

Template Parameters

U	The type to rebind to.

Parameters

other	The allocator to copy from.

6.19.4 Member Function Documentation

6.19.4.1 allocate()

```
template<typename T, uint16_t Flag>
T * Ragot::PSRAMAllocator< T, Flag >::allocate (
             size_type n)  [inline]
```

Allocates memory for n objects of type T in PSRAM.

This method allocates memory for n objects of type T using heap_caps_malloc with the specified flags. If the allocation fails, it throws std::bad_alloc.

Parameters

n	The number of objects to allocate memory for.

Returns

T∗ Pointer to the allocated memory.

6.19.4.2 deallocate()

```
template<typename T, uint16_t Flag>
void Ragot::PSRAMAllocator< T, Flag >::deallocate (
             T * p,
             size_type )  [inline], [noexcept]
```

Deallocates memory for n objects of type T in PSRAM.

This method deallocates memory for n objects of type T using heap_caps_free. It does not throw any exceptions.

Parameters

p	Pointer to the memory to deallocate.
n	The number of objects to deallocate (not used).

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

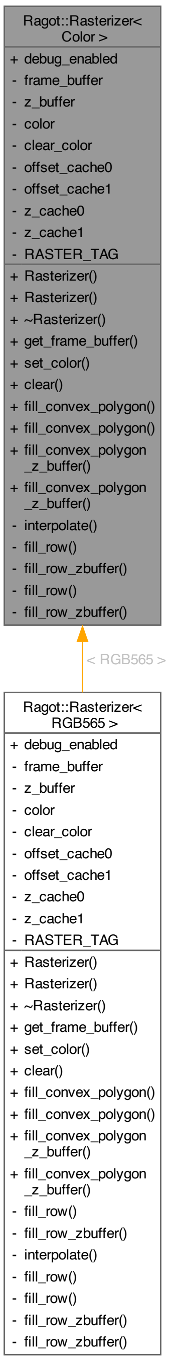
- main/RamAllocator.hpp

6.20 Ragot::Rasterizer< Color > Class Template Reference

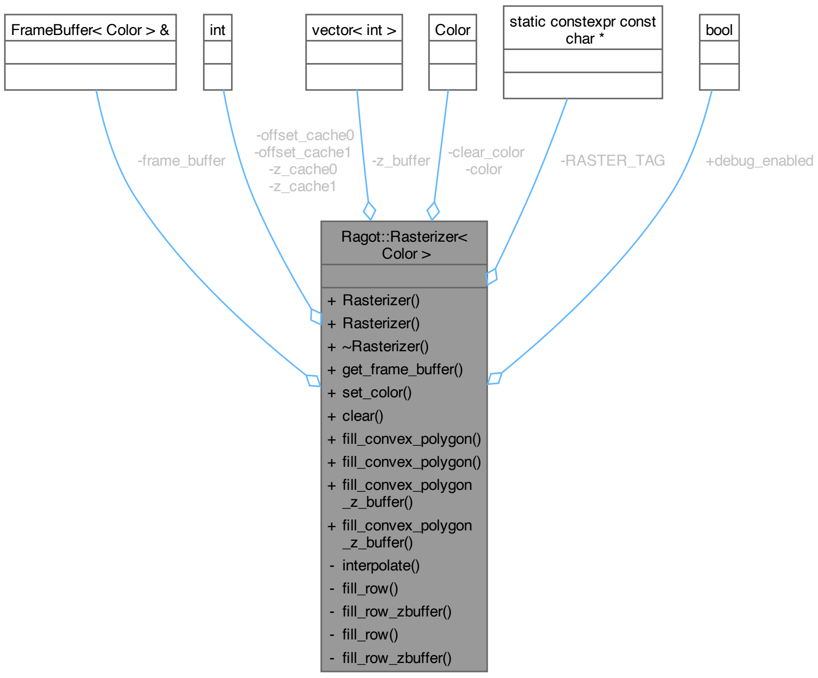
Class for rasterizing polygons in a frame buffer.

`#include <Rasterizer.hpp>`

Inheritance diagram for Ragot::Rasterizer< Color >:



Collaboration diagram for Ragot::Rasterizer< Color >:



Public Member Functions

- Rasterizer (FrameBuffer< Color > &frame)

  Constructs a Rasterizer with a given frame buffer.
- Rasterizer ()=default

  Default constructor for the Rasterizer class (Default).
- ∼Rasterizer ()=default

  Default destructor for the Rasterizer class.
- const FrameBuffer< Color > & get_frame_buffer () const

  Gets the frame buffer associated with this rasterizer.
- void set_color (const Color &new_color)

  Gets the current color used for drawing polygons.
- void clear ()

  Gets the current color used for clearing the frame buffer.
- void fill_convex_polygon (const glm::ivec4 ∗const vertices, const int ∗const indices_begin, const int ∗const indices_end)

  Fills a convex polygon defined by its vertices and indices.
- void fill_convex_polygon (const glm::ivec4 ∗const vertices, const face_t ∗const face)

  Fills a convex polygon defined by its vertices and face structure.
- void fill_convex_polygon_z_buffer (const glm::ivec4 ∗const vertices, const face_t ∗const face)

  Fills a convex polygon in the Z-buffer.
- void fill_convex_polygon_z_buffer (const glm::ivec4 ∗const vertices, const int ∗const indices_begin, const int ∗const indices_end)

  Fills a convex polygon in the Z-buffer using vertex indices.

Public Attributes

- bool debug_enabled = true

Private Member Functions

- template<typename VALUE_TYPE, size_t SHIFT>
  void interpolate (int ∗cache, int v0, int v1, int y_min, int y_max)

  < Enable or disable debug logging for rasterization operations.
- template<unsigned COLOR_SIZE>
  void fill_row (Color ∗start, unsigned left_offset, unsigned right_offset, const Color &color)

  Fills a row of pixels in the frame buffer with a specified color.
- template<unsigned COLOR_SIZE>
  void fill_row_zbuffer (Color ∗start, int ∗zbuffer, unsigned left_offset, unsigned right_offset, int z_start, int dz, const Color &color)

  Fills a row of pixels in the frame buffer with a specified color using Z-buffering.
- void fill_row (RGB565 ∗start, unsigned left_offset, unsigned right_offset, const RGB565 &color)
- void fill_row_zbuffer (RGB565 ∗start, int ∗zbuffer, unsigned left_offset, unsigned right_offset, int z_start, int dz, const RGB565 &color)

Private Attributes

- FrameBuffer< Color > & frame_buffer

  Reference to the frame buffer where polygons will be drawn.
- std::vector< int > z_buffer

  Z-buffer for depth testing, used when painter's algorithm is disabled.
- Color color

  Current color to be used for drawing polygons.
- Color clear_color

  Color used to clear the frame buffer, default is black.

Static Private Attributes

- static int offset_cache0 [1024]

  Cache for left offsets of scanlines >
- static int offset_cache1 [1024]

  Cache for right offsets of scanlines.
- static int z_cache0 [1024]

  Cache for Z-buffer values for left offsets.
- static int z_cache1 [1024]

  Cache for Z-buffer values for right offsets.
- static constexpr const char ∗ RASTER_TAG = "Rasterizer"

6.20.1 Detailed Description

template<typename Color>
class Ragot::Rasterizer< Color >

Class for rasterizing polygons in a frame buffer.

This class provides methods to fill convex polygons in a frame buffer with a specified color. It supports both standard rasterization and Z-buffering techniques.

Template Parameters

Color	The color type used for the frame buffer (e.g., RGB565).

6.20.2 Constructor & Destructor Documentation

6.20.2.1 Rasterizer() `[1/2]`

```
template<typename Color>
Ragot::Rasterizer< Color >::Rasterizer (
            FrameBuffer< Color > & frame)  [inline]
```

Constructs a Rasterizer with a given frame buffer.

Initializes the rasterizer with the specified frame buffer and prepares the Z-buffer if needed.

Parameters

frame	Reference to the frame buffer where polygons will be drawn.

6.20.2.2 Rasterizer() `[2/2]`

```
template<typename Color>
Ragot::Rasterizer< Color >::Rasterizer ()  [default]
```

Default constructor for the Rasterizer class (Default).

6.20.2.3 ∼Rasterizer()

```
template<typename Color>
Ragot::Rasterizer< Color >::∼Rasterizer ()  [default]
```

Default destructor for the Rasterizer class.

Cleans up resources used by the rasterizer.

6.20.3 Member Function Documentation

6.20.3.1 clear()

```
template<typename Color>
void Ragot::Rasterizer< Color >::clear ()  [inline]
```

Gets the current color used for clearing the frame buffer.

This method returns the color that will be used to clear the frame buffer.

Returns

const Color& Reference to the clear color.

6.20.3.2 fill_convex_polygon() [1/2]

```
template<typename COLOR>
void Ragot::Rasterizer< COLOR >::fill_convex_polygon (
            const glm::ivec4 *const vertices,
            const face_t *const face)
```

Fills a convex polygon defined by its vertices and face structure.

This method fills a convex polygon in the frame buffer using the specified vertices and face structure.

Parameters

vertices	Pointer to an array of vertices defining the polygon.
face	Pointer to the face structure containing vertex indices.

6.20.3.3 fill_convex_polygon() [2/2]

```
template<typename COLOR>
void Ragot::Rasterizer< COLOR >::fill_convex_polygon (
            const glm::ivec4 *const vertices,
            const int *const indices_begin,
            const int *const indices_end)
```

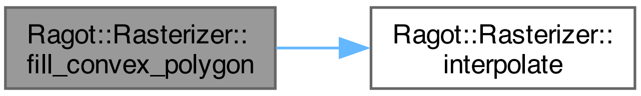
Fills a convex polygon defined by its vertices and indices.

This method fills a convex polygon in the frame buffer using the specified vertices and indices.

Parameters

vertices	Pointer to an array of vertices defining the polygon.
indices_begin	Pointer to the beginning of the indices array.
indices_end	Pointer to the end of the indices array.

Here is the call graph for this function:



6.20.3.4 fill_convex_polygon_z_buffer() [1/2]

```
template<typename COLOR>
template void Ragot::Rasterizer< Color >::fill_convex_polygon_z_buffer (
            const glm::ivec4 *const vertices,
            const face_t *const face)
```

Fills a convex polygon in the Z-buffer.

This method fills a convex polygon in the Z-buffer using the specified vertices and face structure. It performs depth testing to ensure correct rendering order.

Parameters

vertices	Pointer to an array of vertices defining the polygon.
face	Pointer to the face structure containing vertex indices.

6.20.3.5 fill_convex_polygon_z_buffer() [2/2]

```
template<typename COLOR>
template void Ragot::Rasterizer< Color >::fill_convex_polygon_z_buffer (
            const glm::ivec4 *const vertices,
            const int *const indices_begin,
            const int *const indices_end)
```

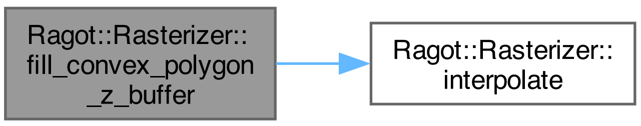
Fills a convex polygon in the Z-buffer using vertex indices.

- This method fills a convex polygon in the Z-buffer using the specified vertices and indices. It performs depth testing to ensure correct rendering order.

Parameters

vertices	Pointer to an array of vertices defining the polygon.
indices_begin	Pointer to the beginning of the indices array.
indices_end	Pointer to the end of the indices array.

Here is the call graph for this function:



6.20.3.6 fill_row() [1/2]

```
template<typename Color>
template<unsigned COLOR_SIZE>
void Ragot::Rasterizer< Color >::fill_row (
            Color * start,
            unsigned left_offset,
            unsigned right_offset,
            const Color & color)  [inline], [private]
```

Fills a row of pixels in the frame buffer with a specified color.

This method fills a row of pixels in the frame buffer with the specified color, from left_offset to right_offset.

Template Parameters

COLOR_SIZE	The size of the color type in bytes.

Parameters

start	Pointer to the first pixel of the scanline.
left_offset	The starting offset (inclusive).
right_offset	The ending offset (exclusive).
color	The color to fill the row with.

6.20.3.7 fill_row() [2/2]

```
void Ragot::Rasterizer< RGB565 >::fill_row< 2 > (
            RGB565 * start,
            unsigned left_offset,
            unsigned right_offset,
            const RGB565 & color)  [private]
```

6.20.3.8 fill_row_zbuffer() [1/2]

```
template<typename Color>
template<unsigned COLOR_SIZE>
void Ragot::Rasterizer< Color >::fill_row_zbuffer (
            Color * start,
            int * zbuffer,
            unsigned left_offset,
            unsigned right_offset,
            int z_start,
            int dz,
            const Color & color)  [inline], [private]
```

Fills a row of pixels in the frame buffer with a specified color using Z-buffering.

This method fills a row of pixels in the frame buffer with the specified color, from left_offset to right_offset, while performing depth testing using the Z-buffer.

Template Parameters

COLOR_SIZE	The size of the color type in bytes.

Parameters

start	Pointer to the first pixel of the scanline.
zbuffer	Pointer to the first element of the Z-buffer.
left_offset	The starting offset (inclusive).
right_offset	The ending offset (exclusive).
z_start	The initial depth value at left_offset.
dz	The increment of depth per pixel.
color	The color to fill the row with.

6.20.3.9 fill_row_zbuffer() [2/2]

```
void Ragot::Rasterizer< RGB565 >::fill_row_zbuffer< 2 > (
            RGB565 * start,
            int * zbuffer,
            unsigned left_offset,
            unsigned right_offset,
            int z_start,
            int dz,
            const RGB565 & color)  [private]
```

6.20.3.10 get_frame_buffer()

```
template<typename Color>
const FrameBuffer< Color > & Ragot::Rasterizer< Color >::get_frame_buffer () const  [inline]
```

Gets the frame buffer associated with this rasterizer.

This method returns a reference to the frame buffer where polygons will be drawn.

Returns

const FrameBuffer<Color>& Reference to the frame buffer.

6.20.3.11 interpolate()

```
template<typename COLOR>
template<typename VALUE_TYPE, size_t SHIFT>
void Ragot::Rasterizer< COLOR >::interpolate (
            int * cache,
            int v0,
            int v1,
            int y_min,
            int y_max)  [private]
```

< Enable or disable debug logging for rasterization operations.

Interpolates pixel offsets for a scanline.

This method interpolates pixel offsets for a scanline based on the provided vertex values and Y range. It fills the offset cache with calculated pixel offsets.

Template Parameters

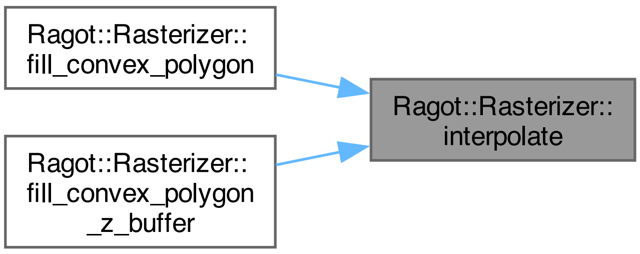
VALUE_TYPE	The type of value used for interpolation (e.g., int32_t).
SHIFT	The bit shift value used for scaling the interpolation.

Parameters

cache	Pointer to the cache where interpolated offsets will be stored.
v0	The starting vertex value.
v1	The ending vertex value.

y_min	The minimum Y coordinate of the scanline.
y_max	The maximum Y coordinate of the scanline.

Here is the caller graph for this function:


Ragot::Rasterizer:: fill_convex_polygon
Ragot::Rasterizer:: fill_convex_polygon _z_buffer
Ragot::Rasterizer:: interpolate

6.20.3.12 set_color()

```
template<typename Color>
void Ragot::Rasterizer< Color >::set_color (
             const Color & new_color)  [inline]
```

Gets the current color used for drawing polygons.

This method returns the current color that will be used to fill polygons.

Returns

const Color& Reference to the current color.

6.20.4 Member Data Documentation

6.20.4.1 clear_color

```
template<typename Color>
Color Ragot::Rasterizer< Color >::clear_color  [private]
```

Color used to clear the frame buffer, default is black.

6.20.4.2 color

```
template<typename Color>
Color Ragot::Rasterizer< Color >::color  [private]
```

Current color to be used for drawing polygons.

6.20.4.3 debug_enabled

```
template<typename Color>
bool Ragot::Rasterizer< Color >::debug_enabled = true
```

6.20.4.4 frame_buffer

```
template<typename Color>
FrameBuffer< Color >& Ragot::Rasterizer< Color >::frame_buffer  [private]
```

Reference to the frame buffer where polygons will be drawn.

6.20.4.5 offset_cache0

```
template<typename Color>
int Ragot::Rasterizer< Color >::offset_cache0[1024]  [static], [private]
```

Cache for left offsets of scanlines >

6.20.4.6 offset_cache1

```
template<typename Color>
int Ragot::Rasterizer< Color >::offset_cache1[1024]  [static], [private]
```

Cache for right offsets of scanlines.

6.20.4.7 RASTER_TAG

```
template<typename Color>
const char* Ragot::Rasterizer< Color >::RASTER_TAG = "Rasterizer"  [static], [constexpr],
[private]
```

6.20.4.8 z_buffer

```
template<typename Color>
std::vector< int > Ragot::Rasterizer< Color >::z_buffer  [private]
```

Z-buffer for depth testing, used when painter's algorithm is disabled.

6.20.4.9 z_cache0

```
template<typename Color>
int Ragot::Rasterizer< Color >::z_cache0[1024]  [static], [private]
```

Cache for Z-buffer values for left offsets.

6.20.4.10 z_cache1

```
template<typename Color>
int Ragot::Rasterizer< Color >::z_cache1[1024]  [static], [private]
```

Cache for Z-buffer values for right offsets.

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

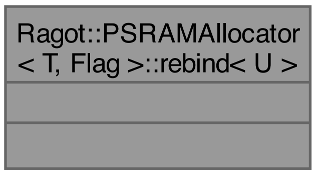
- main/Rasterizer.hpp
- main/Rasterizer.cpp

6.21 Ragot::PSRAMAllocator< T, Flag >::rebind< U > Struct Template Reference

Rebinds the allocator to a different type. This struct allows the PSRAMAllocator to be used with different types while maintaining the same allocation flags.

```
#include <RamAllocator.hpp>
```

Collaboration diagram for Ragot::PSRAMAllocator< T, Flag >::rebind< U >:



Public Types

- using other = PSRAMAllocator<U, Flag>

6.21.1 Detailed Description

template<typename T, uint16_t Flag>
template<typename U>
struct Ragot::PSRAMAllocator< T, Flag >::rebind< U >

Rebinds the allocator to a different type. This struct allows the PSRAMAllocator to be used with different types while maintaining the same allocation flags.

Template Parameters

U	

6.21.2 Member Typedef Documentation

6.21.2.1 other

```
template<typename T, uint16_t Flag>
template<typename U>
using Ragot::PSRAMAllocator< T, Flag >::rebind< U >::other = PSRAMAllocator<U, Flag>
```

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

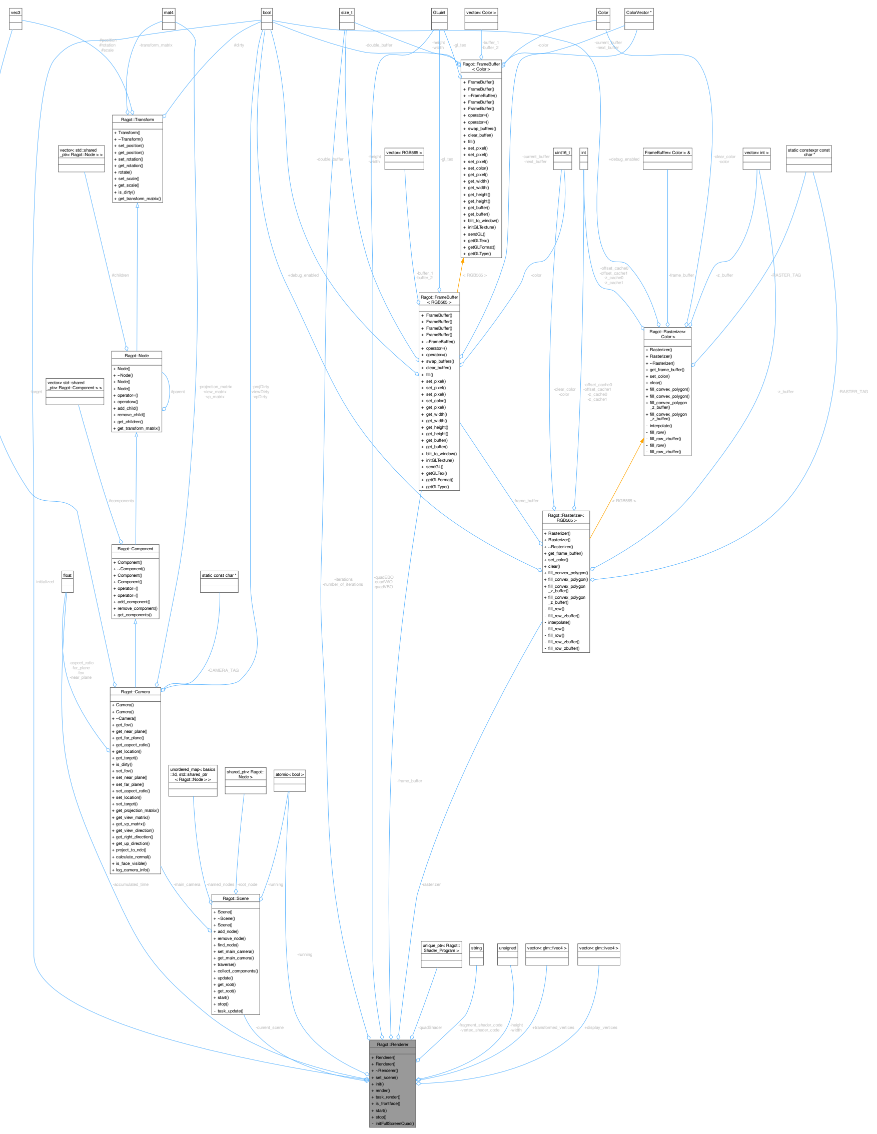
- main/RamAllocator.hpp

6.22 Ragot::Renderer Class Reference

Class for rendering scenes in the Ragot engine.

```
#include <Renderer.hpp>
```

Collaboration diagram for Ragot::Renderer:



Public Member Functions

- Renderer ()=delete

  Construct a new Renderer object (Deleted).
- Renderer (unsigned width, unsigned height)

  Constructs a Renderer with the specified width and height.
- ∼Renderer ()=default

  Default destructor for the Renderer class.
- void set_scene (Scene ∗scene)

  Sets the current scene to be rendered.
- void init ()

  Gets the current scene being rendered.
- void render ()

  Renders the current scene.
- void task_render (std::stop_token stop_token)

Performs the rendering task in a separate thread.
- bool is_frontface (const glm::fvec4 ∗const projected_vertices, const face_t ∗const indices)

  Checks if the face defined by the indices is front-facing.
- void start ()

  Starts the rendering process.
- void stop ()

  Stops the rendering process.

Public Attributes

- std::vector< glm::fvec4 > transformed_vertices

  Vector to store transformed vertices for rendering, used to hold the vertices after applying transformations such as model, view, and projection matrices.
- std::vector< glm::ivec4 > display_vertices

  Vector to store display vertices for rendering, used to hold the vertices after applying viewport transformations and clipping.

Private Member Functions

- void initFullScreenQuad ()

  Initializes the full-screen quad for rendering, used in non-ESP platforms.

Private Attributes

- float accumulated_time = 0.f

  Accumulated time for rendering frames, used for timing and performance measurement.
- size_t iterations = 0

  Number of iterations for rendering, used for performance testing and optimization.
- std::unique_ptr< Shader_Program > quadShader = nullptr

  Number of iterations for performance testing, can be adjusted for different scenarios.
- GLuint quadVAO = 0

  Vertex Array Object for the full-screen quad, used in non-ESP platforms.
- GLuint quadVBO = 0

  Vertex Buffer Object for the full-screen quad, used in non-ESP platforms.
- GLuint quadEBO = 0

  Element Buffer Object for the full-screen quad, used in non-ESP platforms.
- FrameBuffer< RGB565 > frame_buffer

  Frame buffer for rendering, used to store pixel data for the rendered scene.
- Scene ∗ current_scene = nullptr

  Pointer to the current scene being rendered, allows access to scene data and objects.
- Rasterizer< RGB565 > rasterizer

  Rasterizer for rendering polygons in the frame buffer, responsible for filling polygons with color and handling depth testing.
- unsigned width

  Width of the rendering area in pixels, used to define the size of the frame buffer and viewport.
- unsigned height

  Height of the rendering area in pixels, used to define the size of the frame buffer and viewport.
- bool initialized = false

  Flag to indicate if the renderer has been initialized, used to prevent re-initialization and ensure resources are set up correctly.
- std::atomic< bool > running = false

  Flag to indicate if the renderer is currently running, used to control rendering tasks and stop them gracefully.

Static Private Attributes

- static constexpr size_t number_of_iterations = 10000000000000000
- static const std::string vertex_shader_code

    Vertex shader code for rendering, used in non-ESP platforms.
- static const std::string fragment_shader_code

    Fragment shader code for rendering, used in non-ESP platforms.

6.22.1 Detailed Description

Class for rendering scenes in the Ragot engine.

This class is responsible for rendering 3D scenes using a rasterization approach. It manages the frame buffer, rasterizer, and scene to be rendered.

6.22.2 Constructor & Destructor Documentation

6.22.2.1 Renderer() [1/2]

```
Ragot::Renderer::Renderer ()  [delete]
```

Construct a new Renderer object (Deleted).

This constructor is deleted to prevent default construction of the Renderer class.

6.22.2.2 Renderer() [2/2]

```
Ragot::Renderer::Renderer (
            unsigned width,
            unsigned height)
```

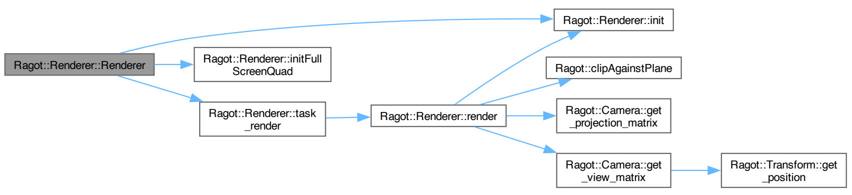
Constructs a Renderer with the specified width and height.

Initializes the renderer with the given dimensions and prepares the frame buffer and rasterizer.

Parameters

width	The width of the rendering area in pixels.
height	The height of the rendering area in pixels.

Here is the call graph for this function:


Ragot::Renderer::Renderer
Ragot::Renderer::init
Ragot::Renderer::initFull ScreenQuad
Ragot::clipAgainstPlane
Ragot::Renderer::task _render
Ragot::Renderer::render
Ragot::Camera::get _projection_matrix
Ragot::Camera::get _view_matrix
Ragot::Transform::get _position

6.22.2.3 ∼Renderer()

```
Ragot::Renderer::∼Renderer ()  [default]
```

Default destructor for the Renderer class.

Cleans up resources used by the renderer.

6.22.3 Member Function Documentation

6.22.3.1 init()

```
void Ragot::Renderer::init ()
```

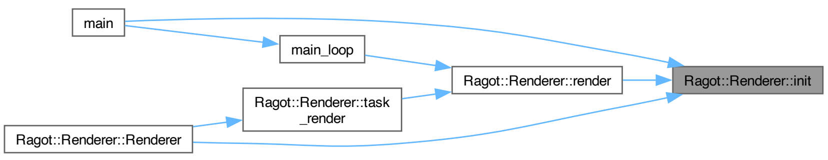
Gets the current scene being rendered.

This method returns a pointer to the current scene being rendered by the renderer. It allows access to the scene's objects and properties for rendering.

Returns

Scene∗ Pointer to the current Scene object.

Here is the caller graph for this function:

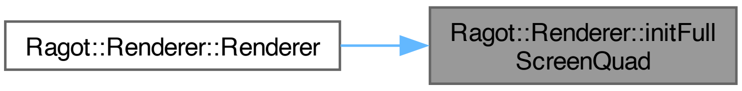


6.22.3.2 initFullScreenQuad()

```
void Ragot::Renderer::initFullScreenQuad ()  [private]
```

Initializes the full-screen quad for rendering, used in non-ESP platforms.

Here is the caller graph for this function:


Ragot::Renderer::Renderer
Ragot::Renderer::initFull ScreenQuad

6.22.3.3 is_frontface()

```
bool Ragot::Renderer::is_frontface (
            const glm::fvec4 *const projected_vertices,
            const face_t *const indices)
```

Checks if the face defined by the indices is front-facing.

This method checks if the face defined by the indices is front-facing based on the projected vertices. It uses the area of the face to determine its orientation.

Parameters

projected_vertices	Pointer to an array of projected vertices in clip space.
indices	Pointer to the face structure containing vertex indices.

Returns

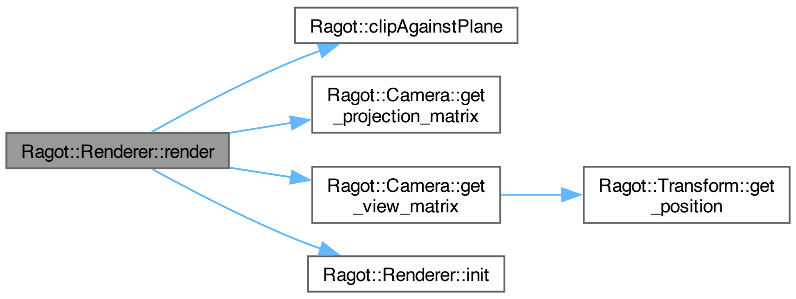
true if the face is front-facing, false otherwise.

6.22.3.4 render()

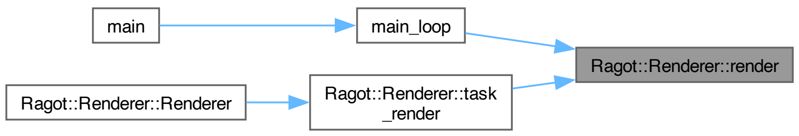
```
void Ragot::Renderer::render ()
```

Renders the current scene.

This method performs the rendering of the current scene by preparing matrices, transforming vertices, and filling polygons in the frame buffer using the rasterizer. It also handles depth testing and color filling for polygons. Here is the call graph for this function:


Ragot::clipAgainstPlane
Ragot::Camera::get_projection_matrix
Ragot::Renderer::render
Ragot::Camera::get_view_matrix
Ragot::Transform::get_position
Ragot::Renderer::init

Here is the caller graph for this function:



6.22.3.5 set_scene()

```
void Ragot::Renderer::set_scene (
             Scene * scene)  [inline]
```

Sets the current scene to be rendered.

This method sets the current scene to be rendered by the renderer. It allows the renderer to access the scene's objects and properties for rendering.

Parameters

scene	Pointer to the Scene object to be set as the current scene.

Here is the caller graph for this function:

main
Ragot::Renderer::set_scene

6.22.3.6 start()

```
void Ragot::Renderer::start ()  [inline]
```

Starts the rendering process.

This method sets the running flag to true, indicating that the renderer is ready to start rendering. Here is the caller graph for this function:

main
Ragot::Renderer::start

6.22.3.7 stop()

```
void Ragot::Renderer::stop ()  [inline]
```

Stops the rendering process.

This method sets the running flag to false, indicating that the renderer should stop rendering.

6.22.3.8 task_render()

```
void Ragot::Renderer::task_render (
             std::stop_token stop_token)
```

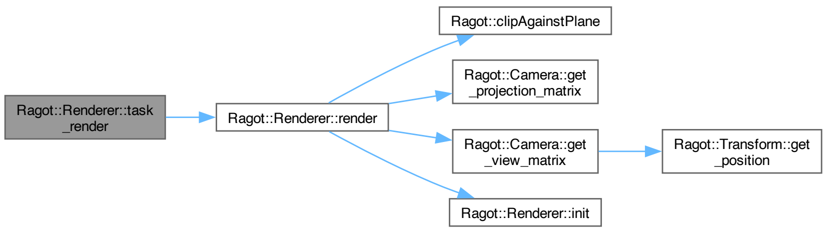
Performs the rendering task in a separate thread.

This method runs the rendering task in a separate thread, allowing for asynchronous rendering.

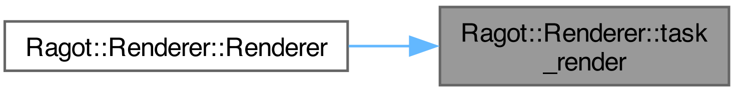
Parameters

stop_token	

Here is the call graph for this function:


Ragot::Renderer::task_render
Ragot::Renderer::render
Ragot::clipAgainstPlane
Ragot::Camera::get_projection_matrix
Ragot::Camera::get_view_matrix
Ragot::Transform::get_position
Ragot::Renderer::init

Here is the caller graph for this function:



6.22.4 Member Data Documentation

6.22.4.1 accumulated_time

```
float Ragot::Renderer::accumulated_time = 0.f  [private]
```

Accumulated time for rendering frames, used for timing and performance measurement.

6.22.4.2 current_scene

```
Scene* Ragot::Renderer::current_scene = nullptr  [private]
```

Pointer to the current scene being rendered, allows access to scene data and objects.

6.22.4.3 display_vertices

```
std::vector< glm::ivec4 > Ragot::Renderer::display_vertices
```

Vector to store display vertices for rendering, used to hold the vertices after applying viewport transformations and clipping.

6.22.4.4 fragment_shader_code

```
const string Ragot::Renderer::fragment_shader_code [static], [private]
```

Initial value:

```
=
    "#version 330\n"
    "in vec2 vTex;"
    "out vec4 FragColor;"
    ""
    "uniform sampler2D uSampler;"
    ""
    "void main()"
    "{"
    "    FragColor = texture(uSampler, vTex);"
    "}"
```

Fragment shader code for rendering, used in non-ESP platforms.

6.22.4.5 frame_buffer

```
FrameBuffer< RGB565 > Ragot::Renderer::frame_buffer [private]
```

Frame buffer for rendering, used to store pixel data for the rendered scene.

6.22.4.6 height

```
unsigned Ragot::Renderer::height [private]
```

Height of the rendering area in pixels, used to define the size of the frame buffer and viewport.

6.22.4.7 initialized

```
bool Ragot::Renderer::initialized = false [private]
```

Flag to indicate if the renderer has been initialized, used to prevent re-initialization and ensure resources are set up correctly.

6.22.4.8 iterations

```
size_t Ragot::Renderer::iterations = 0 [private]
```

Number of iterations for rendering, used for performance testing and optimization.

6.22.4.9 number_of_iterations

```
size_t Ragot::Renderer::number_of_iterations = 10000000000000000  [static], [constexpr], [private]
```

6.22.4.10 quadEBO

```
GLuint Ragot::Renderer::quadEBO = 0  [private]
```

Element Buffer Object for the full-screen quad, used in non-ESP platforms.

6.22.4.11 quadShader

```
std::unique_ptr< Shader_Program > Ragot::Renderer::quadShader = nullptr  [private]
```

Number of iterations for performance testing, can be adjusted for different scenarios.

Shader program for rendering a full-screen quad, used in non-ESP platforms.

6.22.4.12 quadVAO

```
GLuint Ragot::Renderer::quadVAO = 0  [private]
```

Vertex Array Object for the full-screen quad, used in non-ESP platforms.

6.22.4.13 quadVBO

```
GLuint Ragot::Renderer::quadVBO = 0  [private]
```

Vertex Buffer Object for the full-screen quad, used in non-ESP platforms.

6.22.4.14 rasterizer

```
Rasterizer< RGB565 > Ragot::Renderer::rasterizer  [private]
```

Rasterizer for rendering polygons in the frame buffer, responsible for filling polygons with color and handling depth testing.

6.22.4.15 running

```
std::atomic<bool> Ragot::Renderer::running = false  [private]
```

Flag to indicate if the renderer is currently running, used to control rendering tasks and stop them gracefully.

6.22.4.16 transformed_vertices

```
std::vector< glm::fvec4 > Ragot::Renderer::transformed_vertices
```

Vector to store transformed vertices for rendering, used to hold the vertices after applying transformations such as model, view, and projection matrices.

6.22.4.17 vertex_shader_code

```
const string Ragot::Renderer::vertex_shader_code  [static], [private]
```

Initial value:

```
=
    "#version 330\n"
    "layout(location = 0) in vec2 aPos;"
    "layout(location = 1) in vec2 aTex;"
    ""
    "out vec2 vTex;"
    ""
    "void main()"
    "{"
    "   vTex = aTex;"
    "   gl_Position = vec4(aPos, 0.0, 1.0);"
    "}"
```

Vertex shader code for rendering, used in non-ESP platforms.

6.22.4.18 width

```
unsigned Ragot::Renderer::width  [private]
```

Width of the rendering area in pixels, used to define the size of the frame buffer and viewport.

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

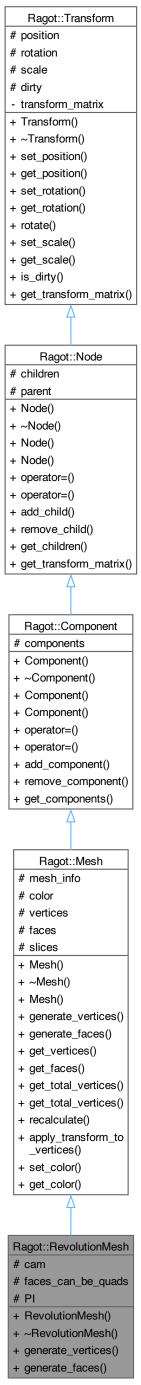
- main/Renderer.hpp
- main/Renderer.cpp

6.23 Ragot::RevolutionMesh Class Reference

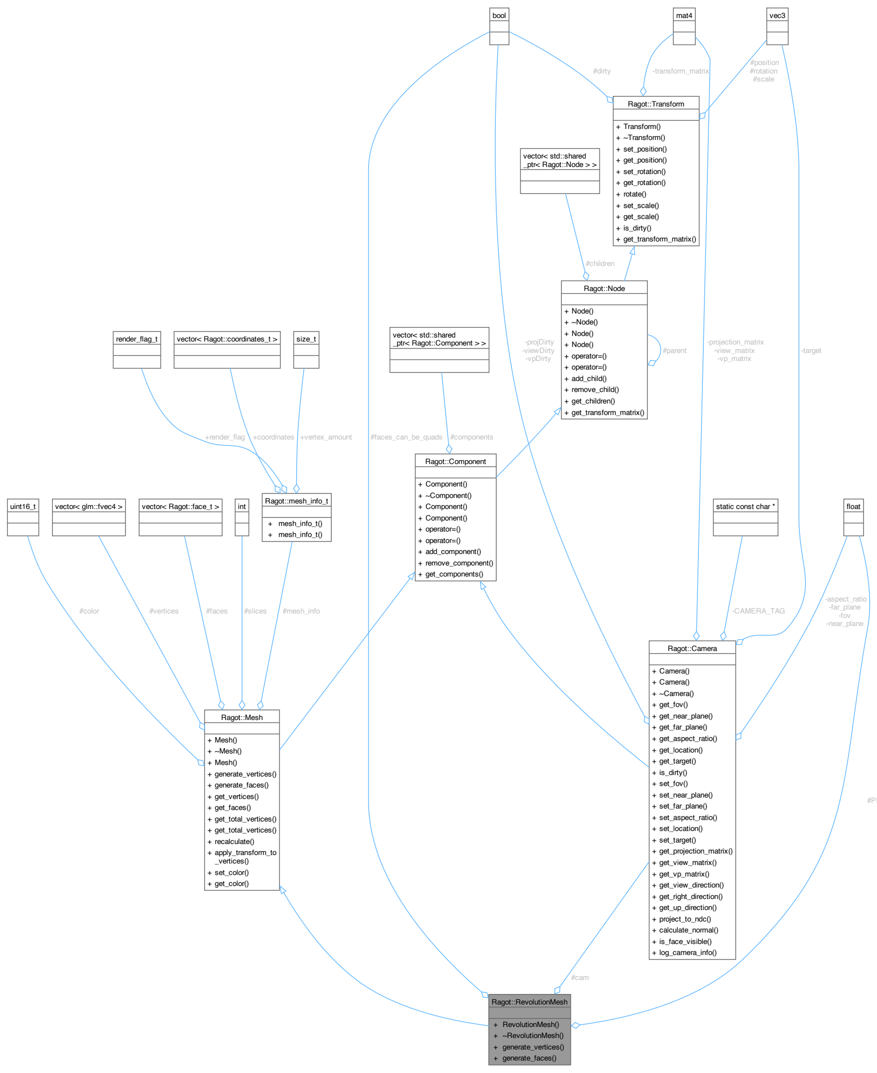
Class for generating revolution meshes.

```
#include <RevolutionMesh.hpp>
```

Inheritance diagram for Ragot::RevolutionMesh:



Collaboration diagram for Ragot::RevolutionMesh:



Public Member Functions

- RevolutionMesh (mesh_info_t &mesh_info, const Camera &cam)

    Constructor for the RevolutionMesh class.
- ∼RevolutionMesh ()=default

    Default destructor for the RevolutionMesh class.
- void generate_vertices () override

    Generates the vertices for the revolution mesh.
- void generate_faces () override

    Generates the faces for the revolution mesh.

Public Member Functions inherited from Ragot::Mesh

- Mesh ()=delete

    Construct a new Mesh object (deleted constructor).

- virtual ∼Mesh ()=default

  Default virtual destructor for the Mesh class.
- Mesh (mesh_info_t &mesh_info)

  Construct a new Mesh object with mesh information.
- const std::vector< glm::fvec4 > & get_vertices () const

  Get the vertices object.
- const std::vector< face_t > & get_faces () const

  Get the faces object.
- const size_t get_total_vertices () const

  Get the total vertices object.
- size_t get_total_vertices ()

  Get the total vertices object.
- void recalculate ()

  Recalculate the mesh vertices and faces.
- void apply_transform_to_vertices ()

  Apply the current transformation to the vertices of the mesh. This method applies the transformation matrix obtained from the Transform class to each vertex in the mesh. This is useful for updating the mesh vertices after any transformation has been applied, such as translation, rotation, or scaling. It modifies the vertices in place, transforming them according to the current transformation matrix.
- void set_color (uint16_t new_color)

  Set the color of the mesh.
- uint16_t get_color () const

  Get the color of the mesh.

Public Member Functions inherited from Ragot::Component

- Component ()=default

  Default constructor for the Component class.
- virtual ∼Component ()=default

  Default virtual destructor for the Component class.
- Component (const Component &)=delete

  Deleted copy constructor for the Component class.
- Component (const Component &&)=delete

  Deleted move constructor for the Component class.
- Component & operator= (const Component &)=delete

  Deleted assignment operator for the Component class.
- Component & operator= (const Component &&)=delete

  Deleted move assignment operator for the Component class.
- void add_component (std::shared_ptr< Component > component)

  Adds a component to the collection.
- void remove_component (std::shared_ptr< Component > component)

  Removes a component from the collection.
- const std::vector< std::shared_ptr< Component > > get_components () const

  Gets the collection of components.

Public Member Functions inherited from Ragot::Node

- Node ()=default

  Default constructor for Node. Initializes an empty node with no parent and no children.
- virtual ∼Node ()=default

  Default destructor for Node. Cleans up the node and its children.
- Node (const Node &)=delete

  Deleted copy constructor for Node. Prevents copying of Node instances.
- Node (const Node &&)=delete

  Deleted move constructor for Node. Prevents moving of Node instances.
- Node & operator= (const Node &)=delete

  Deleted assignment operator for Node. Prevents assignment of Node instances.
- Node & operator= (const Node &&)=delete

  Deleted move assignment operator for Node. Prevents moving of Node instances.
- void add_child (std::shared_ptr< Node > child)

  Get the parent node.
- void remove_child (std::shared_ptr< Node > child)

  Remove a child node.
- const std::vector< std::shared_ptr< Node > > & get_children () const

  Get the parent node.
- mat4 get_transform_matrix () override

  Get the transform matrix object.

Public Member Functions inherited from Ragot::Transform

- Transform ()

  Default constructor for the Transform class.
- virtual ∼Transform ()=default

  Virtual destructor for the Transform class.
- void set_position (const vec3 &pos)

  Sets the position of the object.
- vec3 get_position () const

  Gets the current position of the object.
- void set_rotation (const vec3 &rot)

  Moves the object by a specified vector.
- vec3 get_rotation () const

  Gets the current rotation of the object.
- void rotate (const float angle, const vec3 &axis)

  Rotates the object by a specified angle around a given axis.
- void set_scale (const vec3 &scale)

  Sets the scale of the object.
- vec3 get_scale () const

  Sets the scale of the object uniformly.
- bool is_dirty () const

  Checks if the transformation matrix is dirty (needs recalculation).

Protected Attributes

- const Camera & cam

  Reference to the camera used for rendering, providing view direction and other properties.
- bool faces_can_be_quads

  Flag indicating whether the faces can be rendered as quads or triangles.

Protected Attributes inherited from Ragot::Mesh

- mesh_info_t mesh_info

    Information about the mesh, including coordinates and rendering type.
- uint16_t color = 0xFFFF

    Color of the mesh, default is white (0xFFFF).
- std::vector< glm::fvec4 > vertices

    Vector of vertices representing the mesh in 3D space.
- std::vector< face_t > faces

    Vector of faces representing the mesh, each face can be a triangle or a quad.
- int slices = 16

    Number of slices for generating the mesh, default is 16.

Protected Attributes inherited from Ragot::Component

- std::vector< std::shared_ptr< Component > > components

    Collection of components managed by this Component instance.

Protected Attributes inherited from Ragot::Node

- std::vector< std::shared_ptr< Node > > children

    List of child nodes.
- Node $*$ parent = nullptr

    Pointer to the parent node.

Protected Attributes inherited from Ragot::Transform

- vec3 position

    The position of the object in 3D space.
- vec3 rotation

    The rotation of the object in degrees around each axis (x, y, z).
- vec3 scale

    The scale of the object in 3D space, default is (1, 1, 1).
- bool dirty = true

    Flag indicating whether the transformation matrix needs to be recalculated.

Static Protected Attributes

- static constexpr float PI = 3.14159265358979323846f

    Constant value for Pi, used in calculations involving angles and rotations.

6.23.1 Detailed Description

Class for generating revolution meshes.

This class generates a mesh by revolving a 2D profile around an axis. It inherits from the Mesh class and implements the methods to generate vertices and faces.

6.23.2 Constructor & Destructor Documentation

6.23.2.1 RevolutionMesh()

```
Ragot::RevolutionMesh::RevolutionMesh (
            mesh_info_t & mesh_info,
            const Camera & cam)  [inline]
```

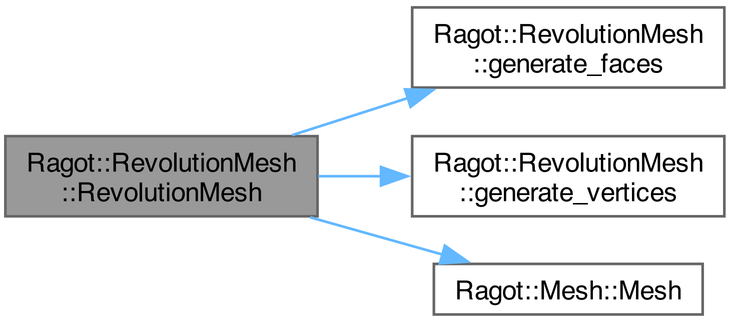
Constructor for the RevolutionMesh class.

Initializes the mesh with the provided mesh information and camera.

Parameters

mesh_info	Information about the mesh, including coordinates and rendering type.
cam	Reference to the camera used for rendering.

Here is the call graph for this function:


Ragot::RevolutionMesh ::generate_faces
Ragot::RevolutionMesh ::RevolutionMesh
Ragot::RevolutionMesh ::generate_vertices
Ragot::Mesh::Mesh

6.23.2.2 ∼RevolutionMesh()

```
Ragot::RevolutionMesh::∼RevolutionMesh ()  [default]
```

Default destructor for the RevolutionMesh class.

Cleans up resources used by the mesh.

6.23.3 Member Function Documentation

6.23.3.1 generate_faces()

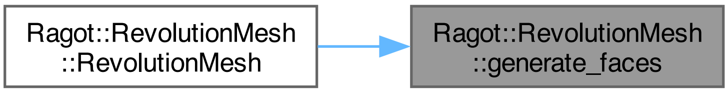
```
void Ragot::RevolutionMesh::generate_faces ()  [override], [virtual]
```

Generates the faces for the revolution mesh.

This method creates the faces of the mesh by connecting the vertices generated by generate_vertices. It can create either triangles or quads based on the mesh information and the flag faces_can_be_quads.

Implements Ragot::Mesh.

Here is the caller graph for this function:


Ragot::RevolutionMesh ::RevolutionMesh
Ragot::RevolutionMesh ::generate_faces

6.23.3.2 generate_vertices()

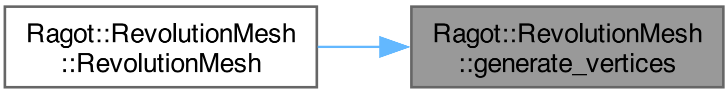
```
void Ragot::RevolutionMesh::generate_vertices ()  [override], [virtual]
```

Generates the vertices for the revolution mesh.

This method calculates the vertices by revolving the 2D profile around the specified axis. It uses the camera's view direction to determine the local space for the vertices.

Implements Ragot::Mesh.

Here is the caller graph for this function:


Ragot::RevolutionMesh ::RevolutionMesh
Ragot::RevolutionMesh ::generate_vertices

6.23.4 Member Data Documentation

6.23.4.1 cam

```
const Camera& Ragot::RevolutionMesh::cam  [protected]
```

Reference to the camera used for rendering, providing view direction and other properties.

6.23.4.2 faces_can_be_quads

```
bool Ragot::RevolutionMesh::faces_can_be_quads  [protected]
```

Flag indicating whether the faces can be rendered as quads or triangles.

6.23.4.3 PI

```
float Ragot::RevolutionMesh::PI = 3.14159265358979323846f  [static], [constexpr], [protected]
```

Constant value for Pi, used in calculations involving angles and rotations.

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

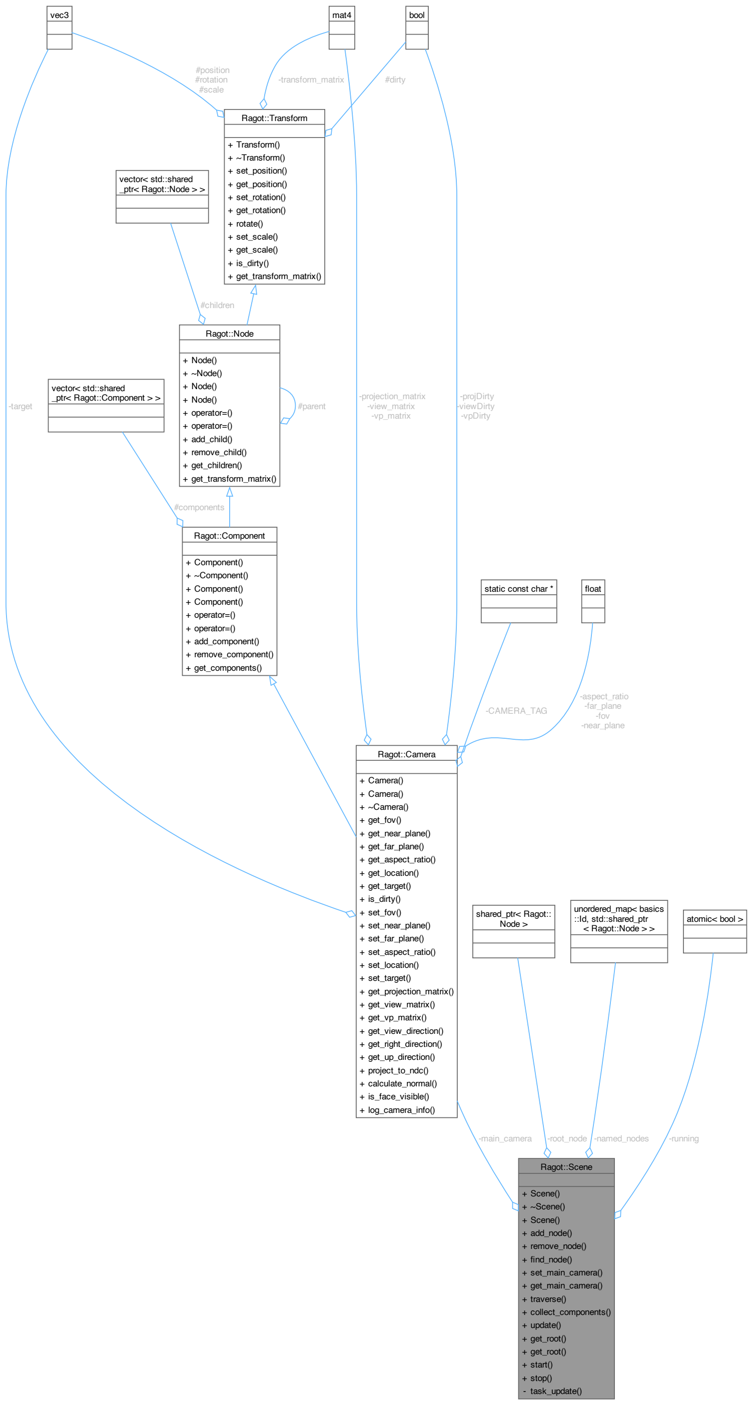
- main/RevolutionMesh.hpp
- main/RevolutionMesh.cpp

6.24 Ragot::Scene Class Reference

Class for managing a 3D scene.

```
#include <Scene.hpp>
```

Collaboration diagram for Ragot::Scene:



Public Member Functions

- Scene ()

  Default constructor for the Scene class.
- ∼Scene ()=default

  Destructor for the Scene class.
- Scene (Camera ∗camera)

Construct a new Scene object.
- void add_node (std::shared_ptr< Node > node, const basics::Id name)

  Adds a node to the scene with a specified name.
- void remove_node (std::shared_ptr< Node > node)

  Removes a node from the scene.
- std::shared_ptr< Node > find_node (const basics::Id name)

  Finds a node in the scene by its name.
- void set_main_camera (Camera ∗camera)

  Sets the main camera for the scene.
- Camera ∗ get_main_camera () const

  Gets the main camera of the scene.
- void traverse (const std::function< void(std::shared_ptr< Node >) > &callback)

  Traverses the scene graph and applies a callback function to each node.
- template<typename T>
  std::vector< std::shared_ptr< T > > collect_components ()

  Collects all components of a specified type from the scene.
- void update (float delta_time)

  Updates the scene with a specified delta time.
- std::shared_ptr< Node > get_root ()

  Get the root object.
- const std::shared_ptr< Node > get_root () const

  Get the root object.
- void start ()

  Starts the scene, setting the running flag to true.
- void stop ()

  Stops the scene, setting the running flag to false.

Private Member Functions

- void task_update (std::stop_token, float delta_time)

  Task to update the scene in a separate thread.

Private Attributes

- Camera ∗ main_camera = nullptr

  Pointer to the main camera used for rendering the scene.
- std::shared_ptr< Node > root_node

  Shared pointer to the root node of the scene graph.
- std::unordered_map< basics::Id, std::shared_ptr< Node > > named_nodes

  Map of named nodes for quick access by name.
- std::atomic< bool > running = false

  Flag indicating whether the scene is currently running or not.

6.24.1 Detailed Description

Class for managing a 3D scene.

The Scene class provides methods to manage nodes, cameras, and scene traversal. It allows adding and removing nodes, setting the main camera, and traversing the scene graph.

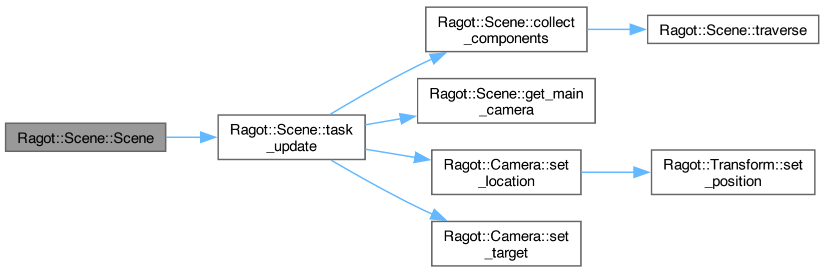
6.24.2 Constructor & Destructor Documentation

6.24.2.1 Scene() [1/2]

```
Ragot::Scene::Scene ()
```

Default constructor for the Scene class.

Initializes the scene with a root node and prepares it for use. Here is the call graph for this function:


Ragot::Scene::Scene
Ragot::Scene::task _update
Ragot::Scene::collect _components
Ragot::Scene::get_main _camera
Ragot::Camera::set _location
Ragot::Camera::set _target
Ragot::Scene::traverse
Ragot::Transform::set _position

6.24.2.2 ∼Scene()

```
Ragot::Scene::∼Scene () [default]
```

Destructor for the Scene class.

6.24.2.3 Scene() [2/2]

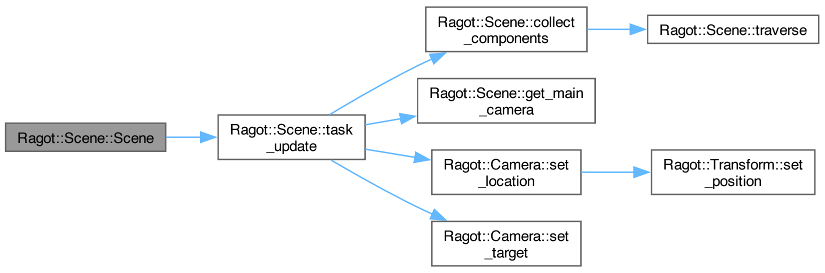
```
Ragot::Scene::Scene (
            Camera * camera)
```

Construct a new Scene object.

Parameters

camera	Pointer to the Camera object to be used as the main camera for the scene.

Here is the call graph for this function:


Ragot::Scene::Scene
Ragot::Scene::task _update
Ragot::Scene::collect _components
Ragot::Scene::get_main _camera
Ragot::Camera::set _location
Ragot::Camera::set _target
Ragot::Scene::traverse
Ragot::Transform::set _position

6.24.3 Member Function Documentation

6.24.3.1 add_node()

```
void Ragot::Scene::add_node (
             std::shared_ptr< Node > node,
             const basics::Id name)
```

Adds a node to the scene with a specified name.

Parameters

node	Shared pointer to the Node object to be added to the scene.
name	Unique identifier for the node, used for quick access.

6.24.3.2 collect_components()

```
template<typename T>
template std::vector< std::shared_ptr< Camera > > Ragot::Scene::collect_components< Camera >
()
```

Collects all components of a specified type from the scene.

This method traverses the scene graph and collects all components of the specified type.

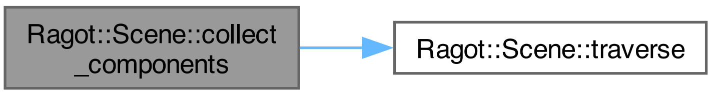
Template Parameters

T	The type of component to collect.

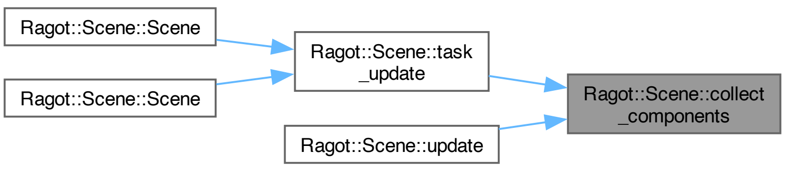
Returns

std::vector<std::shared_ptr<T>> A vector containing shared pointers to the collected components.

Here is the call graph for this function:



Here is the caller graph for this function:



6.24.3.3 find_node()

```
std::shared_ptr< Node > Ragot::Scene::find_node (
            const basics::Id name)
```

Finds a node in the scene by its name.

Parameters

name	Unique identifier for the node to be found.

Returns

std::shared_ptr<Node> Shared pointer to the found Node object, or nullptr if not found.

6.24.3.4 get_main_camera()

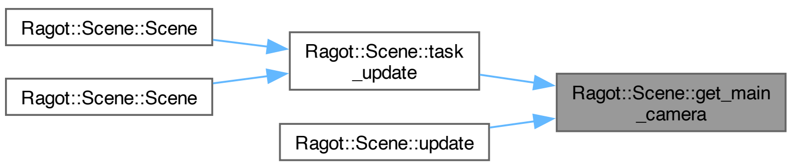
```
Camera * Ragot::Scene::get_main_camera () const [inline]
```

Gets the main camera of the scene.

Returns

Camera* Pointer to the main Camera object used for rendering the scene.

Here is the caller graph for this function:



6.24.3.5 get_root() [1/2]

```
std::shared_ptr< Node > Ragot::Scene::get_root () [inline]
```

Get the root object.

Returns

std::shared_ptr < Node >

6.24.3.6 get_root() [2/2]

```
const std::shared_ptr< Node > Ragot::Scene::get_root () const  [inline]
```

Get the root object.

Returns

const std::shared_ptr < Node >

6.24.3.7 remove_node()

```
void Ragot::Scene::remove_node (
             std::shared_ptr< Node > node)
```

Removes a node from the scene.

Parameters

node	Shared pointer to the Node object to be removed from the scene.

6.24.3.8 set_main_camera()

```
void Ragot::Scene::set_main_camera (
             Camera * camera)
```

Sets the main camera for the scene.

Parameters

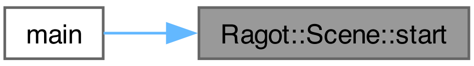
camera	Pointer to the Camera object to be set as the main camera for the scene.

6.24.3.9 start()

```
void Ragot::Scene::start ()  [inline]
```

Starts the scene, setting the running flag to true.

This method is used to indicate that the scene is active and should be updated. Here is the caller graph for this function:



6.24.3.10 stop()

```
void Ragot::Scene::stop ()  [inline]
```

Stops the scene, setting the running flag to false.

This method is used to indicate that the scene is no longer active and should not be updated.

6.24.3.11 task_update()

```
void Ragot::Scene::task_update (
             std::stop_token stop_token,
             float delta_time)  [private]
```

Task to update the scene in a separate thread.

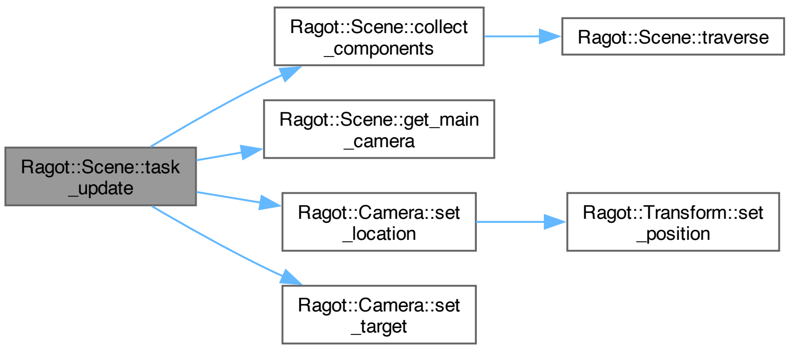
This method runs in a separate thread and updates the scene based on the delta time. It uses a stop token to allow for graceful termination of the task.

Parameters

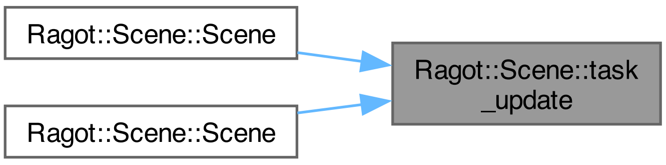
stop_token	Token to signal when the task should stop.
delta_time	The time elapsed since the last update, in seconds.

< Espera inicial para asegurar que la cámara esté lista.

< Espera inicial para asegurar que la cámara esté lista.Here is the call graph for this function:


Ragot::Scene::collect _components
Ragot::Scene::traverse
Ragot::Scene::get_main _camera
Ragot::Scene::task _update
Ragot::Camera::set _location
Ragot::Transform::set _position
Ragot::Camera::set _target

Here is the caller graph for this function:



6.24.3.12 traverse()

```
void Ragot::Scene::traverse (
             const std::function< void(std::shared_ptr< Node >) > & callback)
```

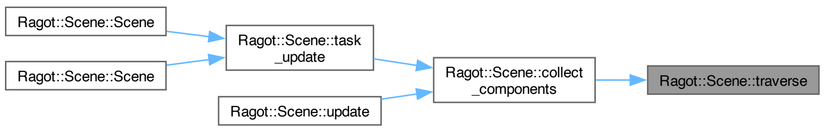
Traverses the scene graph and applies a callback function to each node.

This method allows for custom operations on each node in the scene graph.

Parameters

callback	Function to be called for each node in the scene graph.

Here is the caller graph for this function:



6.24.3.13 update()

```
void Ragot::Scene::update (
             float delta_time)
```

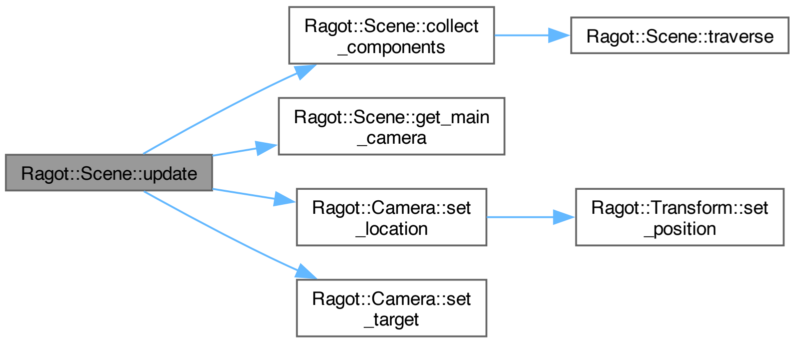
Updates the scene with a specified delta time.

This method updates the scene, allowing for animations or other time-based changes.

Parameters

delta_time	The time elapsed since the last update, in seconds.

Here is the call graph for this function:


Ragot::Scene::collect_components
Ragot::Scene::traverse
Ragot::Scene::get_main_camera
Ragot::Scene::update
Ragot::Camera::set_location
Ragot::Transform::set_position
Ragot::Camera::set_target

6.24.4 Member Data Documentation

6.24.4.1 main_camera

```
Camera* Ragot::Scene::main_camera = nullptr  [private]
```

Pointer to the main camera used for rendering the scene.

6.24.4.2 named_nodes

```
std::unordered_map<basics::Id, std::shared_ptr < Node > > Ragot::Scene::named_nodes  [private]
```

Map of named nodes for quick access by name.

6.24.4.3 root_node

```
std::shared_ptr< Node > Ragot::Scene::root_node  [private]
```

Shared pointer to the root node of the scene graph.

6.24.4.4 running

```
std::atomic<bool> Ragot::Scene::running = false  [private]
```

Flag indicating whether the scene is currently running or not.

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

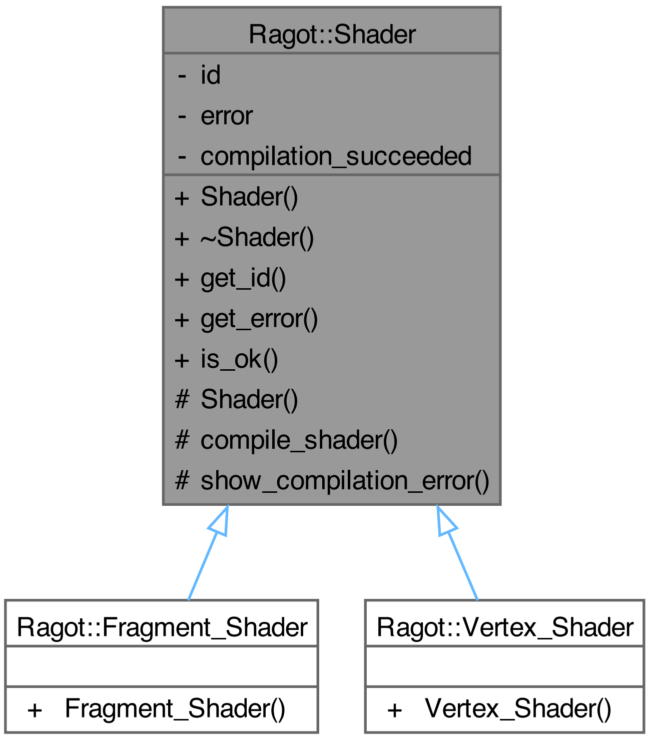
- main/Scene.hpp
- main/Scene.cpp

6.25 Ragot::Shader Class Reference

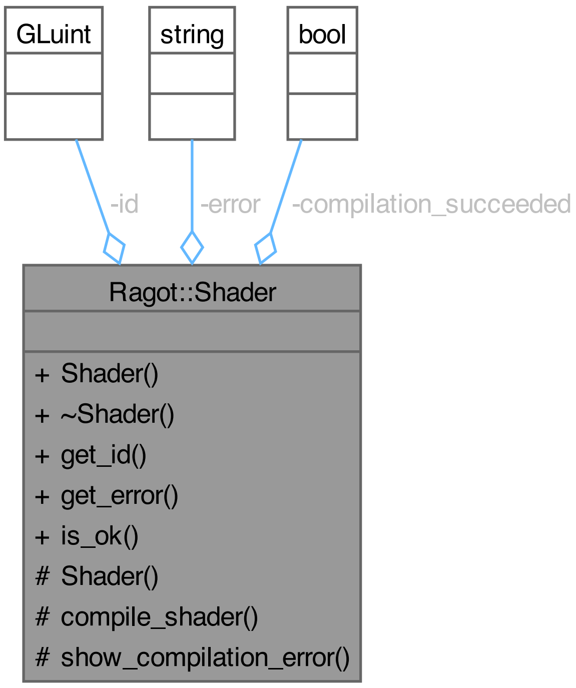
Class for managing an OpenGL shader.

```
#include <Shader_Program.hpp>
```

Inheritance diagram for Ragot::Shader:



Collaboration diagram for Ragot::Shader:



Public Member Functions

- Shader ()=delete

    Deleted default constructor.
- ∼Shader ()

    Destructor for the Shader class.
- GLuint get_id () const

    Gets the shader ID.
- string ∗ get_error ()

    Gets the compilation error message.
- bool is_ok () const

    Checks if the shader is compiled successfully.

Protected Member Functions

- Shader (const vector< string > &source_code, GLenum type)

    Constructor for the Shader class.
- GLuint compile_shader ()

    Compiles the shader.
- void show_compilation_error ()

    Displays compilation errors.

Private Attributes

- GLuint id

  Shader ID.
- string error

  Compilation error message.
- bool compilation_succeeded

  Flag indicating if compilation succeeded.

6.25.1 Detailed Description

Class for managing an OpenGL shader.

6.25.2 Constructor & Destructor Documentation

6.25.2.1 Shader() [1/2]

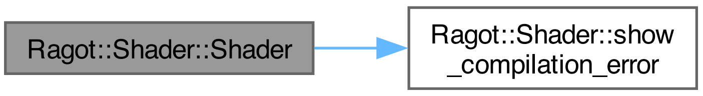
```
Ragot::Shader::Shader (
             const vector< string > & source_code,
             GLenum type)  [protected]
```

Constructor for the Shader class.

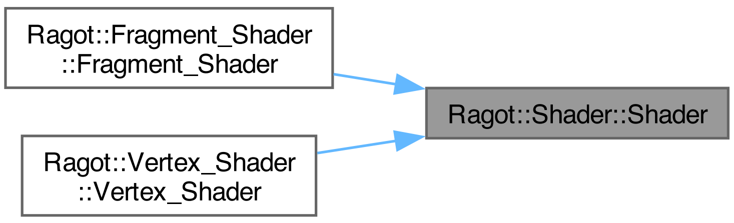
Parameters

source_code	Vector of shader source code.
type	Shader type (e.g., GL_VERTEX_SHADER, GL_FRAGMENT_SHADER).

Here is the call graph for this function:



Here is the caller graph for this function:


Ragot::Fragment_Shader ::Fragment_Shader
Ragot::Vertex_Shader ::Vertex_Shader
Ragot::Shader::Shader

6.25.2.2 Shader() [2/2]

```
Ragot::Shader::Shader ()  [delete]
```

Deleted default constructor.

6.25.2.3 ∼Shader()

```
Ragot::Shader::∼Shader ()  [inline]
```

Destructor for the Shader class.

6.25.3 Member Function Documentation

6.25.3.1 compile_shader()

```
GLuint Ragot::Shader::compile_shader ()  [protected]
```

Compiles the shader.

Returns

Shader ID.

6.25.3.2 get_error()

```
string * Ragot::Shader::get_error ()  [inline]
```

Gets the compilation error message.

Returns

Pointer to the error message string.

6.25.3.3 get_id()

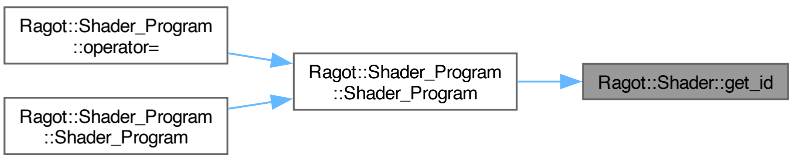
```
GLuint Ragot::Shader::get_id () const  [inline]
```

Gets the shader ID.

Returns

Shader ID.

Here is the caller graph for this function:


Ragot::Shader_Program ::operator=
Ragot::Shader_Program ::Shader_Program
Ragot::Shader_Program ::Shader_Program
Ragot::Shader::get_id

6.25.3.4 is_ok()

```
bool Ragot::Shader::is_ok () const  [inline]
```

Checks if the shader is compiled successfully.

Returns

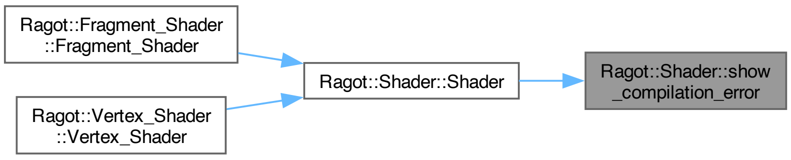
True if compilation succeeded, false otherwise.

6.25.3.5 show_compilation_error()

```
void Ragot::Shader::show_compilation_error ()  [protected]
```

Displays compilation errors.

Here is the caller graph for this function:


Ragot::Fragment_Shader ::Fragment_Shader
Ragot::Vertex_Shader ::Vertex_Shader
Ragot::Shader::Shader
Ragot::Shader::show _compilation_error

6.25.4 Member Data Documentation

6.25.4.1 compilation_succeeded

```
bool Ragot::Shader::compilation_succeeded  [private]
```

Flag indicating if compilation succeeded.

6.25.4.2 error

```
string Ragot::Shader::error  [private]
```

Compilation error message.

6.25.4.3 id

```
GLuint Ragot::Shader::id  [private]
```

Shader ID.

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

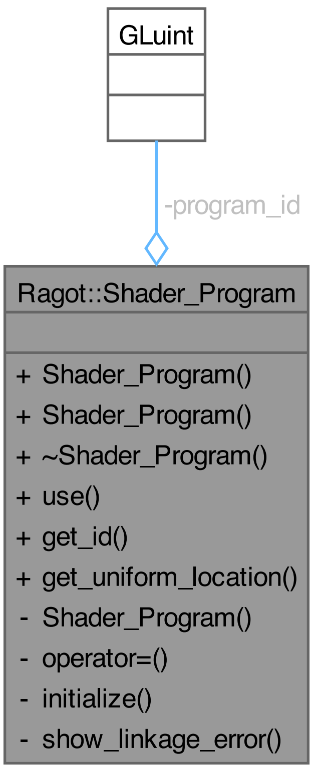
- main/Shader_Program.hpp
- main/Shader_Program.cpp

6.26 Ragot::Shader_Program Class Reference

Class for managing an OpenGL shader program.

```
#include <Shader_Program.hpp>
```

Collaboration diagram for Ragot::Shader_Program:



Public Member Functions

- Shader_Program (const vector< string > &source_code_vertex, const vector< string > &source_code_↩ fragment)

  Constructor for the Shader_Program class.
- Shader_Program ()=delete

  Deleted default constructor.
- ∼Shader_Program ()

  Destructor for the Shader_Program class.
- void use () const

  Uses the shader program.
- GLuint get_id () const

  Gets the shader program ID.
- GLuint get_uniform_location (string uniform_name) const

  Gets the uniform location in the shader program.

Private Member Functions

- Shader_Program (const Shader_Program &)=delete

  Deleted copy constructor.
- Shader_Program & operator= (const Shader_Program &)=delete

  Deleted copy assignment operator.
- void initialize (GLuint vertex_shader_id, GLuint fragment_shader_id)

  Initializes the shader program.
- void show_linkage_error ()

  Displays linkage errors.

Private Attributes

- GLuint program_id

  Shader program ID.

6.26.1 Detailed Description

Class for managing an OpenGL shader program.

6.26.2 Constructor & Destructor Documentation

6.26.2.1 Shader_Program() [1/3]

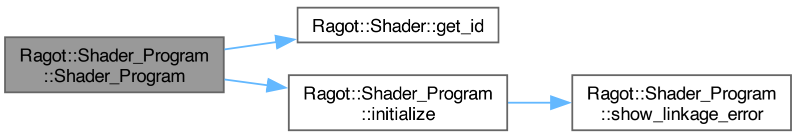
```
Ragot::Shader_Program::Shader_Program (
            const vector< string > & source_code_vertex,
            const vector< string > & source_code_fragment)
```

Constructor for the Shader_Program class.

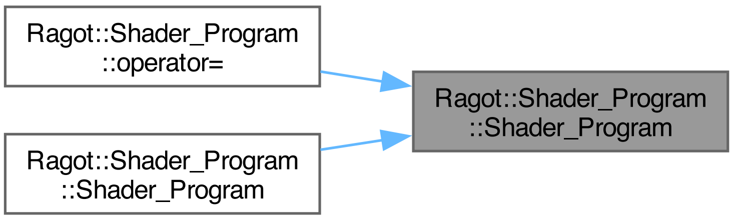
Parameters

source_code_vertex	Vector of vertex shader source code.
source_code_fragment	Vector of fragment shader source code.

Here is the call graph for this function:


Ragot::Shader_Program ::Shader_Program
Ragot::Shader::get_id
Ragot::Shader_Program ::initialize
Ragot::Shader_Program ::show_linkage_error

Here is the caller graph for this function:



6.26.2.2 Shader_Program() `[2/3]`

```
Ragot::Shader_Program::Shader_Program ()  [delete]
```

Deleted default constructor.

6.26.2.3 ∼Shader_Program()

```
Ragot::Shader_Program::∼Shader_Program ()  [inline]
```

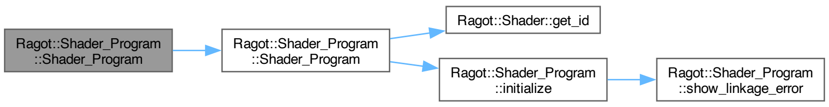
Destructor for the Shader_Program class.

6.26.2.4 Shader_Program() `[3/3]`

```
Ragot::Shader_Program::Shader_Program (
            const Shader_Program & )  [private], [delete]
```

Deleted copy constructor.

Here is the call graph for this function:



6.26.3 Member Function Documentation

6.26.3.1 get_id()

```
GLuint Ragot::Shader_Program::get_id () const  [inline]
```

Gets the shader program ID.

Returns

Shader program ID.

6.26.3.2 get_uniform_location()

```
GLuint Ragot::Shader_Program::get_uniform_location (
            string uniform_name) const  [inline]
```

Gets the uniform location in the shader program.

Parameters

uniform_name	Name of the uniform.

Returns

Uniform location.

6.26.3.3 initialize()

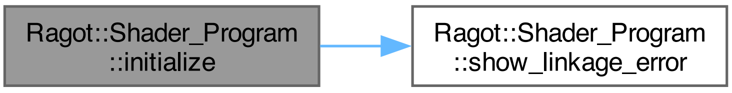
```
void Ragot::Shader_Program::initialize (
            GLuint vertex_shader_id,
            GLuint fragment_shader_id)  [private]
```

Initializes the shader program.

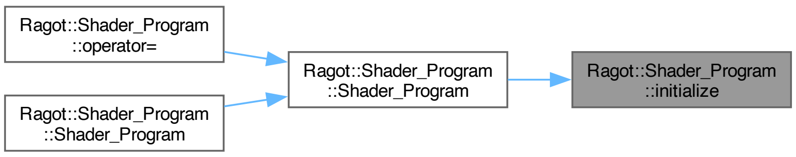
Parameters

vertex_shader_id	Vertex shader ID.
fragment_shader↩ _id	Fragment shader ID.

Here is the call graph for this function:


Ragot::Shader_Program ::initialize
Ragot::Shader_Program ::show_linkage_error

Here is the caller graph for this function:

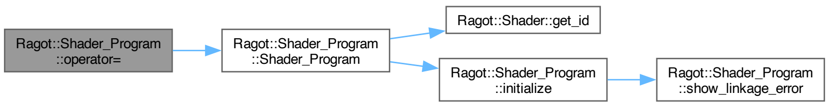

Ragot::Shader_Program ::operator=
Ragot::Shader_Program ::Shader_Program
Ragot::Shader_Program ::Shader_Program
Ragot::Shader_Program ::initialize

6.26.3.4 operator=()

```
Shader_Program & Ragot::Shader_Program::operator= (
             const Shader_Program & )  [private], [delete]
```

Deleted copy assignment operator.

Here is the call graph for this function:

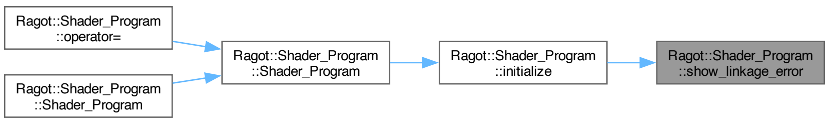

Ragot::Shader_Program ::operator=
Ragot::Shader_Program ::Shader_Program
Ragot::Shader::get_id
Ragot::Shader_Program ::initialize
Ragot::Shader_Program ::show_linkage_error

6.26.3.5 show_linkage_error()

```
void Ragot::Shader_Program::show_linkage_error ()  [private]
```

Displays linkage errors.

Here is the caller graph for this function:


Ragot::Shader_Program ::operator=
Ragot::Shader_Program ::Shader_Program
Ragot::Shader_Program ::Shader_Program
Ragot::Shader_Program ::initialize
Ragot::Shader_Program ::show_linkage_error

6.26.3.6 use()

```
void Ragot::Shader_Program::use () const  [inline]
```

Uses the shader program.

6.26.4 Member Data Documentation

6.26.4.1 program_id

```
GLuint Ragot::Shader_Program::program_id  [private]
```

Shader program ID.

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

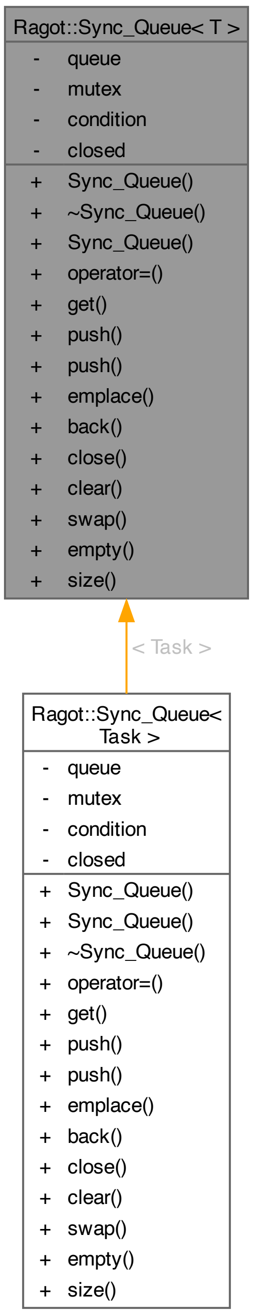
- main/Shader_Program.hpp
- main/Shader_Program.cpp

6.27 Ragot::Sync_Queue< T > Class Template Reference

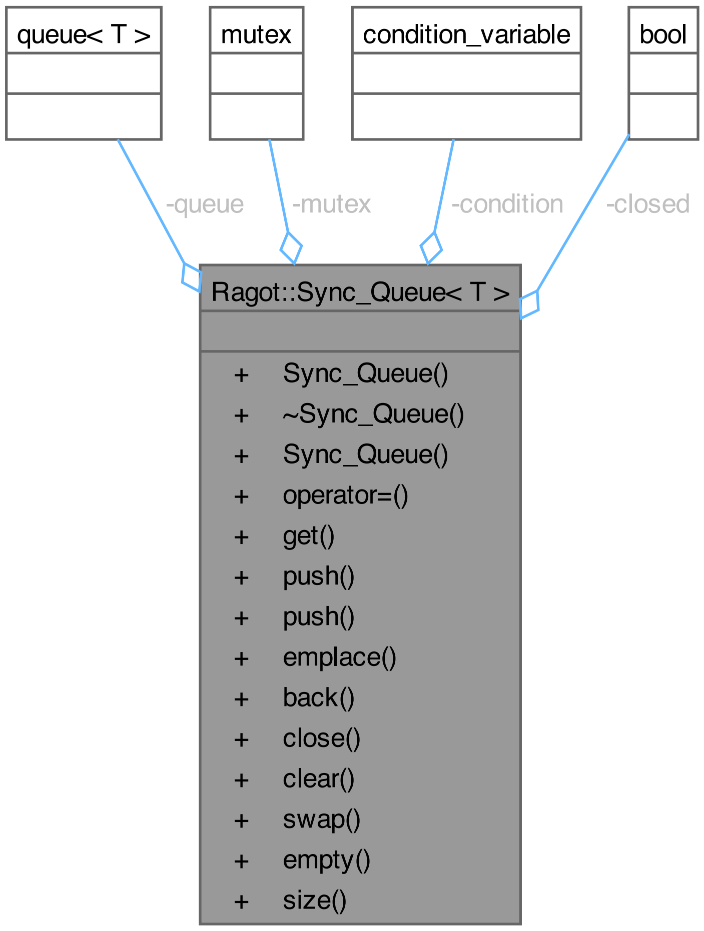
A thread-safe queue implementation.

```
#include <Sync_Queue.hpp>
```

Inheritance diagram for Ragot::Sync_Queue< T >:



Collaboration diagram for Ragot::Sync_Queue< T >:



Public Types

- using value_type = T

Public Member Functions

- Sync_Queue ()=default

  Construct a new Sync_Queue object.
- ∼Sync_Queue ()

  Destroy the Sync_Queue object.
- Sync_Queue (const Sync_Queue &)=delete

  Construct a new Sync_Queue object.
- Sync_Queue & operator= (const Sync_Queue &)=delete

  Assignment operator for Sync_Queue.
- std::optional< value_type > get ()

Retrieves an element from the queue.

- void push (const value_type &value)

  Pushes an element into the queue.
- template<typename ... ARGUMENTS>
  void push (ARGUMENTS &&...arguments)

  Pushes an element into the queue.
- template<typename ... ARGUMENTS>
  void emplace (ARGUMENTS &&...arguments)

  Emplaces an element into the queue.
- value_type & back ()

  Retrieves the front element of the queue without removing it.
- void close ()

  Closes the queue.
- void clear ()

  Clears the queue.
- void swap (Sync_Queue &other)

  Swaps the contents of this queue with another queue.
- bool empty () const

  Checks if the queue is empty.
- size_t size () const

  Gets the size of the queue.

Private Attributes

- std::queue< T > queue

  The underlying queue to store elements, used for thread-safe operations.
- std::mutex mutex

  Mutex to protect access to the queue, ensuring that only one thread can modify the queue at a time.
- std::condition_variable condition

  Condition variable to notify waiting threads when elements are available in the queue or when the queue is closed.
- bool closed = false

  Flag to indicate whether the queue is closed, preventing further pushes and allowing threads to exit gracefully when the queue is empty.

6.27.1 Detailed Description

template<typename T>
class Ragot::Sync_Queue< T >

A thread-safe queue implementation.

This class provides a synchronized queue that allows multiple threads to safely push and pop elements. It uses mutexes and condition variables to ensure thread safety.

6.27.2 Member Typedef Documentation

6.27.2.1 value_type

```
template<typename T>
using Ragot::Sync_Queue< T >::value_type = T
```

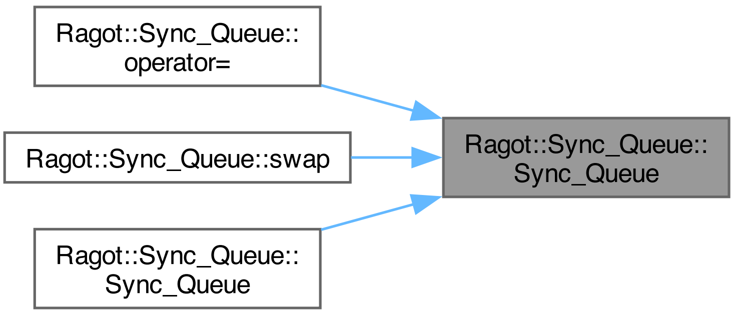
6.27.3 Constructor & Destructor Documentation

6.27.3.1 Sync_Queue() [1/2]

```
template<typename T>
Ragot::Sync_Queue< T >::Sync_Queue ()  [default]
```

Construct a new Sync_Queue object.

Here is the caller graph for this function:

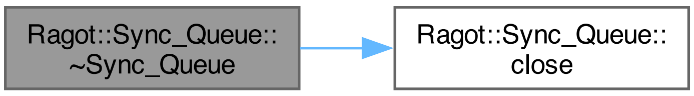

Ragot::Sync_Queue::operator=
Ragot::Sync_Queue::swap
Ragot::Sync_Queue::Sync_Queue
Ragot::Sync_Queue::Sync_Queue

6.27.3.2 ∼Sync_Queue()

```
template<typename T>
Ragot::Sync_Queue< T >::∼Sync_Queue ()  [inline]
```

Destroy the Sync_Queue object.

Closes the queue and releases any resources held by it. Here is the call graph for this function:

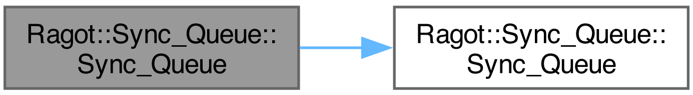


6.27.3.3 Sync_Queue() [2/2]

```
template<typename T>
Ragot::Sync_Queue< T >::Sync_Queue (
            const Sync_Queue< T > & )  [delete]
```

Construct a new Sync_Queue object.

This constructor is deleted to prevent copying of the Sync_Queue object. It ensures that the queue cannot be copied, which is important for thread safety. Here is the call graph for this function:



6.27.4 Member Function Documentation

6.27.4.1 back()

```
template<typename T>
value_type & Ragot::Sync_Queue< T >::back ()  [inline]
```

Retrieves the front element of the queue without removing it.

This method returns a reference to the front element of the queue. It is not thread-safe and should be used with caution.

Returns

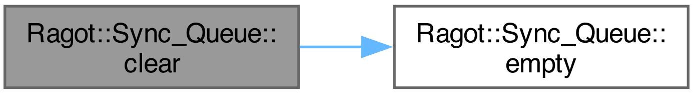
value_type& A reference to the front element of the queue.

6.27.4.2 clear()

```
template<typename T>
void Ragot::Sync_Queue< T >::clear ()  [inline]
```

Clears the queue.

This method removes all elements from the queue and resets it to an empty state. It is thread-safe and can be called while other threads are accessing the queue. Here is the call graph for this function:

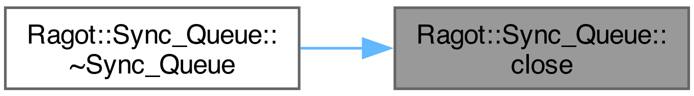


6.27.4.3 close()

```
template<typename T>
void Ragot::Sync_Queue< T >::close ()  [inline]
```

Closes the queue.

This method sets the closed flag to true and notifies all waiting threads. After closing, no more elements can be pushed into the queue. Here is the caller graph for this function:



6.27.4.4 emplace()

```
template<typename T>
template<typename ... ARGUMENTS>
void Ragot::Sync_Queue< T >::emplace (
             ARGUMENTS &&...  arguments)  [inline]
```

Emplaces an element into the queue.

This method constructs an element in place and adds it to the queue if it is not closed. It notifies one waiting thread that an element is available.

Template Parameters

ARGUMENTS	The types of arguments used to construct the element.

Parameters

arguments	The arguments used to construct the element.

6.27.4.5 empty()

```
template<typename T>
bool Ragot::Sync_Queue< T >::empty () const  [inline]
```

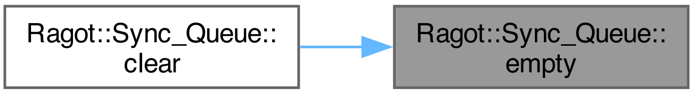
Checks if the queue is empty.

This method checks if the queue is empty by acquiring a lock on the mutex. It returns true if the queue is empty, false otherwise.

Returns

true if the queue is empty, false otherwise.

Here is the caller graph for this function:



6.27.4.6 get()

```
template<typename T>
std::optional< value_type > Ragot::Sync_Queue< T >::get ()  [inline]
```

Retrieves an element from the queue.

This method blocks until an element is available or the queue is closed. If the queue is closed and empty, it returns std::nullopt.

Returns

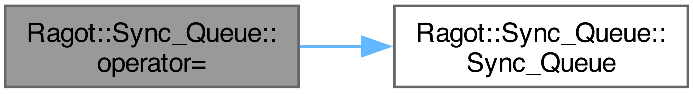
std::optional<value_type> The retrieved element or std::nullopt if the queue is closed and empty.

6.27.4.7 operator=()

```
template<typename T>
Sync_Queue & Ragot::Sync_Queue< T >::operator= (
             const Sync_Queue< T > & )  [delete]
```

Assignment operator for Sync_Queue.

This assignment operator is deleted to prevent copying of the Sync_Queue object. It ensures that the queue cannot be assigned, which is important for thread safety. Here is the call graph for this function:



6.27.4.8 push() [1/2]

```
template<typename T>
template<typename ... ARGUMENTS>
void Ragot::Sync_Queue< T >::push (
            ARGUMENTS &&... arguments) [inline]
```

Pushes an element into the queue.

This method adds an element to the queue if it is not closed. It notifies one waiting thread that an element is available.

Parameters

value	The value to be pushed into the queue.

6.27.4.9 push() [2/2]

```
template<typename T>
void Ragot::Sync_Queue< T >::push (
            const value_type & value) [inline]
```

Pushes an element into the queue.

This method adds an element to the queue if it is not closed. It notifies one waiting thread that an element is available.

Parameters

value	The value to be pushed into the queue.

6.27.4.10 size()

```
template<typename T>
size_t Ragot::Sync_Queue< T >::size () const [inline]
```

Gets the size of the queue.

This method returns the number of elements in the queue by acquiring a lock on the mutex. It is thread-safe and can be called while other threads are accessing the queue.

Returns

size_t The number of elements in the queue.

6.27.4.11 swap()

```
template<typename T>
void Ragot::Sync_Queue< T >::swap (
            Sync_Queue< T > & other) [inline]
```

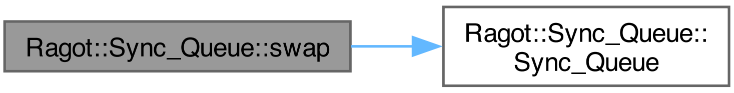
Swaps the contents of this queue with another queue.

This method swaps the contents of this queue with another Sync_Queue. It locks both mutexes to ensure thread safety during the swap operation.

Parameters

other	The other Sync_Queue to swap with.

Here is the call graph for this function:



6.27.5 Member Data Documentation

6.27.5.1 closed

```
template<typename T>
bool Ragot::Sync_Queue< T >::closed = false  [private]
```

Flag to indicate whether the queue is closed, preventing further pushes and allowing threads to exit gracefully when the queue is empty.

6.27.5.2 condition

```
template<typename T>
std::condition_variable Ragot::Sync_Queue< T >::condition  [private]
```

Condition variable to notify waiting threads when elements are available in the queue or when the queue is closed.

6.27.5.3 mutex

```
template<typename T>
std::mutex Ragot::Sync_Queue< T >::mutex  [mutable], [private]
```

Mutex to protect access to the queue, ensuring that only one thread can modify the queue at a time.

6.27.5.4 queue

```
template<typename T>
std::queue<T> Ragot::Sync_Queue< T >::queue  [private]
```

The underlying queue to store elements, used for thread-safe operations.

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

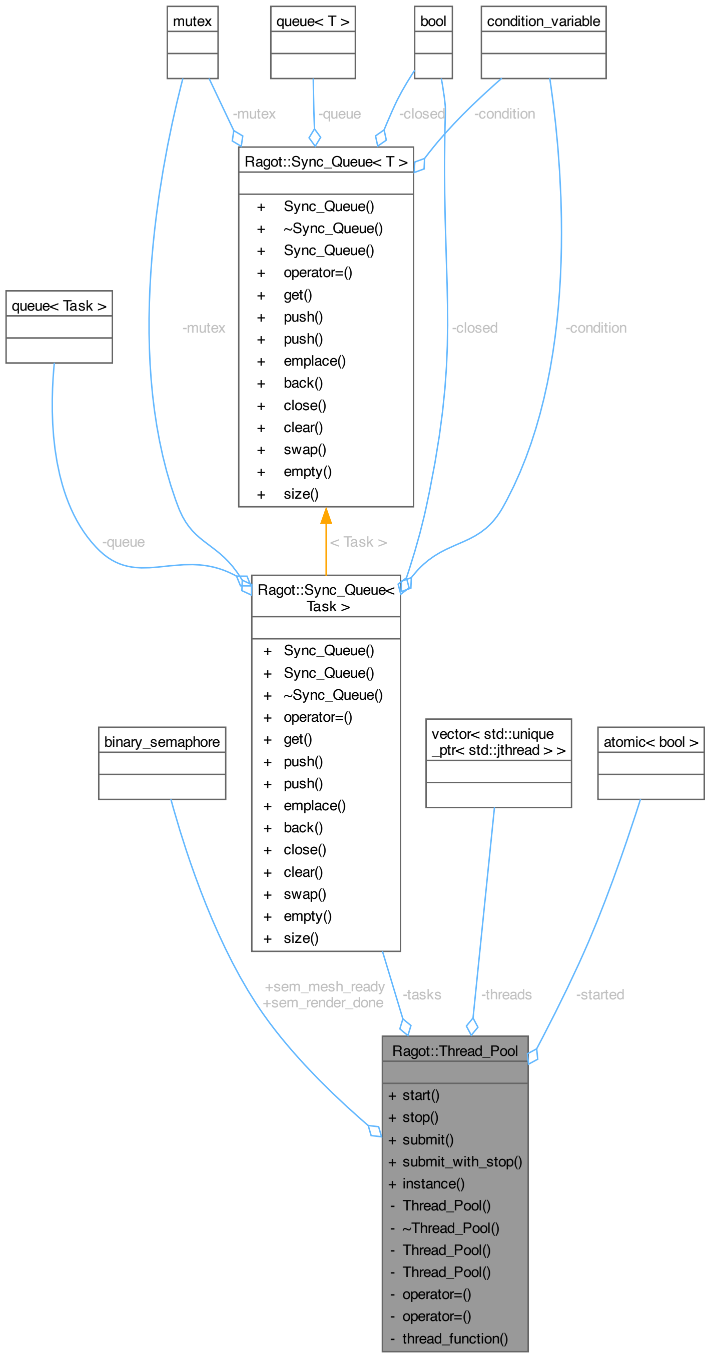
- main/Sync_Queue.hpp

6.28 Ragot::Thread_Pool Class Reference

A thread pool for managing concurrent tasks.

```
#include <Thread_Pool.hpp>
```

Collaboration diagram for Ragot::Thread_Pool:



Public Types

- using Task = std::function < void (std::stop_token) >

Public Member Functions

- void start ()

  Starts the thread pool by creating threads and binding them to the thread_function.
- void stop ()

  Stops the thread pool by closing the task queue and requesting all threads to stop.
- template<typename F, typename... Args>
  std::future< std::invoke_result_t< F, Args... > > submit (F &&f, Args &&... args)

  Submits a task to the thread pool for execution.
- template<typename F, typename... Args>
  std::future< std::invoke_result_t< F, std::stop_token, Args... > > submit_with_stop (F &&f, Args &&... args)

  Submits a task to the thread pool for execution with a stop token.

Static Public Member Functions

- static Thread_Pool & instance ()

  Singleton instance of the Thread_Pool class.

Public Attributes

- std::binary_semaphore sem_mesh_ready {0}

  Semaphore to signal that the mesh is ready for rendering.
- std::binary_semaphore sem_render_done {0}

  Semaphore to signal that the rendering is done.

Private Member Functions

- Thread_Pool ()

  Private constructor for the Thread_Pool class.
- ∼Thread_Pool ()

  Destructor for the Thread_Pool class.
- Thread_Pool (Thread_Pool &)=delete

  Construct a new Thread_Pool object (Deleted). This constructor is deleted to prevent copying of the Thread_Pool object.
- Thread_Pool (Thread_Pool &&)=delete

  Move constructor for the Thread_Pool class (Deleted). This move constructor is deleted to prevent moving of the Thread_Pool object.
- Thread_Pool & operator= (Thread_Pool &)=delete

  Assignment operator for the Thread_Pool class (Deleted). This assignment operator is deleted to prevent copying of the Thread_Pool object.
- Thread_Pool & operator= (Thread_Pool &&)=delete

  Move assignment operator for the Thread_Pool class (Deleted). This move assignment operator is deleted to prevent moving of the Thread_Pool object.
- void thread_function (std::stop_token)

  The function executed by each thread in the thread pool.

Private Attributes

- Sync_Queue< Task > tasks

  A synchronized queue for managing tasks in the thread pool.
- std::vector< std::unique_ptr< std::jthread > > threads

  A vector of unique pointers to jthread objects representing the threads in the pool.
- std::atomic< bool > started

  Flag indicating whether the thread pool has been started or not.

6.28.1 Detailed Description

A thread pool for managing concurrent tasks.

This class provides a thread pool that can execute tasks concurrently using a specified number of threads. It allows submitting tasks and handles synchronization using semaphores.

6.28.2 Member Typedef Documentation

6.28.2.1 Task

```
using Ragot::Thread_Pool::Task = std::function < void (std::stop_token) >
```

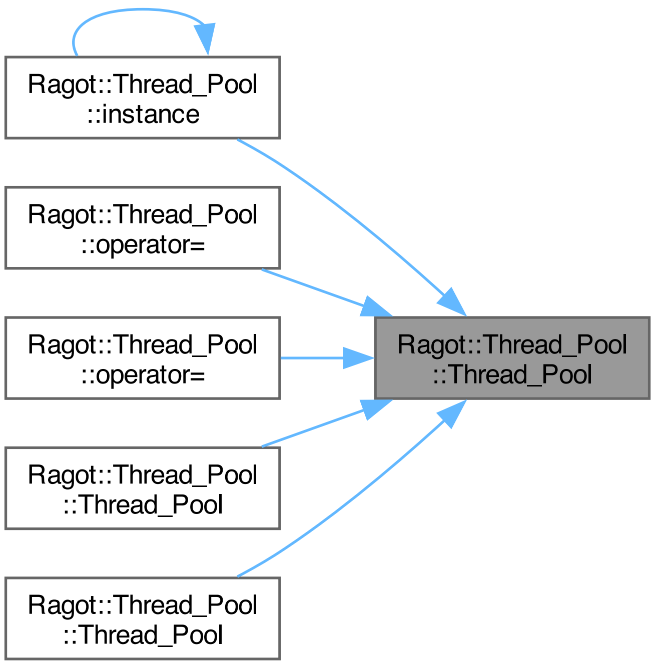
6.28.3 Constructor & Destructor Documentation

6.28.3.1 Thread_Pool() [1/3]

```
Ragot::Thread_Pool::Thread_Pool ()  [inline], [private]
```

Private constructor for the Thread_Pool class.

Initializes the thread pool with the number of threads equal to the number of hardware cores available. If the number of cores is zero, it defaults to two threads. Here is the caller graph for this function:

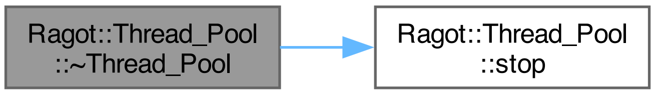

Ragot::Thread_Pool ::instance
Ragot::Thread_Pool ::operator=
Ragot::Thread_Pool ::operator=
Ragot::Thread_Pool ::Thread_Pool
Ragot::Thread_Pool ::Thread_Pool
Ragot::Thread_Pool ::Thread_Pool

6.28.3.2 ∼Thread_Pool()

```
Ragot::Thread_Pool::∼Thread_Pool ()  [inline], [private]
```

Destructor for the Thread_Pool class.

Cleans up resources used by the thread pool, releases semaphores, and stops all threads if they are running. Here is the call graph for this function:

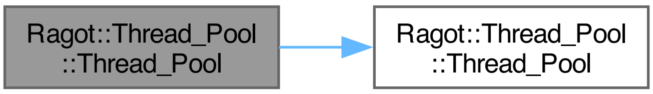


6.28.3.3 Thread_Pool() [2/3]

```
Ragot::Thread_Pool::Thread_Pool (
            Thread_Pool & )  [private], [delete]
```

Construct a new Thread_Pool object (Deleted). This constructor is deleted to prevent copying of the Thread_Pool object.

Here is the call graph for this function:

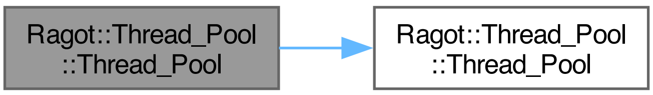

Ragot::Thread_Pool ::Thread_Pool
Ragot::Thread_Pool ::Thread_Pool

6.28.3.4 Thread_Pool() [3/3]

```
Ragot::Thread_Pool::Thread_Pool (
            Thread_Pool && )  [private], [delete]
```

Move constructor for the Thread_Pool class (Deleted). This move constructor is deleted to prevent moving of the Thread_Pool object.

Here is the call graph for this function:


Ragot::Thread_Pool ::Thread_Pool
Ragot::Thread_Pool ::Thread_Pool

6.28.4 Member Function Documentation

6.28.4.1 instance()

```
static Thread_Pool & Ragot::Thread_Pool::instance ()  [inline], [static]
```

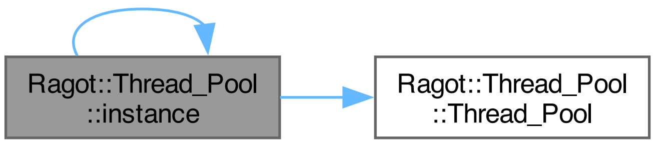
Singleton instance of the Thread_Pool class.

This method provides access to the singleton instance of the Thread_Pool. It ensures that only one instance of the thread pool exists throughout the application.

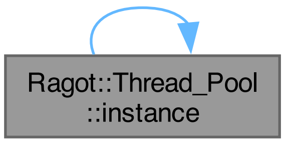
Returns

Thread_Pool& Reference to the singleton instance of the Thread_Pool.

Here is the call graph for this function:



Here is the caller graph for this function:

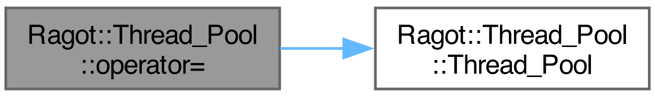


6.28.4.2 operator=() `[1/2]`

```
Thread_Pool & Ragot::Thread_Pool::operator= (
             Thread_Pool && )  [private], [delete]
```

Move assignment operator for the Thread_Pool class (Deleted). This move assignment operator is deleted to prevent moving of the Thread_Pool object.

Here is the call graph for this function:

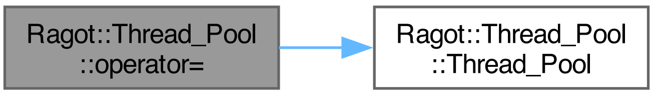


6.28.4.3 operator=() [2/2]

```
Thread_Pool & Ragot::Thread_Pool::operator= (
            Thread_Pool & )  [private], [delete]
```

Assignment operator for the Thread_Pool class (Deleted). This assignment operator is deleted to prevent copying of the Thread_Pool object.

Here is the call graph for this function:

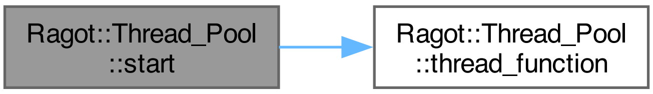

Ragot::Thread_Pool ::operator=
Ragot::Thread_Pool ::Thread_Pool

6.28.4.4 start()

```
void Ragot::Thread_Pool::start ()  [inline]
```

Starts the thread pool by creating threads and binding them to the thread_function.

This method initializes the threads in the pool and starts them, allowing them to execute tasks concurrently. It asserts that the thread pool has not already been started to prevent multiple initializations. Here is the call graph for this function:

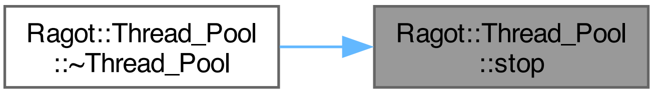

Ragot::Thread_Pool ::start
Ragot::Thread_Pool ::thread_function

6.28.4.5 stop()

```
void Ragot::Thread_Pool::stop ()  [inline]
```

Stops the thread pool by closing the task queue and requesting all threads to stop.

This method stops the thread pool, ensuring that all threads are requested to stop gracefully. It asserts that the thread pool has been started before attempting to stop it. Here is the caller graph for this function:



6.28.4.6 submit()

```
template<typename F, typename... Args>
std::future< std::invoke_result_t< F, Args... > > Ragot::Thread_Pool::submit (
            F && f,
            Args &&... args) [inline]
```

Submits a task to the thread pool for execution.

This method allows submitting a task to the thread pool, which will be executed by one of the threads. It returns a future that can be used to retrieve the result of the task once it is completed.

Template Parameters

F	The type of the function to be executed.
Args	The types of the arguments to be passed to the function.

Parameters

f	The function to be executed.
args	The arguments to be passed to the function.

Returns

std::future<std::invoke_result_t<F, Args...>> A future representing the result of the task.

6.28.4.7 submit_with_stop()

```
template<typename F, typename... Args>
std::future< std::invoke_result_t< F, std::stop_token, Args... > > Ragot::Thread_Pool↩
::submit_with_stop (
            F && f,
            Args &&... args) [inline]
```

Submits a task to the thread pool for execution with a stop token.

This method allows submitting a task to the thread pool, which will be executed by one of the threads. It accepts a stop token that can be used to request cancellation of the task. It returns a future that can be used to retrieve the result of the task once it is completed.

Template Parameters

F	The type of the function to be executed.
Args	The types of the arguments to be passed to the function.

Parameters

f	The function to be executed.
args	The arguments to be passed to the function.

Returns

std::future<std::invoke_result_t<F, std::stop_token, Args...>> A future representing the result of the task.

6.28.4.8 thread_function()

```
void Ragot::Thread_Pool::thread_function (
            std::stop_token stop_token)  [private]
```

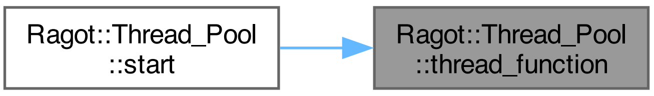
The function executed by each thread in the thread pool.

This function continuously retrieves tasks from the task queue and executes them. It uses a stop token to allow threads to gracefully exit when requested.

Parameters

stop_token	The stop token used to request cancellation of the task.

Here is the caller graph for this function:


Ragot::Thread_Pool ::start
Ragot::Thread_Pool ::thread_function

6.28.5 Member Data Documentation

6.28.5.1 sem_mesh_ready

```
std::binary_semaphore Ragot::Thread_Pool::sem_mesh_ready {0}
```

Semaphore to signal that the mesh is ready for rendering.

6.28.5.2 sem_render_done

```
std::binary_semaphore Ragot::Thread_Pool::sem_render_done {0}
```

Semaphore to signal that the rendering is done.

6.28.5.3 started

```
std::atomic< bool > Ragot::Thread_Pool::started  [private]
```

Flag indicating whether the thread pool has been started or not.

6.28.5.4 tasks

```
Sync_Queue< Task > Ragot::Thread_Pool::tasks  [private]
```

A synchronized queue for managing tasks in the thread pool.

This queue is used to store tasks that need to be executed by the threads in the pool. It provides thread-safe operations for pushing and popping tasks.

6.28.5.5 threads

```
std::vector< std::unique_ptr < std::jthread > > Ragot::Thread_Pool::threads  [private]
```

A vector of unique pointers to jthread objects representing the threads in the pool.

This vector holds the threads that will execute tasks concurrently. Each thread is managed by a unique pointer to ensure proper resource management.

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

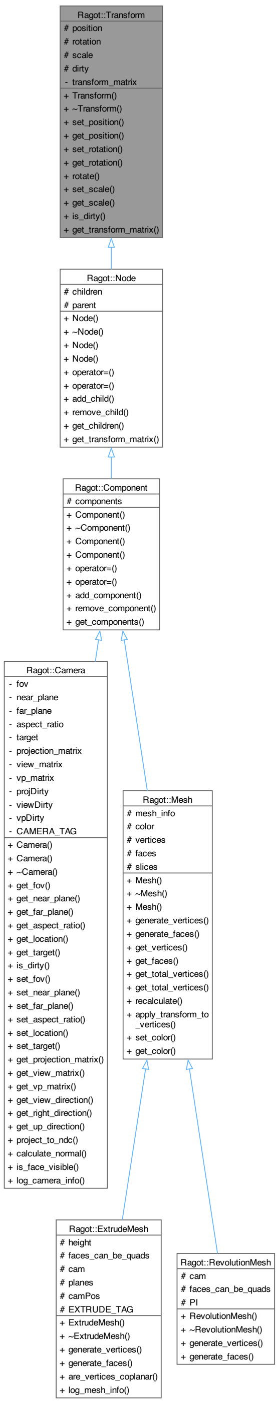
- main/Thread_Pool.hpp
- main/Thread_Pool.cpp

6.29 Ragot::Transform Class Reference

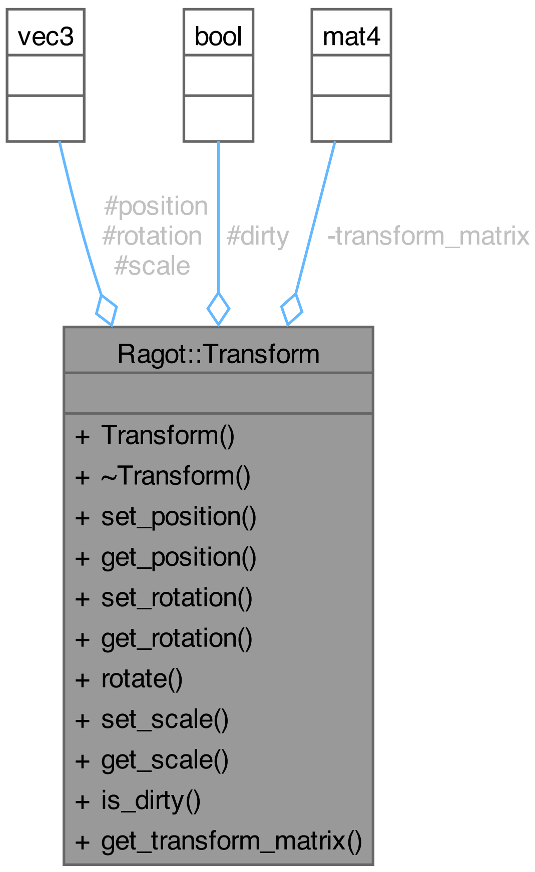
A class representing a 3D transformation with position, rotation, and scale.

```
#include <Transform.hpp>
```

Inheritance diagram for Ragot::Transform:



Collaboration diagram for Ragot::Transform:



Public Member Functions

- Transform ()

  Default constructor for the Transform class.
- virtual ∼Transform ()=default

  Virtual destructor for the Transform class.
- void set_position (const vec3 &pos)

  Sets the position of the object.
- vec3 get_position () const

  Gets the current position of the object.
- void set_rotation (const vec3 &rot)

  Moves the object by a specified vector.
- vec3 get_rotation () const

  Gets the current rotation of the object.
- void rotate (const float angle, const vec3 &axis)

  Rotates the object by a specified angle around a given axis.
- void set_scale (const vec3 &scale)

Sets the scale of the object.
- vec3 get_scale () const

  Sets the scale of the object uniformly.
- bool is_dirty () const

  Checks if the transformation matrix is dirty (needs recalculation).
- virtual mat4 get_transform_matrix ()

  Gets the transformation matrix that combines position, rotation, and scale.

Protected Attributes

- vec3 position

  The position of the object in 3D space.
- vec3 rotation

  The rotation of the object in degrees around each axis (x, y, z).
- vec3 scale

  The scale of the object in 3D space, default is (1, 1, 1).
- bool dirty = true

  Flag indicating whether the transformation matrix needs to be recalculated.

Private Attributes

- mat4 transform_matrix

  The transformation matrix that combines position, rotation, and scale.

6.29.1 Detailed Description

A class representing a 3D transformation with position, rotation, and scale.

This class provides methods to set and get the position, rotation, and scale of a 3D object. It also provides a method to get the transformation matrix that combines these transformations.

6.29.2 Constructor & Destructor Documentation

6.29.2.1 Transform()

```
Ragot::Transform::Transform ()  [inline]
```

Default constructor for the Transform class.

Initializes position to (0, 0, 0), rotation to (0, 0, 0), and scale to (1, 1, 1).

6.29.2.2 ∼Transform()

```
virtual Ragot::Transform::∼Transform ()  [virtual], [default]
```

Virtual destructor for the Transform class.

Ensures proper cleanup of derived classes.

6.29.3 Member Function Documentation

6.29.3.1 get_position()

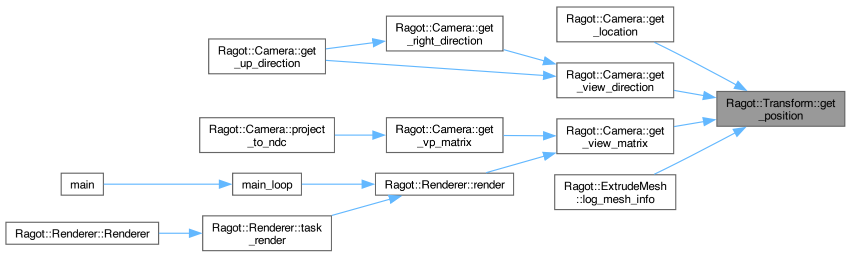
```
vec3 Ragot::Transform::get_position () const  [inline]
```

Gets the current position of the object.

Returns

vec3 The current position.

Here is the caller graph for this function:


Ragot::Camera::get_up_direction
Ragot::Camera::get_right_direction
Ragot::Camera::get_location
Ragot::Camera::get_view_direction
Ragot::Transform::get_position
Ragot::Camera::project_to_ndc
Ragot::Camera::get_vp_matrix
Ragot::Camera::get_view_matrix
main
main_loop
Ragot::Renderer::render
Ragot::ExtrudeMesh::log_mesh_info
Ragot::Renderer::Renderer
Ragot::Renderer::task_render

6.29.3.2 get_rotation()

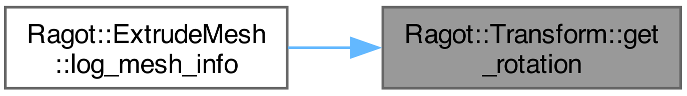
```
vec3 Ragot::Transform::get_rotation () const  [inline]
```

Gets the current rotation of the object.

Returns

vec3 The current rotation in degrees around each axis (x, y, z).

Here is the caller graph for this function:


Ragot::ExtrudeMesh::log_mesh_info
Ragot::Transform::get_rotation

6.29.3.3 get_scale()

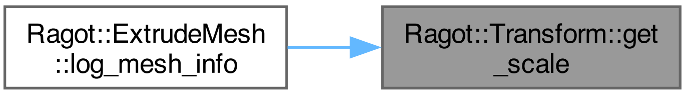
```
vec3 Ragot::Transform::get_scale () const  [inline]
```

Sets the scale of the object uniformly.

Parameters

scale	The new uniform scale factor.

Here is the caller graph for this function:


Ragot::ExtrudeMesh ::log_mesh_info
Ragot::Transform::get _scale

6.29.3.4 get_transform_matrix()

```
virtual mat4 Ragot::Transform::get_transform_matrix ()  [inline], [virtual]
```

Gets the transformation matrix that combines position, rotation, and scale.

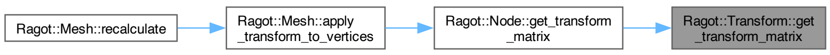
This method recalculates the transformation matrix if it is dirty.

Returns

mat4 The transformation matrix.

Reimplemented in Ragot::Node.

Here is the caller graph for this function:


Ragot::Mesh::recalculate
Ragot::Mesh::apply _transform_to_vertices
Ragot::Node::get_transform _matrix
Ragot::Transform::get _transform_matrix

6.29.3.5 is_dirty()

```
bool Ragot::Transform::is_dirty () const  [inline]
```

Checks if the transformation matrix is dirty (needs recalculation).

Returns

true if the transformation matrix is dirty, false otherwise.

6.29.3.6 rotate()

```
void Ragot::Transform::rotate (
             const float angle,
             const vec3 & axis)  [inline]
```

Rotates the object by a specified angle around a given axis.

Parameters

angle	The angle in degrees to rotate.
axis	The axis around which to rotate, as a vec3.

6.29.3.7 set_position()

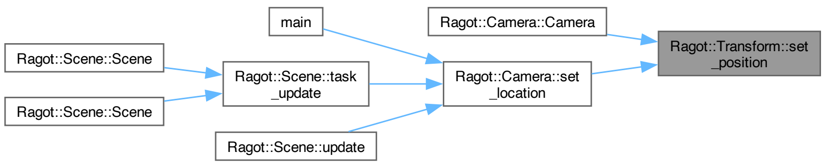
```
void Ragot::Transform::set_position (
             const vec3 & pos)  [inline]
```

Sets the position of the object.

Parameters

pos	The new position as a vec3.

Here is the caller graph for this function:



6.29.3.8 set_rotation()

```
void Ragot::Transform::set_rotation (
             const vec3 & rot)  [inline]
```

Moves the object by a specified vector.

Parameters

offset	The vector by which to move the object.

6.29.3.9 set_scale()

```
void Ragot::Transform::set_scale (
             const vec3 & scale)  [inline]
```

Sets the scale of the object.

Parameters

scale	The new scale as a vec3.

6.29.4 Member Data Documentation

6.29.4.1 dirty

```
bool Ragot::Transform::dirty = true  [protected]
```

Flag indicating whether the transformation matrix needs to be recalculated.

6.29.4.2 position

```
vec3 Ragot::Transform::position  [protected]
```

The position of the object in 3D space.

6.29.4.3 rotation

```
vec3 Ragot::Transform::rotation  [protected]
```

The rotation of the object in degrees around each axis (x, y, z).

6.29.4.4 scale

```
vec3 Ragot::Transform::scale  [protected]
```

The scale of the object in 3D space, default is (1, 1, 1).

6.29.4.5 transform_matrix

```
mat4 Ragot::Transform::transform_matrix  [private]
```

The transformation matrix that combines position, rotation, and scale.

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

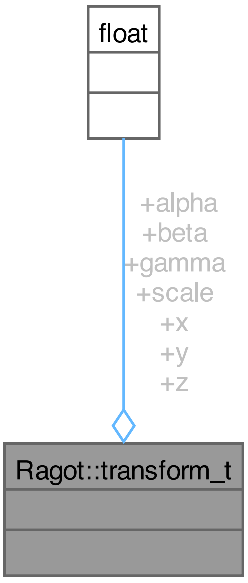
- main/Transform.hpp

6.30 Ragot::transform_t Struct Reference

Represents a transformation in 3D space.

```
#include <CommonTypes.hpp>
```

Collaboration diagram for Ragot::transform_t:



Public Attributes

- float x
- float y
- float z
- float alpha
- float beta
- float gamma
- float scale

6.30.1 Detailed Description

Represents a transformation in 3D space.

This structure holds the position, rotation (in Euler angles), and scale of an object in 3D space.

6.30.2 Member Data Documentation

6.30.2.1 alpha

```
float Ragot::transform_t::alpha
```

6.30.2.2 beta

```
float Ragot::transform_t::beta
```

6.30.2.3 gamma

```
float Ragot::transform_t::gamma
```

6.30.2.4 scale

```
float Ragot::transform_t::scale
```

6.30.2.5 x

```
float Ragot::transform_t::x
```

6.30.2.6 y

```
float Ragot::transform_t::y
```

6.30.2.7 z

```
float Ragot::transform_t::z
```

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

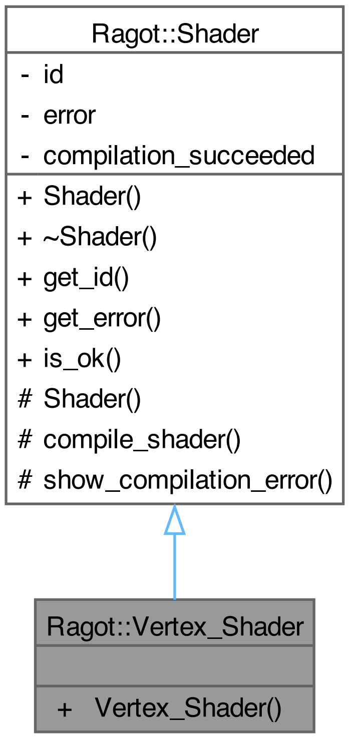
- main/CommonTypes.hpp

6.31 Ragot::Vertex_Shader Class Reference

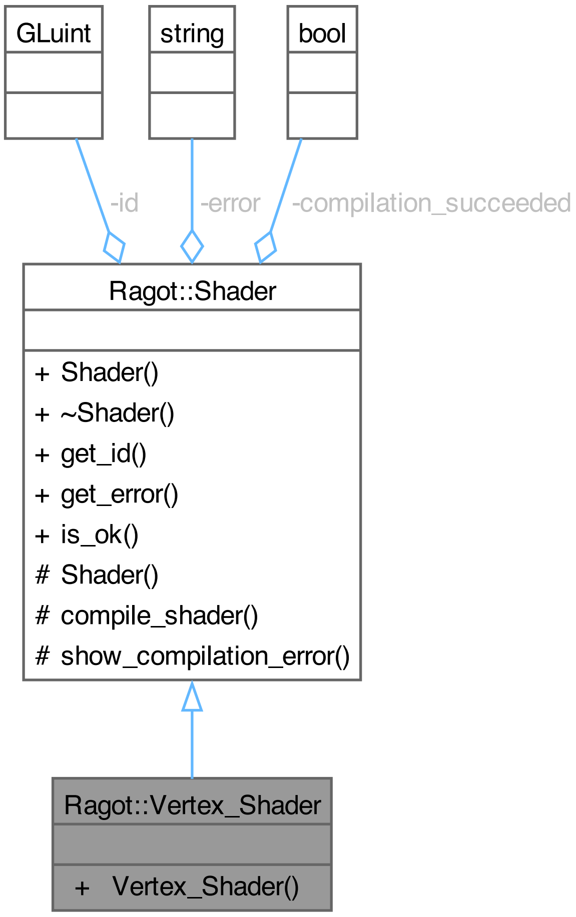
Class for managing an OpenGL vertex shader.

```
#include <Shader_Program.hpp>
```

Inheritance diagram for Ragot::Vertex_Shader:



Collaboration diagram for Ragot::Vertex_Shader:



Public Member Functions

- Vertex_Shader (const vector< string > &source_code)

  Constructor for the Vertex_Shader class.

Public Member Functions inherited from Ragot::Shader

- Shader ()=delete

  Deleted default constructor.
- ∼Shader ()

  Destructor for the Shader class.
- GLuint get_id () const

  Gets the shader ID.
- string ∗ get_error ()

  Gets the compilation error message.
- bool is_ok () const

  Checks if the shader is compiled successfully.

Additional Inherited Members

Protected Member Functions inherited from Ragot::Shader

- Shader (const vector< string > &source_code, GLenum type)

  Constructor for the Shader class.
- GLuint compile_shader ()

  Compiles the shader.
- void show_compilation_error ()

  Displays compilation errors.

6.31.1 Detailed Description

Class for managing an OpenGL vertex shader.

6.31.2 Constructor & Destructor Documentation

6.31.2.1 Vertex_Shader()

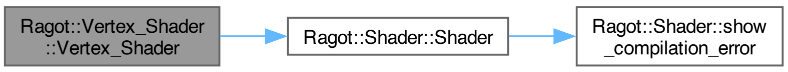
```
Ragot::Vertex_Shader::Vertex_Shader (
            const vector< string > & source_code)  [inline]
```

Constructor for the Vertex_Shader class.

Parameters

source_code	Vector of vertex shader source code.

Here is the call graph for this function:


Ragot::Vertex_Shader ::Vertex_Shader
Ragot::Shader::Shader
Ragot::Shader::show _compilation_error

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

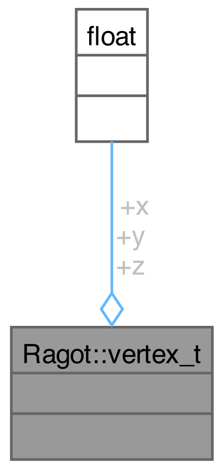
- main/Shader_Program.hpp

6.32 Ragot::vertex_t Struct Reference

Represents a vertex in 3D space.

`#include <CommonTypes.hpp>`

Collaboration diagram for Ragot::vertex_t:



Public Attributes

- float x
- float y
- float z

6.32.1 Detailed Description

Represents a vertex in 3D space.

This structure holds the x, y, and z coordinates of a vertex.

6.32.2 Member Data Documentation

6.32.2.1 x

```
float Ragot::vertex_t::x
```

6.32.2.2 y

```
float Ragot::vertex_t::y
```

6.32.2.3 z

```
float Ragot::vertex_t::z
```

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

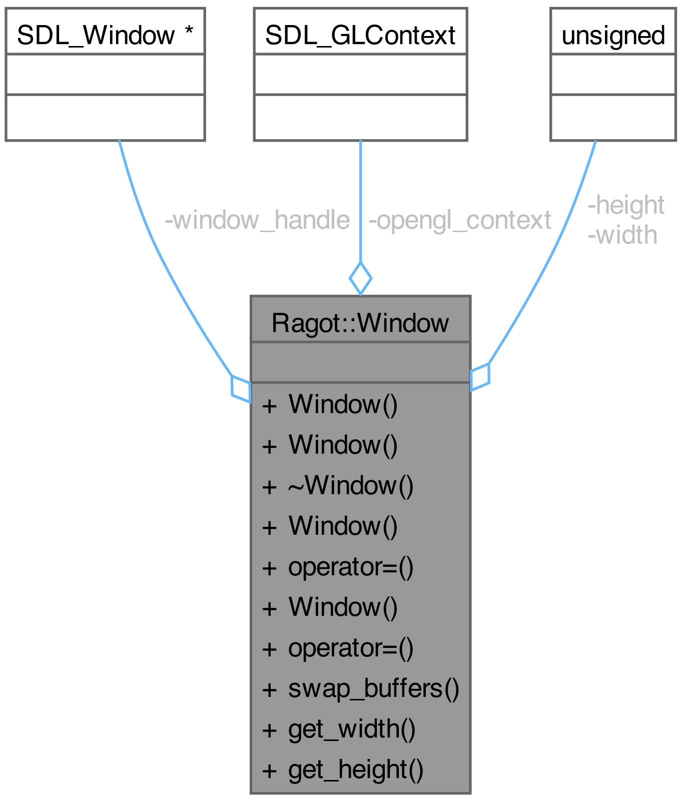
- main/CommonTypes.hpp

6.33 Ragot::Window Class Reference

Class for managing an SDL window with OpenGL context.

```
#include <Window.hpp>
```

Collaboration diagram for Ragot::Window:



Classes

- struct OpenGL_Context_Settings

  Struct for OpenGL context settings.

Public Types

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

    Enum for window position.

Public Member Functions

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

    Constructor for the Window class.
- Window (const char ∗title, int left_x, int top_y, unsigned width, unsigned height, const OpenGL_Context_Settings &context_details)

    Constructor for the Window class.
- ∼Window ()

    Destructor for the Window class.
- Window (const Window &)=delete

    Deleted copy constructor.
- Window & operator= (const Window &)=delete

    Deleted copy assignment operator.
- Window (Window &&other) noexcept

    Move constructor for the Window class.
- Window & operator= (Window &&other) noexcept

    Move assignment operator for the Window class.
- void swap_buffers ()

    Swaps the OpenGL buffers.
- unsigned get_width ()

    Gets the width of the window.
- unsigned get_height ()

    Gets the height of the window.

Private Attributes

- SDL_Window ∗ window_handle

    Handle to the SDL window.
- SDL_GLContext opengl_context

    OpenGL context.
- unsigned width

    Width of the window.
- unsigned height

    Height of the window.

6.33.1 Detailed Description

Class for managing an SDL window with OpenGL context.

6.33.2 Member Enumeration Documentation

6.33.2.1 Position

```
enum Ragot::Window::Position
```

Enum for window position.

Enumerator

UNDEFINED	Undefined position.
CENTERED	Centered position.

6.33.3 Constructor & Destructor Documentation

6.33.3.1 Window() [1/4]

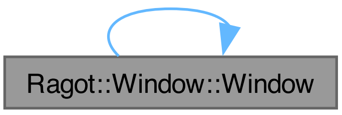
```
Ragot::Window::Window (
            const std::string & title,
            int left_x,
            int top_y,
            unsigned width,
            unsigned height,
            const OpenGL_Context_Settings & context_details)  [inline]
```

Constructor for the Window class.

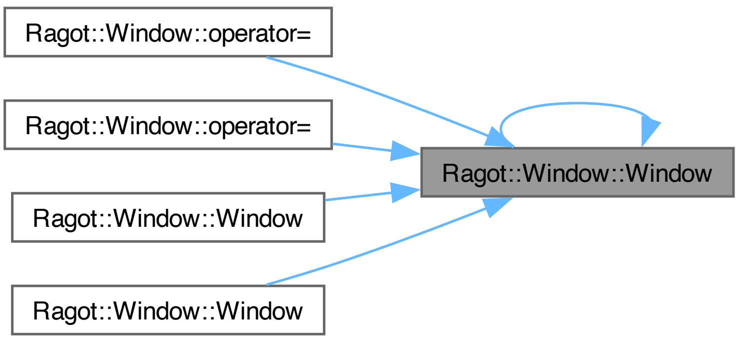
Parameters

title	Title of the window.
left_x	X coordinate of the window position.
top_y	Y coordinate of the window position.
width	Width of the window.
height	Height of the window.
context_details	OpenGL context settings.

Here is the call graph for this function:



Here is the caller graph for this function:



6.33.3.2 Window() [2/4]

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

Constructor for the Window class.

Parameters

title	Title of the window.
left_x	X coordinate of the window position.
top_y	Y coordinate of the window position.
width	Width of the window.
height	Height of the window.
context_details	OpenGL context settings.

6.33.3.3 ~Window()

```
Ragot::Window::~Window ()
```

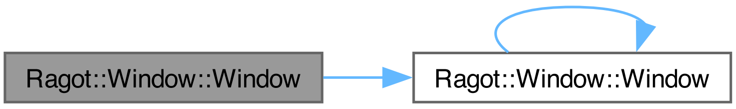
Destructor for the Window class.

6.33.3.4 Window() [3/4]

```
Ragot::Window::Window (
             const Window & )  [delete]
```

Deleted copy constructor.

Here is the call graph for this function:



6.33.3.5 Window() [4/4]

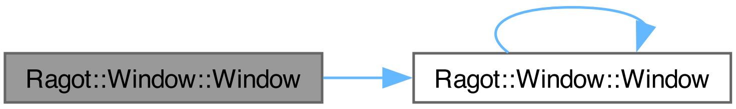
```
Ragot::Window::Window (
             Window && other)  [inline], [noexcept]
```

Move constructor for the Window class.

Parameters

other	Other window to move from.

Here is the call graph for this function:



6.33.4 Member Function Documentation

6.33.4.1 get_height()

```
unsigned Ragot::Window::get_height ()  [inline]
```

Gets the height of the window.

Returns

Height of the window.

6.33.4.2 get_width()

```
unsigned Ragot::Window::get_width ()  [inline]
```

Gets the width of the window.

Returns

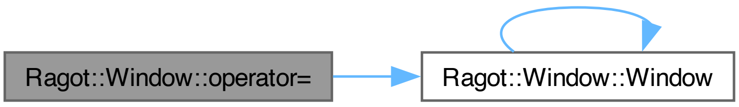
Width of the window.

6.33.4.3 operator=() [1/2]

```
Window & Ragot::Window::operator= (
             const Window & )  [delete]
```

Deleted copy assignment operator.

Here is the call graph for this function:



6.33.4.4 operator=() [2/2]

```
Window & Ragot::Window::operator= (
             Window && other)  [inline], [noexcept]
```

Move assignment operator for the Window class.

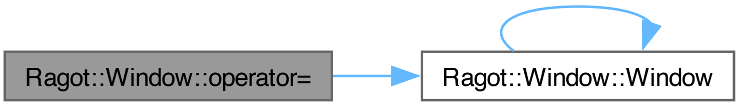
Parameters

other	Other window to move from.

Returns

Reference to the moved window.

Here is the call graph for this function:

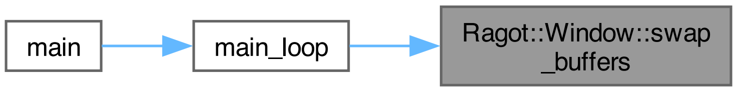


6.33.4.5 swap_buffers()

```
void Ragot::Window::swap_buffers ()  [inline]
```

Swaps the OpenGL buffers.

Here is the caller graph for this function:



6.33.5 Member Data Documentation

6.33.5.1 height

```
unsigned Ragot::Window::height  [private]
```

Height of the window.

6.33.5.2 opengl_context

```
SDL_GLContext Ragot::Window::opengl_context  [private]
```

OpenGL context.

6.33.5.3 width

```
unsigned Ragot::Window::width  [private]
```

Width of the window.

6.33.5.4 window_handle

```
SDL_Window* Ragot::Window::window_handle  [private]
```

Handle to the SDL window.

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

- main/Window.hpp
- main/Window.cpp

Chapter 7

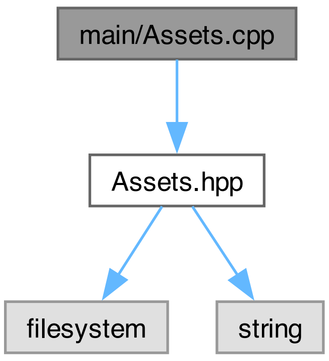
File Documentation

7.1 main/Assets.cpp File Reference

This file implements the Assets class, which manages the asset paths for the application.

```
#include "Assets.hpp"
```

Include dependency graph for Assets.cpp:



Namespaces

- namespace Ragot

Variables

- Assets & Ragot::assets = Assets::instance()

7.1.1 Detailed Description

This file implements the Assets class, which manages the asset paths for the application.

Author

Andrés Ragot (github.com/andresragot)

The Assets class provides a singleton instance to manage asset paths, allowing for easy retrieval of asset files. It initializes the base path based on the executable file path, and provides a method to get the full path of an asset by its name.

Version

1,0

Date

2025-06-01

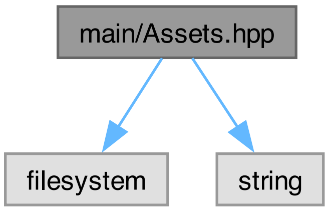
Copyright

7.2 main/Assets.hpp File Reference

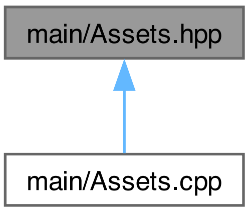
This file defines the Assets class, which manages the asset paths for the application. It provides a singleton instance to access asset paths based on the executable's location. The class initializes the base path for assets depending on whether the build is in debug or release mode. It also provides a method to retrieve the full path of an asset by its name.

```
#include <filesystem>
#include <string>
```

Include dependency graph for Assets.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Ragot::Assets

  Manages the asset paths for the application.

Namespaces

- namespace Ragot

7.2.1 Detailed Description

This file defines the Assets class, which manages the asset paths for the application. It provides a singleton instance to access asset paths based on the executable's location. The class initializes the base path for assets depending on whether the build is in debug or release mode. It also provides a method to retrieve the full path of an asset by its name.

Author

Andrés Ragpt (github.com/andresragot)

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

7.3 Assets.hpp

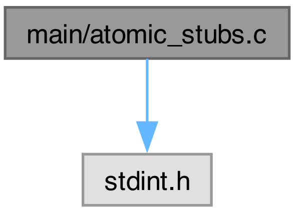
Go to the documentation of this file.

```

#pragma once

#include <filesystem>
#include <string>

namespace Ragot
{
    using std::string;
    using std::filesystem::path;

    class Assets
    {
        path base_path = "./";

    public:

        static Assets & instance()
        {
            static Assets assets;
            return assets;
        }

    private:

        Assets() = default;
        Assets(const Assets&) = delete;
        Assets(const Assets&&) = delete;
        Assets& operator = (const Assets&) = delete;
        Assets& operator = (const Assets&&) = delete;

    public:

        void initialize(const string& executable_file_path)
        {
            #if defined NDEBUG
                base_path = path{ executable_file_path }.parent_path() / "assets";
            #else
                base_path = path{ "../../assets/" };
            #endif
        }

        path get_asset_path(const string & asset_name)
        {
            return base_path/asset_name;
        }

    };

    extern Assets & assets;
}
```

7.4 main/atomic_stubs.c File Reference

This file provides atomic operations using GCC built-in functions.

`#include <stdint.h>`
Include dependency graph for atomic_stubs.c:



Functions

- int __atomic_test_and_set (volatile void ∗ptr, int memorder)
- void __atomic_clear (volatile void ∗ptr, int memorder)

7.4.1 Detailed Description

This file provides atomic operations using GCC built-in functions.

Author

Andrés Ragot (github.com/andresragot)

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

7.4.2 Function Documentation

7.4.2.1 __atomic_clear()

```
void __atomic_clear (
            volatile void * ptr,
            int memorder)
```

7.4.2.2 __atomic_test_and_set()

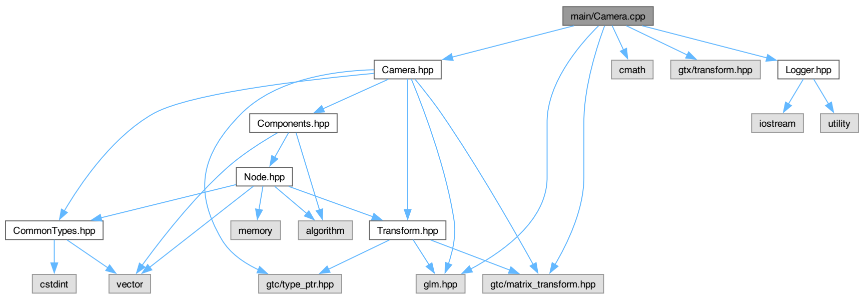
```
int __atomic_test_and_set (
            volatile void * ptr,
            int memorder)
```

7.5 main/Camera.cpp File Reference

This file implements the Camera class, which manages camera properties and operations in a 3D space.

```
#include "Camera.hpp"
#include <cmath>
#include <gtx/transform.hpp>
#include <gtc/matrix_transform.hpp>
#include <glm.hpp>
#include "Logger.hpp"
```

Include dependency graph for Camera.cpp:



Namespaces

- namespace Ragot

Macros

- #define GLM_ENABLE_EXPERIMENTAL

7.5.1 Detailed Description

This file implements the Camera class, which manages camera properties and operations in a 3D space.

Author

Andrés Ragot (github.com/andresragot)

The Camera class provides functionality to log camera information, calculate normals for faces, and determine visibility of faces based on their orientation relative to the camera. It uses GLM (OpenGL Mathematics) for vector and matrix operations, and includes methods for calculating normals and checking face visibility.

Version

0.1

Date

2025-06-01

Copyright

7.5.2 Macro Definition Documentation

7.5.2.1 GLM_ENABLE_EXPERIMENTAL

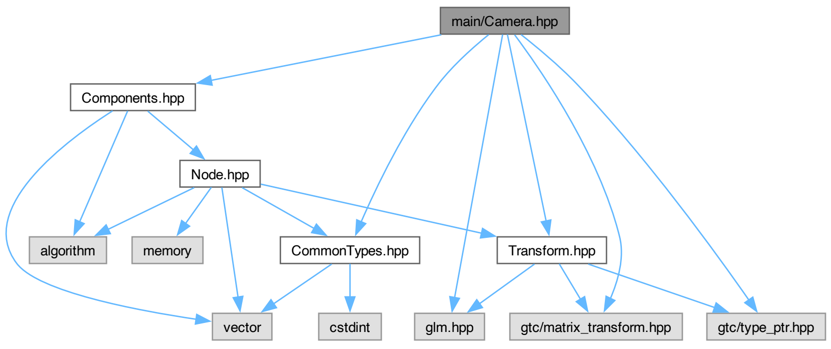
```
#define GLM_ENABLE_EXPERIMENTAL
```

7.6 main/Camera.hpp File Reference

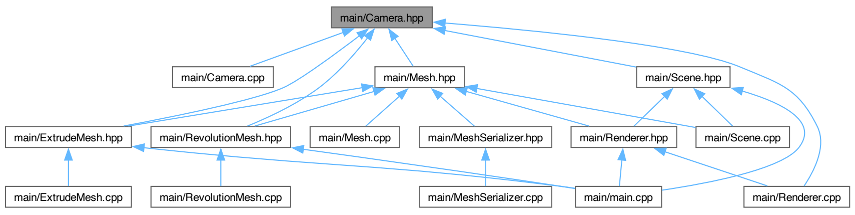
This file implements the Camera class, which manages camera properties and operations in a 3D space.

```
#include "CommonTypes.hpp"
#include "Components.hpp"
#include "Transform.hpp"
#include <glm.hpp>
#include <gtc/matrix_transform.hpp>
#include <gtc/type_ptr.hpp>
```

Include dependency graph for Camera.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Ragot::Camera

  Represents a camera in a 3D space, managing its properties and transformations.

Namespaces

- namespace Ragot

7.6.1 Detailed Description

This file implements the Camera class, which manages camera properties and operations in a 3D space.

Author

Andrés Ragot (github.com/andresragot)

The Camera class provides functionality to log camera information, calculate normals for faces, and determine visibility of faces based on their orientation relative to the camera. It uses GLM (OpenGL Mathematics) for vector and matrix operations, and includes methods for calculating normals and checking face visibility.

Version

0.1

Date

2025-06-01

Copyright

7.7 Camera.hpp

Go to the documentation of this file.

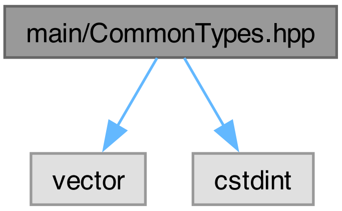
```


#pragma once

#include "CommonTypes.hpp"
#include "Components.hpp"
#include "Transform.hpp"
#include <glm.hpp>
#include <gtc/matrix_transform.hpp>
#include <gtc/type_ptr.hpp>

namespace Ragot
{
    class Camera : public Component
    {
        static const char* CAMERA_TAG;

    public:
        using Matrix4x4 = glm::mat4;

    private:
        float fov;
        float near_plane;
        float far_plane;
        float aspect_ratio;

        glm::vec3 target;

        mutable Matrix4x4 projection_matrix;
        mutable Matrix4x4 view_matrix;
        mutable Matrix4x4 vp_matrix;
        mutable bool projDirty = true;
        mutable bool viewDirty = true;
        mutable bool vpDirty   = true;

    public:
        Camera() = delete;
        Camera(float aspect_ratio = 1.f,
               float near_plane   = 1.f,
               float far_plane    = 100.f,
               float fov_deg      = 60.f)
            : fov(fov_deg), near_plane(near_plane), far_plane(far_plane), aspect_ratio(aspect_ratio)
        {
            set_position(glm::vec3(0.f));
            target = glm::vec3(0.f, 0.f, -1.f);
        }
        ~Camera() = default;

        // --- Getters ---
        float get_fov() const           { return fov; }

        float get_near_plane() const    { return near_plane; }

        float get_far_plane() const     { return far_plane; }

        float get_aspect_ratio() const  { return aspect_ratio; }

        glm::vec3 get_location() const  { return get_position(); }

        glm::vec3 get_target() const    { return target; }

        bool is_dirty() const { return projDirty || viewDirty || vpDirty; }

        // --- Setters (mark dirty) ---

        void set_fov(float deg)            { fov = deg; projDirty = true; vpDirty = true; }

        void set_near_plane(float np)      { near_plane = np; projDirty = true; vpDirty = true; }

        void set_far_plane(float fp)       { far_plane = fp; projDirty = true; vpDirty = true; }

        void set_aspect_ratio(float ar)    { aspect_ratio = ar; projDirty = true; vpDirty = true; }

        void set_location(const glm::vec3 &p) { set_position(p); viewDirty = true; vpDirty = true; }

        void set_target(const glm::vec3 &t)   { target = t; viewDirty = true; vpDirty = true; }

        const Matrix4x4& get_projection_matrix() const
        {
            if (projDirty)
            {
                projection_matrix = glm::perspective(glm::radians(fov), aspect_ratio, near_plane,
    far_plane);
                projDirty = false;
            }
            return projection_matrix;
        }

        const Matrix4x4& get_view_matrix() const
        {
            if (viewDirty)
            {
                view_matrix = glm::lookAt(
                    get_position(),
                    target,
                    glm::vec3(0.f,1.f,0.f)
                );
                viewDirty = false;
            }
            return view_matrix;
        }

        const Matrix4x4& get_vp_matrix() const
        {
            if (vpDirty)
            {
                vp_matrix = get_projection_matrix() * get_view_matrix();
                vpDirty = false;
            }
            return vp_matrix;
        }

        glm::vec3 get_view_direction() const
        {
            return glm::normalize(target - get_position());
        }

        glm::vec3 get_right_direction() const
        {
            return glm::normalize(glm::cross(get_view_direction(), glm::vec3(0.f,1.f,0.f)));
        }

        glm::vec3 get_up_direction() const
        {
            return glm::normalize(glm::cross(get_right_direction(), get_view_direction()));
        }

        glm::vec3 project_to_ndc(const glm::vec4 &worldPos) const
        {
            glm::vec4 clip = get_vp_matrix() * worldPos;
            return (clip.w == 0.f) ? glm::vec3(0.f) : glm::vec3(clip) / clip.w;
        }

        vertex_t calculate_normal(const vertex_t &v1, const vertex_t &v2, const vertex_t &v3);

        bool is_face_visible(const vertex_t &v1, const vertex_t &v2, const vertex_t &v3);

        void log_camera_info() const;

    };
}
```

7.8 main/CommonTypes.hpp File Reference

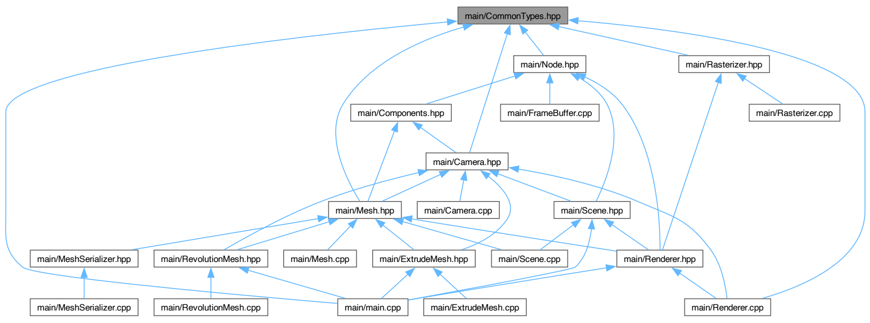
This file defines common types and structures used in the Ragot engine, including camera transformations, mesh information, and rendering flags.

```
#include <vector>
#include <cstdint>
```

Include dependency graph for CommonTypes.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct Ragot::Camera_Transform

  Represents the transformation of a camera in 3D space.
- struct Ragot::transform_t

  Represents a transformation in 3D space.
- struct Ragot::vertex_t

  Represents a vertex in 3D space.
- struct Ragot::face_t

  Represents a face in a 3D mesh.
- struct Ragot::coordinates_t

  Represents 2D coordinates.
- struct Ragot::mesh_info_t

  Represents information about a mesh.

Namespaces

- namespace Ragot

Enumerations

- enum Ragot::render_flag_t : uint8_t { Ragot::RENDER_NONE , Ragot::RENDER_REVOLUTION , Ragot::RENDER_EXTRUDE , Ragot::RENDER_MAX }

  Flags for rendering types.

Variables

- constexpr float Ragot::PI = 3.141592653f

  Mathematical constant PI.

7.8.1 Detailed Description

This file defines common types and structures used in the Ragot engine, including camera transformations, mesh information, and rendering flags.

Author

Andrés Ragot (github.com/andresragot)

The CommonTypes.hpp file provides essential data structures for representing camera transformations, vertex and face definitions, rendering flags, and mesh information. It includes structures for camera position and direction, vertex coordinates, face definitions (triangles and quads), and rendering flags for different mesh types. The file also defines a constant for the mathematical constant PI.

Version

1.0

Date

2025-06-01

Copyright

7.9 CommonTypes.hpp

Go to the documentation of this file.

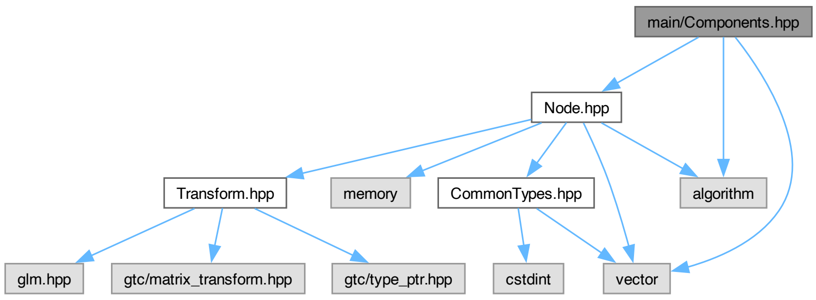
```
#pragma once
#include <vector>
#include <cstdint>

namespace Ragot
{
    struct Camera_Transform
    {
        float x, y, z;            // Posición de la cámara
        float dir_x, dir_y, dir_z; // Dirección de la cámara (vector normalizado)
    };

    struct transform_t
    {
        float x, y, z;
        float alpha, beta, gamma;
        float scale;
    };

    struct vertex_t
    {
        float x;
        float y;
        float z;
    };

    struct face_t
    {
        bool is_quad;
        int v1;
        int v2;
        int v3;
        int v4;
    };


    enum render_flag_t : uint8_t
    {
        RENDER_NONE,
        RENDER_REVOLUTION,
        RENDER_EXTRUDE,
        RENDER_MAX
    };


    struct coordinates_t
    {
        float x;
        float y;
    };


    struct mesh_info_t
    {
        size_t vertex_amount = 0;
        render_flag_t render_flag = RENDER_NONE;
        std::vector < coordinates_t > coordinates;

        mesh_info_t () = default;
        mesh_info_t (std::vector < coordinates_t > & coords, render_flag_t flag)
        : vertex_amount (coords.size ()), render_flag (flag), coordinates (coords)
        {
        }
    };

    constexpr float PI = 3.141592653f;
}
```

7.10 main/Components.hpp File Reference

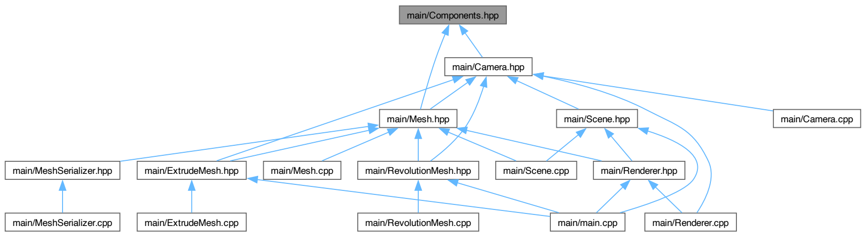
This file defines the Component class, which serves as a base class for components in the Ragot engine.

```
#include "Node.hpp"
#include <vector>
```

```
#include <algorithm>
```
Include dependency graph for Components.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Ragot::Component

    Base class for components in the Ragot engine.

Namespaces

- namespace Ragot

7.10.1 Detailed Description

This file defines the Component class, which serves as a base class for components in the Ragot engine.

Author

Andrés Ragot (github.com/andresragot)

The Component class inherits from Node and provides functionality to manage a collection of components. It allows adding and removing components, and provides access to the list of components.

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

7.11 Components.hpp

Go to the documentation of this file.

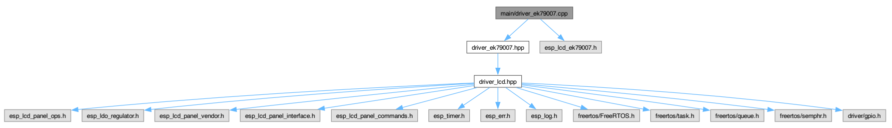
```

#pragma once

#include "Node.hpp"
#include <vector>
#include <algorithm>


namespace Ragot
{
    class Component : public Node
    {
    public:

        Component() = default;

        virtual ~Component() = default;

        Component(const Component & ) = delete;

        Component(const Component &&) = delete;

        Component & operator = (const Component & ) = delete;

        Component & operator = (const Component &&) = delete;

    protected:

        std::vector < std::shared_ptr < Component > > components;

    public:

        void add_component(std::shared_ptr < Component > component)
        {
            if (component)
            {
                components.emplace_back(component);
                component->parent = this;
            }
        }

        void remove_component (std::shared_ptr < Component > component)
        {
            if (component)
            {
                auto it = std::remove(components.begin(), components.end(), component);
                if (it != components.end())
                {
                    components.erase(it, components.end());
                    component->parent = nullptr;
                }
            }
        }

        const std::vector<std::shared_ptr < Component > > get_components() const { return components;
    }

    };
}
```

7.12 main/driver_ek79007.cpp File Reference

This file implements the DriverEK79007 class, which manages the initialization and operation of the EK79007 LCD panel driver. The DriverEK79007 class inherits from the DriverLCD class and provides methods to initialize, deinitialize, and send frame buffers to the LCD panel.

```
#include "driver_ek79007.hpp"
#include "esp_lcd_ek79007.h"
```

Include dependency graph for driver_ek79007.cpp:



Namespaces

- namespace Ragot

Functions

- static bool Ragot::panel_refresh_callback (esp_lcd_panel_handle_t panel, esp_lcd_dpi_panel_event_data←_t ∗edata, void ∗user_ctx)

Variables

- static const char ∗ Ragot::TAG = "DriverEK79007"

7.12.1 Detailed Description

This file implements the DriverEK79007 class, which manages the initialization and operation of the EK79007 LCD panel driver. The DriverEK79007 class inherits from the DriverLCD class and provides methods to initialize, deinitialize, and send frame buffers to the LCD panel.

Author

Andrés Ragot (github.com/andresragot)

Version

1.0

Date

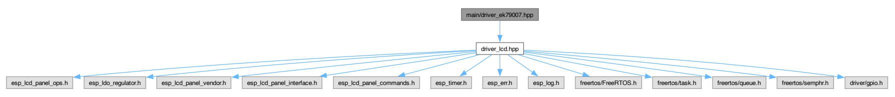
2025-04-17

Copyright

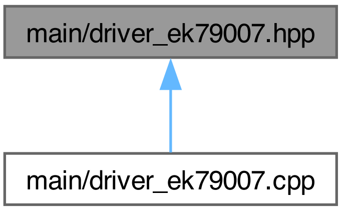
7.13 main/driver_ek79007.hpp File Reference

This file implements the DriverEK79007 class, which manages the initialization and operation of the EK79007 LCD panel driver. The DriverEK79007 class inherits from the DriverLCD class and provides methods to initialize, deinitialize, and send frame buffers to the LCD panel.

```
#include "driver_lcd.hpp"
```
Include dependency graph for driver_ek79007.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Ragot::DriverEK79007

    Driver for the EK79007 LCD panel.

Namespaces

- namespace Ragot

7.13.1 Detailed Description

This file implements the DriverEK79007 class, which manages the initialization and operation of the EK79007 LCD panel driver. The DriverEK79007 class inherits from the DriverLCD class and provides methods to initialize, deinitialize, and send frame buffers to the LCD panel.

Author

Andrés Ragot (github.com/andresragot)

Version

1.0

Date

2025-04-17

Copyright

7.14 driver_ek79007.hpp

Go to the documentation of this file.

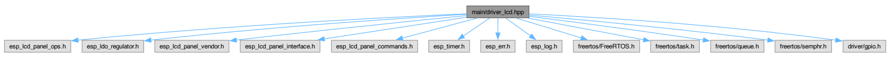
```

#pragma once

#include "driver_lcd.hpp"


namespace Ragot
{
    class DriverEK79007 : public DriverLCD
    {
    public:

        DriverEK79007();

       ~DriverEK79007() override;

        esp_err_t init(gpio_num_t reset_pin, gpio_num_t bk_pin) override;

        esp_err_t deinit() override;

        esp_err_t set_pixel(uint32_t x, uint32_t y, uint32_t color) override {return ESP_FAIL;};

        esp_err_t send_frame_buffer(const void * frame_buffer) override;

        IRAM_ATTR bool refresh_frame_buffer( esp_lcd_panel_handle_t panel,
esp_lcd_dpi_panel_event_data_t * edata, void * user_ctx);

    private:
        uint16_t panel_clk_freq_mhz;
        uint32_t hsync_pulse_width;
        uint32_t hsync_back_porch;
        uint32_t hsync_front_porch;
        uint32_t vsync_pulse_width;
        uint32_t vsync_back_porch;
        uint32_t vsync_front_porch;
        uint8_t  mipi_lane_num;
        uint16_t mipi_dsi_max_data_rate_mbps;

    private:
        void bsp_enable_dsi_phy_power();

        void bsp_init_lcd_backlight(gpio_num_t bk_pin);

    public:
        SemaphoreHandle_t refresh_semaphore;
    };
}
```

7.15 main/driver_lcd.hpp File Reference

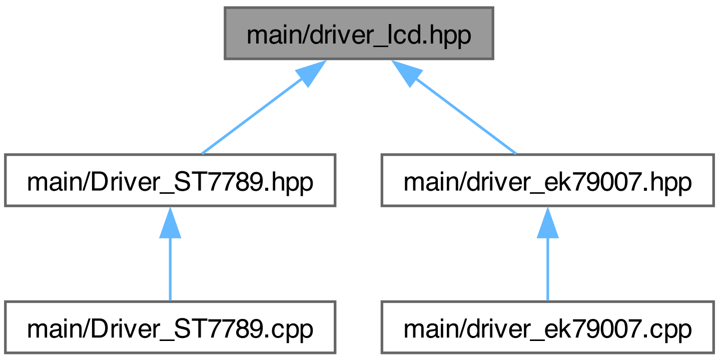
This file defines the DriverLCD class, which serves as a base class for LCD drivers in the Ragot engine.

```
#include "esp_lcd_panel_ops.h"
#include "esp_ldo_regulator.h"
#include "esp_lcd_panel_vendor.h"
#include "esp_lcd_panel_interface.h"
#include "esp_lcd_panel_commands.h"
#include "esp_timer.h"
#include "esp_err.h"
#include "esp_log.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/queue.h"
#include "freertos/semphr.h"
#include "driver/gpio.h"
```

Include dependency graph for driver_lcd.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Ragot::DriverLCD

  Base class for LCD drivers.

Namespaces

- namespace Ragot

7.15.1 Detailed Description

This file defines the DriverLCD class, which serves as a base class for LCD drivers in the Ragot engine.

Author

Andrés Ragot (github.com/andresragot)

The DriverLCD class provides an interface for initializing, deinitializing, setting pixels, and sending frame buffers to an LCD panel. It includes methods to get the width, height, pixel format, and handler of the LCD panel. The class is designed to be inherited by specific LCD driver implementations, such as the EK79007 driver.

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025

7.16 driver_lcd.hpp

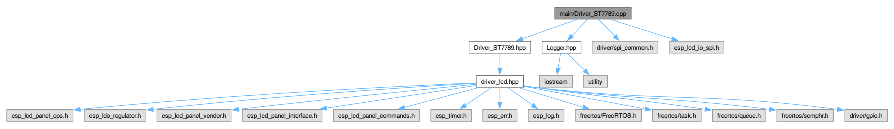
Go to the documentation of this file.

```


#pragma once

#include "esp_lcd_panel_ops.h"
#include "esp_ldo_regulator.h"
#include "esp_lcd_panel_vendor.h"
#include "esp_lcd_panel_interface.h"
#include "esp_lcd_panel_commands.h"
#include "esp_timer.h"
#include "esp_err.h"
#include "esp_log.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/queue.h"
#include "freertos/semphr.h"
#include "driver/gpio.h"
#ifdef CONFIG_IDF_TARGET_ESP32P4
#include "esp_lcd_mipi_dsi.h"
#endif

namespace Ragot
{
      class DriverLCD
      {
      public:

            DriverLCD() = default;

            virtual ~DriverLCD() = default;

            virtual esp_err_t init(gpio_num_t reset_pin, gpio_num_t bk_pin) = 0;

            virtual esp_err_t deinit() = 0;

            virtual esp_err_t set_pixel(uint32_t x, uint32_t y, uint32_t color) = 0;

            virtual esp_err_t send_frame_buffer( const void * frame_buffer) = 0;

            const size_t  get_width() const { return width; }

                  size_t  get_width()       { return width; }

            const size_t get_height() const { return height; }

                  size_t get_height()       { return height; }


            const lcd_color_rgb_pixel_format_t get_pixel_format() const { return pixel_format; }

                  lcd_color_rgb_pixel_format_t get_pixel_format()       { return pixel_format; }

            const esp_lcd_panel_handle_t get_handler() const { return handler; }

                  esp_lcd_panel_handle_t get_handler()       { return handler; }

            const bool is_initialized() const { return initialized; }

                  bool is_initialized()       { return initialized; }

      protected:
            bool initialized = false;
            size_t width;
            size_t height;
            lcd_color_rgb_pixel_format_t pixel_format;
            esp_lcd_panel_handle_t handler;
    };
}
```

7.17 main/Driver_ST7789.cpp File Reference

This file implements the Driver_ST7789 class, which manages the initialization and operation of the ST7789 LCD panel driver. The Driver_ST7789 class inherits from the DriverLCD class and provides methods to initialize, deinitialize, and send frame buffers to the LCD panel.

```
#include "Driver_ST7789.hpp"
#include "Logger.hpp"
#include "driver/spi_common.h"
#include "esp_lcd_io_spi.h"
```
Include dependency graph for Driver_ST7789.cpp:



Namespaces

- namespace Ragot

Functions

- static bool Ragot::panel_refresh_callback (esp_lcd_panel_io_handle_t panel, esp_lcd_panel_io_event_↩ data_t ∗edata, void ∗user_ctx)

7.17.1 Detailed Description

This file implements the Driver_ST7789 class, which manages the initialization and operation of the ST7789 LCD panel driver. The Driver_ST7789 class inherits from the DriverLCD class and provides methods to initialize, deinitialize, and send frame buffers to the LCD panel.

Author

Andrés Ragot (github.com/andresragot)

Version

1.0

Date

2025-06-01

Copyright

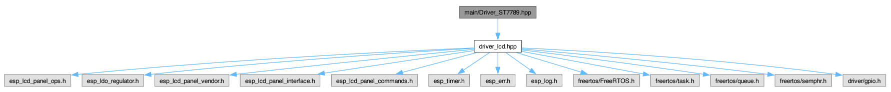
Copyright (c) 2025 MIT License

7.18 main/Driver_ST7789.hpp File Reference

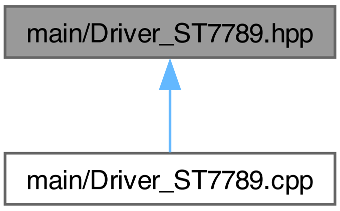
This file implements the Driver_ST7789 class, which manages the initialization and operation of the ST7789 LCD panel driver. The Driver_ST7789 class inherits from the DriverLCD class and provides methods to initialize, deinitialize, and send frame buffers to the LCD panel.

```
#include "driver_lcd.hpp"
```

Include dependency graph for Driver_ST7789.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Ragot::Driver_ST7789

  Driver for the ST7789 LCD panel.

Namespaces

- namespace Ragot

7.18.1 Detailed Description

This file implements the Driver_ST7789 class, which manages the initialization and operation of the ST7789 LCD panel driver. The Driver_ST7789 class inherits from the DriverLCD class and provides methods to initialize, deinitialize, and send frame buffers to the LCD panel.

Author

Andrés Ragot (github.com/andresragot)

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

7.19 Driver_ST7789.hpp

Go to the documentation of this file.

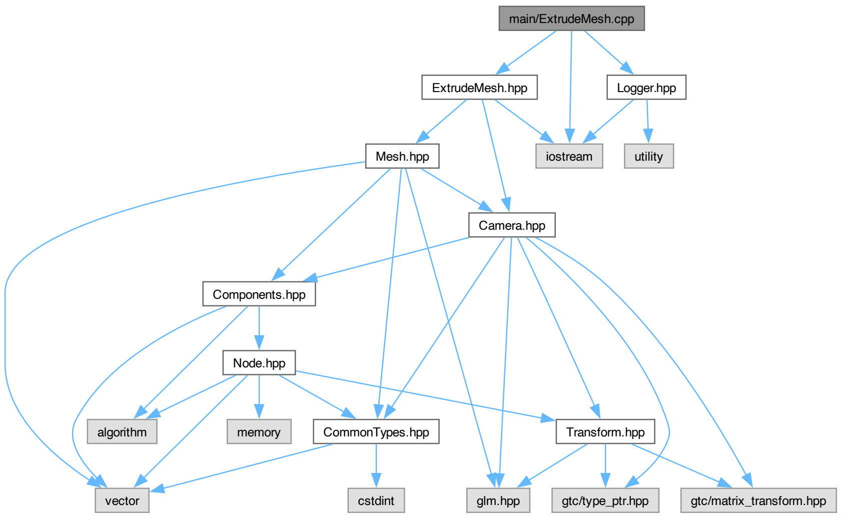
```

#pragma once
#include "driver_lcd.hpp"

namespace Ragot
{
    class Driver_ST7789 : public DriverLCD
    {
    private:
        static constexpr const char * TAG = "[Driver_ST7789]...";

    public:
        Driver_ST7789 ();

        virtual ~Driver_ST7789() = default;

        esp_err_t init(gpio_num_t reset_pin, gpio_num_t bk_pin) override;

        esp_err_t deinit() override;

        esp_err_t set_pixel(uint32_t x, uint32_t y, uint32_t color) override {return ESP_FAIL;};

        esp_err_t send_frame_buffer( const void * frame_buffer) override;

        IRAM_ATTR bool refresh_frame_buffer(void * user_ctx);

    private:
        void bsp_init_lcd_backlight(gpio_num_t bk_pin);

    public:
        SemaphoreHandle_t refresh_semaphore;
    };
}
```

7.20 main/ExtrudeMesh.cpp File Reference

This file implements the ExtrudeMesh class, which manages the extrusion of a mesh in 3D space. The ExtrudeMesh class inherits from the Mesh class and provides methods to generate vertices and faces for the extruded mesh. It also includes methods for culling faces based on the camera's view direction and logging mesh information.

```
#include "ExtrudeMesh.hpp"
#include <iostream>
#include "Logger.hpp"
```

Include dependency graph for ExtrudeMesh.cpp:



Namespaces

- namespace Ragot

7.20.1 Detailed Description

This file implements the ExtrudeMesh class, which manages the extrusion of a mesh in 3D space. The ExtrudeMesh class inherits from the Mesh class and provides methods to generate vertices and faces for the extruded mesh. It also includes methods for culling faces based on the camera's view direction and logging mesh information.

Author

Andrés Ragot (github.com/andresragot)

The ExtrudeMesh class is designed to create a 3D mesh by extruding a 2D shape along a specified height. It uses the GLM library for vector and matrix operations, and includes functionality for face culling based on the camera's view direction. The class also provides a method to log detailed information about the mesh, including its position, rotation, scale, and vertex data.

Version

1.0

Date

2025-06-01

Copyright

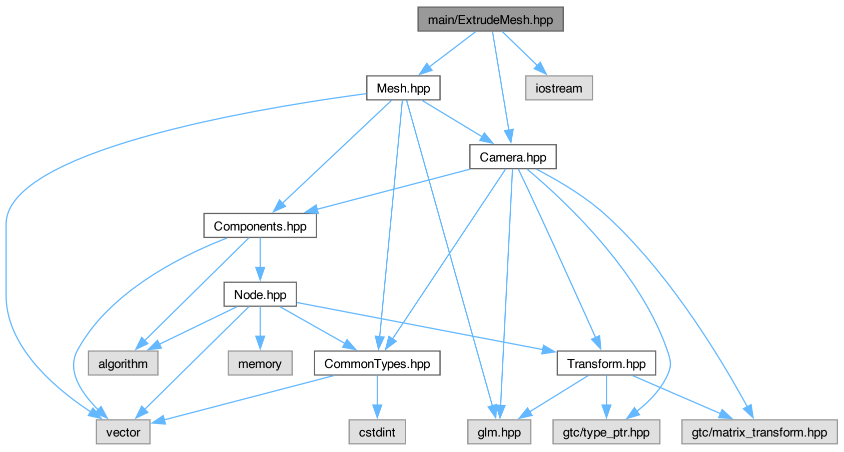
Copyright (c) 2025 MIT License

7.21 main/ExtrudeMesh.hpp File Reference

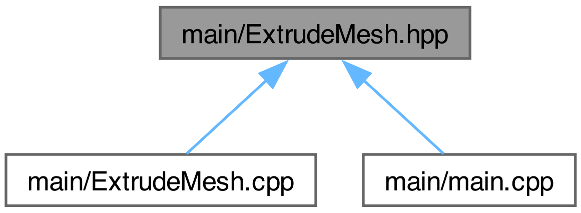
This file implements the ExtrudeMesh class, which manages the extrusion of a mesh in 3D space. The ExtrudeMesh class inherits from the Mesh class and provides methods to generate vertices and faces for the extruded mesh. It also includes methods for culling faces based on the camera's view direction and logging mesh information.

```
#include "Mesh.hpp"
#include "Camera.hpp"
#include <iostream>
```

Include dependency graph for ExtrudeMesh.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Ragot::ExtrudeMesh

  Represents a 3D mesh created by extruding a 2D shape along a specified height. This class inherits from the Mesh class and provides methods to generate vertices and faces for the extruded mesh. It also includes methods for culling faces based on the camera's view direction and logging mesh information.

Namespaces

- namespace Ragot

7.21.1 Detailed Description

This file implements the ExtrudeMesh class, which manages the extrusion of a mesh in 3D space. The ExtrudeMesh class inherits from the Mesh class and provides methods to generate vertices and faces for the extruded mesh. It also includes methods for culling faces based on the camera's view direction and logging mesh information.

Author

Andrés Ragot (github.com/andresragot)

The ExtrudeMesh class is designed to create a 3D mesh by extruding a 2D shape along a specified height. It uses the GLM library for vector and matrix operations, and includes functionality for face culling based on the camera's view direction. The class also provides a method to log detailed information about the mesh, including its position, rotation, scale, and vertex data.

Version

1.0

Date

2025-06-01

7.22 ExtrudeMesh.hpp

Go to the documentation of this file.

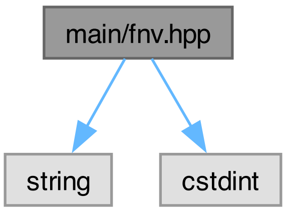
```


#pragma once

#include "Mesh.hpp"
#include "Camera.hpp"
#include <iostream>

namespace Ragot
{
    class ExtrudeMesh : public Mesh
    {
    protected:
        static const char* EXTRUDE_TAG;

        float height = 1.0f;

        bool faces_can_be_quads = false;

        const Camera & cam;

        glm::vec4 planes[4];

        glm::vec3 camPos;

    public:
        ExtrudeMesh (mesh_info_t & mesh_info, const Camera & cam) : Mesh (mesh_info), cam (cam)
        {
            vertices.reserve (mesh_info.coordinates.size() * 2    );
               faces.reserve (mesh_info.coordinates.size() * 3 - 3);
            // Si son 14 vertices -> 39 - 28 = 11
            // Si son 12 vertices -> 33 - 24 = 9
            // Si son 10 vertices -> 27 - 20 = 7
            // Si son  8 vertices ->  9 - 16 = -7
            // Si son  6 vertices -> 15 - 12 = 3
            // Si son 4 vertices ->   9 -  5 = 4

            // % 8 porque como están las coordenadas duplicadas...
            faces_can_be_quads = (mesh_info.vertex_amount % 8 == 0 || mesh_info.vertex_amount == 4);
            generate_vertices();
            generate_faces();

            std::cout « "Etrude Vertices: " « vertices.size() « std::endl;
            std::cout « "Extrude Faces: " « faces.size() « std::endl;
        }

       ~ExtrudeMesh() = default;
```

```

        void generate_vertices () override;

        void generate_faces    () override;

        bool are_vertices_coplanar (const glm::fvec4 & v1, const glm::fvec4 & v2, const glm::fvec4 &
    v3, const glm::fvec4 & v4, float tolerance = 0.1f);

        void log_mesh_info() const;
    };
}
```

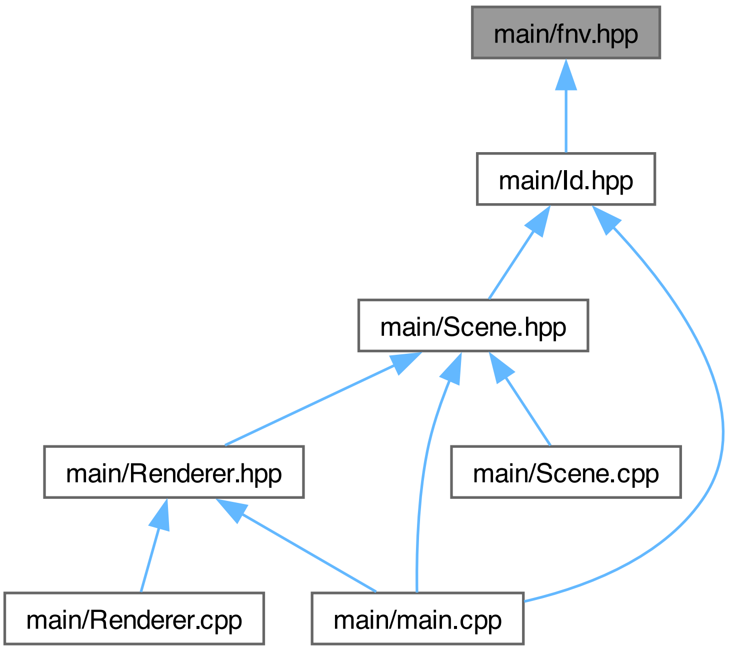
7.23 main/fnv.hpp File Reference

```
#include <string>
#include <cstdint>
```

Include dependency graph for fnv.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace basics
- namespace basics::internal

Macros

- #define FNV(X)
- #define FNV32(X)
- #define FNV64(X)

Functions

- template<size_t LENGTH>
  constexpr uint32_t basics::internal::static_fnv32 (const char ∗chars)
- template<> constexpr uint32_t basics::internal::static_fnv32< 1 > (const char ∗)
- template<size_t LENGTH>
  constexpr uint64_t basics::internal::static_fnv64 (const char ∗chars)
- template<> constexpr uint64_t basics::internal::static_fnv64< 1 > (const char ∗)
- template<size_t LENGTH>
  constexpr uint32_t basics::static_fnv32 (const char(&chars)[LENGTH])
- template<size_t LENGTH>
  constexpr uint64_t basics::static_fnv64 (const char(&chars)[LENGTH])

- template<size_t LENGTH>
  constexpr unsigned basics::static_fnv (const char(&chars)[LENGTH])
- template<size_t LENGTH>
  uint32_t basics::fnv32 (const char(&chars)[LENGTH])
- uint32_t basics::fnv32 (const std::string &s)
- constexpr unsigned operator""_fnv (const char ∗c)

Variables

- constexpr uint32_t basics::internal::fnv_basis_32 = 0x811c9dc5u
- constexpr uint32_t basics::internal::fnv_prime_32 = 0x01000193u
- constexpr uint64_t basics::internal::fnv_basis_64 = 0xcbf29ce484222325u
- constexpr uint64_t basics::internal::fnv_prime_64 = 0x00000100000001b3u

7.23.1 Macro Definition Documentation

7.23.1.1 FNV

```
#define FNV(
              X)
```

Value:

```
basics::static_fnv   (#X)
```

7.23.1.2 FNV32

```
#define FNV32(
              X)
```

Value:

```
basics::static_fnv32 (#X)
```

7.23.1.3 FNV64

```
#define FNV64(
              X)
```

Value:

```
basics::static_fnv64 (#X)
```

7.23.2 Function Documentation

7.23.2.1 operator""""_fnv()

```
unsigned operator""_fnv (
             const char * c)  [constexpr]
```

7.24 fnv.hpp

Go to the documentation of this file.

```
/*
 * FNV
 * Copyright © 2017+ Ángel Rodríguez Ballesteros
 *
 * Distributed under the Boost Software License, version  1.0
 * See documents/LICENSE.TXT or www.boost.org/LICENSE_1_0.txt
 *
 * angel.rodriguez@esne.edu
 *
 * C1712171830
 */

#ifndef BASICS_FNV_HEADER
#define BASICS_FNV_HEADER

    #include <string>
    #include <cstdint>

    namespace basics
    {

        namespace internal
        {

            constexpr uint32_t fnv_basis_32 = 0x811c9dc5u;
            constexpr uint32_t fnv_prime_32 = 0x01000193u;
            constexpr uint64_t fnv_basis_64 = 0xcbf29ce484222325u;
            constexpr uint64_t fnv_prime_64 = 0x00000100000001b3u;

            template< size_t LENGTH >
            constexpr uint32_t static_fnv32 (const char * chars)
            {
                return (static_fnv32< LENGTH - 1 > (chars) ^ chars[LENGTH - 2]) * fnv_prime_32;
            }

            template< >
            constexpr uint32_t static_fnv32< 1 > (const char * )
            {
                return fnv_basis_32;
            }

            template< size_t LENGTH >
            constexpr uint64_t static_fnv64 (const char * chars)
            {
                return (static_fnv64< LENGTH - 1 > (chars) ^ chars[LENGTH - 2]) * fnv_prime_64;
            }

            template< >
            constexpr uint64_t static_fnv64< 1 > (const char * )
            {
                return fnv_basis_64;
            }

        }

        // ------------------------------------------------------------------------------------------

        template< size_t LENGTH >
        constexpr uint32_t static_fnv32 (const char (& chars)[LENGTH])
        {
            return internal::static_fnv32< LENGTH > (chars);
        }

        // ------------------------------------------------------------------------------------------

        template< size_t LENGTH >
        constexpr uint64_t static_fnv64 (const char (& chars)[LENGTH])
        {
            return internal::static_fnv64< LENGTH > (chars);
        }

        // ------------------------------------------------------------------------------------------

        #if BASICS_INT_SIZE == 4

            template< size_t LENGTH >
            constexpr unsigned static_fnv (const char (& chars)[LENGTH])
            {
                return internal::static_fnv32< LENGTH > (chars);
            }

        #else
```

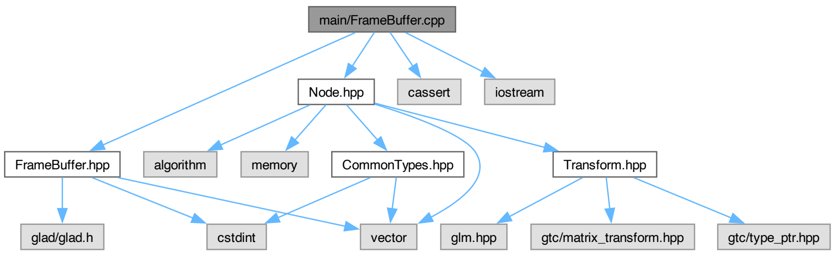
```

            template< size_t LENGTH >
            constexpr unsigned static_fnv (const char (& chars)[LENGTH])
            {
                return static_cast < unsigned > (internal::static_fnv64< LENGTH > (chars));
            }

        #endif

        // ------------------------------------------------------------------------------------------

        template< size_t LENGTH >
        uint32_t fnv32 (const char (& chars)[LENGTH])
        {
            uint32_t hash = internal::fnv_basis_32;

            for (size_t index = 0; index < LENGTH; ++index)
            {
                hash ^= chars[index];  // Use array indexing instead
                hash *= internal::fnv_prime_32;
            }

            return hash;
        }

        inline uint32_t fnv32 (const std::string & s)
        {
            uint32_t hash = internal::fnv_basis_32;

            for (auto c : s)
            {
                hash ^= c;
                hash *= internal::fnv_prime_32;
            }

            return hash;
        }

    }

    constexpr unsigned operator "" _fnv (const char * c)
    {
        return c ? 1 : operator "" _fnv ("2");
    }

    // ----------------------------------------------------------------------------------------------

    #define FNV(X)   basics::static_fnv   (#X)
    #define FNV32(X) basics::static_fnv32 (#X)
    #define FNV64(X) basics::static_fnv64 (#X)

#endif
```

7.25 main/FrameBuffer.cpp File Reference

This file implements the FrameBuffer class, which manages a frame buffer for rendering graphics.

```
#include "FrameBuffer.hpp"
#include <cassert>
#include "Node.hpp"
#include <iostream>
```

Include dependency graph for FrameBuffer.cpp:



Namespaces

- namespace Ragot

7.25.1 Detailed Description

This file implements the FrameBuffer class, which manages a frame buffer for rendering graphics.

Author

Andrés Ragot (github.com/andresragot)

The FrameBuffer class provides methods to create a frame buffer, swap buffers, clear the buffer, fill it with a color, set and get pixels, and manage OpenGL textures. It supports both single and double buffering modes. The class is designed to be used in graphics applications where rendering performance and buffer management are crucial.

Version

0.1

Date

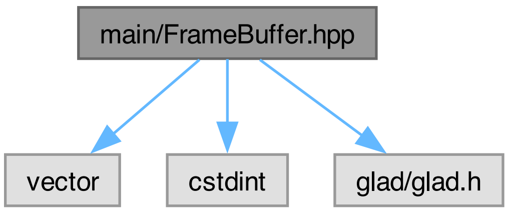
2025-06-01

Copyright

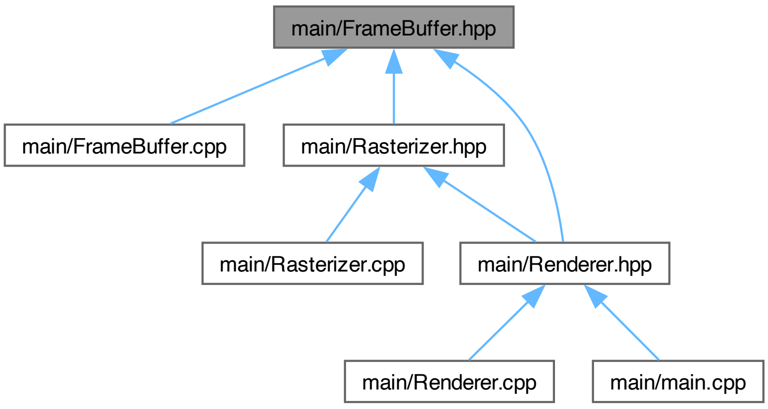
7.26 main/FrameBuffer.hpp File Reference

```
#include <vector>
#include <cstdint>
#include <glad/glad.h>
```

Include dependency graph for FrameBuffer.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Ragot::FrameBuffer< Color >

  Class to manage a frame buffer for rendering graphics.

Namespaces

- namespace Ragot

Typedefs

- using Ragot::RGB565 = uint16_t
- using Ragot::RGB888 = uint32_t
- using Ragot::RGBA8888 = uint32_t
- using Ragot::RGB8 = uint8_t

  Color Index.

Enumerations

- enum Ragot::Buffer : uint8_t { Ragot::CURRENT_BUFFER = (1 << 0) , Ragot::NEXT_BUFFER = (1 << 1) , Ragot::MAX_BUFFER = (1 << 2) }

  Enum to represent the different buffers in a frame buffer.

7.27 FrameBuffer.hpp

Go to the documentation of this file.

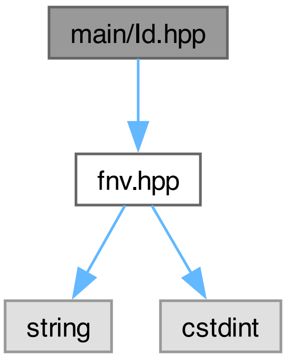
```


#pragma once
#include <vector>
#include <cstdint>
#if ESP_PLATFORM == 1
#include "RamAllocator.hpp"
#else
#include <glad/glad.h>
#endif

namespace Ragot
{
    using RGB565   = uint16_t;
    using RGB888   = uint32_t;
    using RGBA8888 = uint32_t;
    using RGB8     = uint8_t ;

    enum Buffer : uint8_t
    {
        CURRENT_BUFFER  = ( 1 « 0),
        NEXT_BUFFER     = ( 1 « 1),
        MAX_BUFFER      = ( 1 « 2)
    };

    template <typename Color>
    class FrameBuffer
    {
    public:
        using TYPE = Color;
        #if ESP_PLATFORM == 1
        using ColorVector = std::vector < Color, PSRAMAllocator< Color, MALLOC_CAP_8BIT > >;
        #else
        using ColorVector = std::vector < Color >;
        #endif

    private:


        bool double_buffer;
        size_t width;
        size_t height;
        Color color;
        ColorVector   buffer_1;
        ColorVector   buffer_2;
        ColorVector * current_buffer;
        ColorVector * next_buffer;


    public:
        FrameBuffer (size_t width, size_t height, bool double_buffer);

        FrameBuffer () = delete;

       ~FrameBuffer () = default;
```

```

        // Nada más queremos que haya un FrameBuffer por ahora.

        FrameBuffer (const FrameBuffer &) = delete;

        FrameBuffer (const FrameBuffer &&) = delete;

        FrameBuffer & operator = (const FrameBuffer &) = delete;

        FrameBuffer & operator = (const FrameBuffer &&) = delete;

        void swap_buffers();

        void clear_buffer( Buffer buffer_to_clear = NEXT_BUFFER );

        void fill (Color color = 0, Buffer buffer_to_fill = NEXT_BUFFER);

        void set_pixel (size_t x, size_t y, Color color);

        void set_pixel (size_t offset, Color color);

        void set_pixel (size_t offset);

        void set_color (Color color);

        Color get_pixel (size_t x, size_t y) const;

        size_t get_width ()        { return width;  }

        size_t get_width ()  const { return width;  }

        size_t get_height ()       { return height; }

        size_t get_height () const { return height; }

        const Color * get_buffer() const { return current_buffer->data(); }

              Color * get_buffer()       { return current_buffer->data(); }

        inline void blit_to_window () const
        {
            // Implementar la función para blit a la ventana
            std::copy(current_buffer->begin(), current_buffer->end(), next_buffer->begin());
        }


        #if ESP_PLATFORM != 1

        void initGLTexture();

        void sendGL() const;

        static GLenum getGLFormat();

        static GLenum getGLType();

        GLuint getGLTex () const { return gl_tex; }

    private:
        GLuint gl_tex = 0;
        #endif
    };
}
```

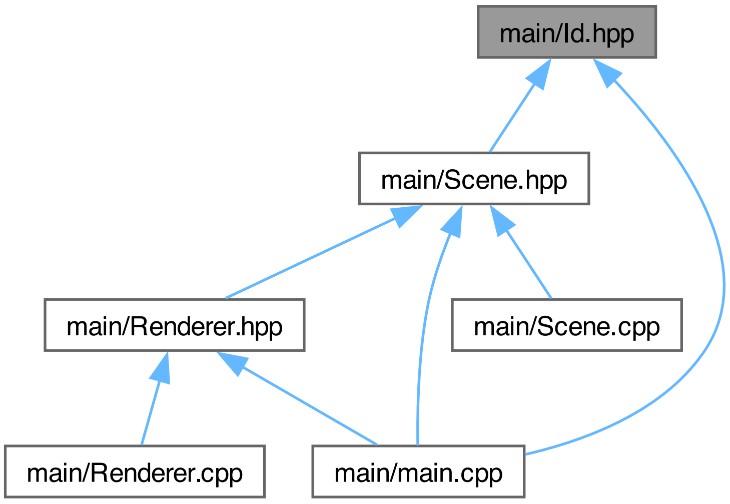
7.28 main/Id.hpp File Reference

```
#include "fnv.hpp"
```

Include dependency graph for Id.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace basics

Macros

- #define ID(X)

Typedefs

- typedef unsigned int basics::Id

7.28.1 Macro Definition Documentation

7.28.1.1 ID

```
#define ID(
                X)
```

Value:

```
FNV(X)
```

7.29 Id.hpp

Go to the documentation of this file.

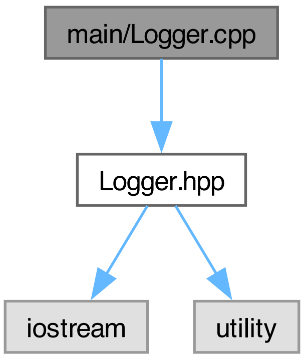
```
/*
 * ID
 * Copyright © 2017+ Ángel Rodríguez Ballesteros
 *
 * Distributed under the Boost Software License, version  1.0
 * See documents/LICENSE.TXT or www.boost.org/LICENSE_1_0.txt
 *
 * angel.rodriguez@esne.edu
 *
 * C1712211447
 */

#ifndef BASICS_ID_HEADER
#define BASICS_ID_HEADER

    #include "fnv.hpp"

    #define ID(X) FNV(X)

    namespace basics
    {

        typedef unsigned int Id;

    }

#endif
```

7.30 main/Logger.cpp File Reference

This file implements the Logger class, which provides a singleton logger for the Ragot engine.

```
#include "Logger.hpp"
```

Include dependency graph for Logger.cpp:



Namespaces

- namespace Ragot

Variables

- Logger & Ragot::logger = Logger::instance()

7.30.1 Detailed Description

This file implements the Logger class, which provides a singleton logger for the Ragot engine.

Author

Andrés Ragot (github.com/andresragot)

The Logger class allows logging messages with different severity levels (INFO, WARNING, ERROR). It supports formatted logging using printf-style format strings and can be used across different platforms. The logger can be configured to set the log level, and it uses the ESP-IDF logging system on ESP platforms.

Version

1.0

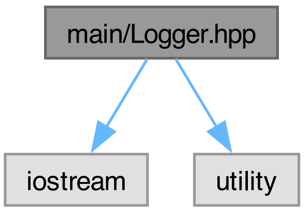
Date

2025-06-01

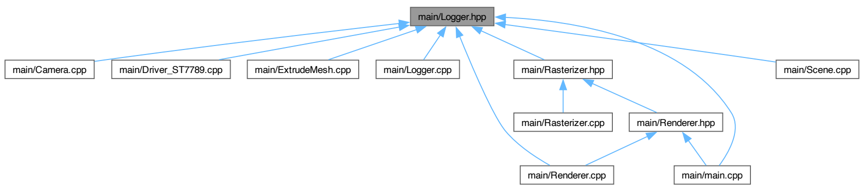
7.31 main/Logger.hpp File Reference

This file implements the Logger class, which provides a singleton logger for the Ragot engine.

```
#include <iostream>
#include <utility>
```
Include dependency graph for Logger.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Ragot::Logger

    Singleton logger class for the Ragot engine.

Namespaces

- namespace Ragot

7.31.1 Detailed Description

This file implements the Logger class, which provides a singleton logger for the Ragot engine.

Author

Andrés Ragot (github.com/andresragot)

The Logger class allows logging messages with different severity levels (INFO, WARNING, ERROR). It supports formatted logging using printf-style format strings and can be used across different platforms. The logger can be configured to set the log level, and it uses the ESP-IDF logging system on ESP platforms.

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

7.32 Logger.hpp

Go to the documentation of this file.

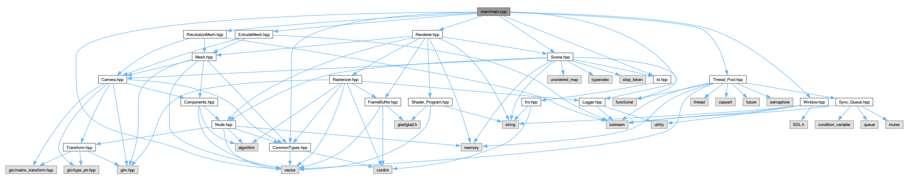
```


#pragma once

#include <iostream>
#include <utility>

#if ESP_PLATFORM == 1
#include "esp_log.h"
#endif

namespace Ragot
{
    class Logger
    {
    public:

        static Logger &instance()
        {
            static Logger instance;
            return instance;
        }

        template < typename... Args >
        void Log (const char * TAG, uint8_t level, const char * fmt, Args... args)
        {
            if (level > logLevel)
                return;

            #if ESP_PLATFORM == 1
            esp_log_write ((esp_log_level_t)level, TAG, fmt, std::forward<Args>(args)...);
            #else
            // 1) Calculamos el tamaño del buffer necesario
            int needed = std::snprintf(nullptr, 0, fmt, std::forward<Args>(args)...) + 1;
            std::vector<char> buffer(needed);

            // 2) Rellenamos el buffer con el texto formateado
            std::snprintf(buffer.data(), buffer.size(), fmt, std::forward<Args>(args)...);

            // 3) Lo imprimimos
            std::cout « "[" « TAG « "]: " « buffer.data() « std::endl;
            #endif
        }

    private:

        Logger () = default;

       ~Logger () = default;

        Logger (const Logger & ) = delete;

        Logger (const Logger &&) = delete;

        Logger & operator = (const Logger & ) = delete;

        Logger & operator = (const Logger &&) = delete;

    private:

        uint8_t logLevel = 0; // 0 = INFO, 1 = WARNING, 2 = ERROR

    public:

        void setLogLevel (uint8_t level)
        {
            logLevel = level;
            #if ESP_PLATFORM == 1
            esp_log_level_set("*", (esp_log_level_t)level);
            #endif
        }

        uint8_t getLogLevel () const
        {
            return logLevel;
        }

    };

    extern Logger & logger;
}
```

7.33 main/main.cpp File Reference

This file contains the main function for the Ragot engine, which initializes the renderer and scene, and starts the main rendering loop.

```
#include "Renderer.hpp"
#include <vector>
#include "CommonTypes.hpp"
#include "Scene.hpp"
#include "RevolutionMesh.hpp"
#include "ExtrudeMesh.hpp"
#include "Id.hpp"
#include "Logger.hpp"
#include "Thread_Pool.hpp"
#include "Window.hpp"
```

Include dependency graph for main.cpp:



Functions

- void main_loop (Renderer &renderer, Scene &scene, Window &window)

  Main loop for the Ragot engine on non-ESP platforms.
- int main (int argc, char ∗argv[ ])

  Main function for the Ragot engine.

Variables

- static const char ∗ MAIN_TAG = "Main"

7.33.1 Detailed Description

This file contains the main function for the Ragot engine, which initializes the renderer and scene, and starts the main rendering loop.

Author

Andrés Ragot (github.com/andresragot)

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

7.33.2 Function Documentation

7.33.2.1 main()

```
int main (
            int argc,
            char * argv[ ])
```

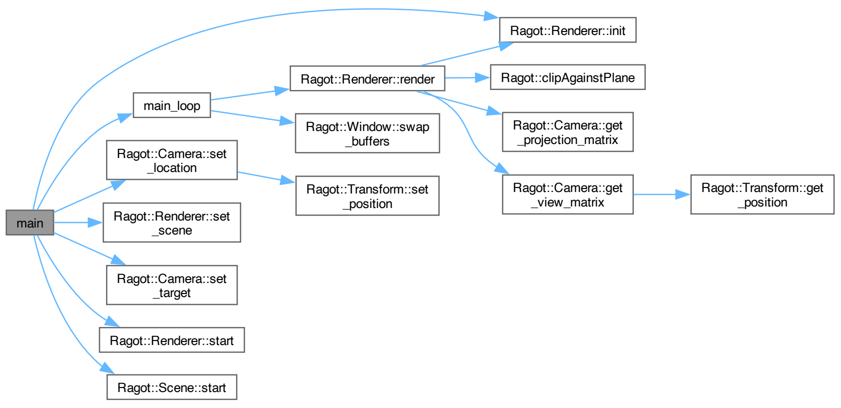
Main function for the Ragot engine.

This function initializes the logger, sets up the scene, creates the renderer, and enters the main rendering loop.

Returns

int Exit status of the program.

Here is the call graph for this function:


Ragot::Renderer::init
Ragot::Renderer::render
Ragot::clipAgainstPlane
main_loop
Ragot::Window::swap_buffers
Ragot::Camera::get_projection_matrix
Ragot::Camera::set_location
Ragot::Transform::set_position
Ragot::Camera::get_view_matrix
Ragot::Transform::get_position
main
Ragot::Renderer::set_scene
Ragot::Camera::set_target
Ragot::Renderer::start
Ragot::Scene::start

7.33.2.2 main_loop()

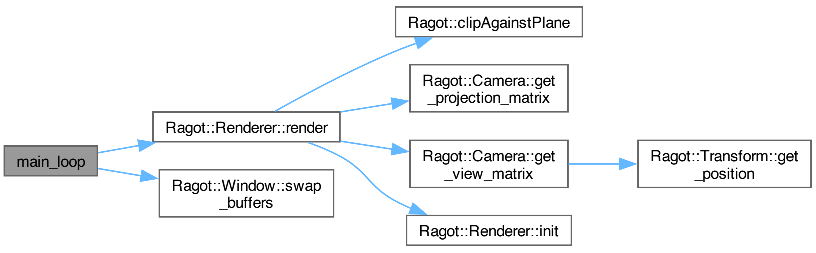
```
void main_loop (
             Renderer & renderer,
             Scene & scene,
             Window & window)
```

Main loop for the Ragot engine on non-ESP platforms.

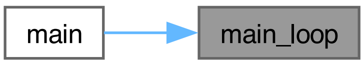
Parameters

renderer	Renderer instance to handle rendering.
scene	Scene instance containing the 3D objects.
window	Window instance for displaying the rendered output.

Here is the call graph for this function:


Ragot::clipAgainstPlane
Ragot::Camera::get_projection_matrix
Ragot::Renderer::render
main_loop
Ragot::Camera::get_view_matrix
Ragot::Transform::get_position
Ragot::Window::swap_buffers
Ragot::Renderer::init

Here is the caller graph for this function:



7.33.3 Variable Documentation

7.33.3.1 MAIN_TAG

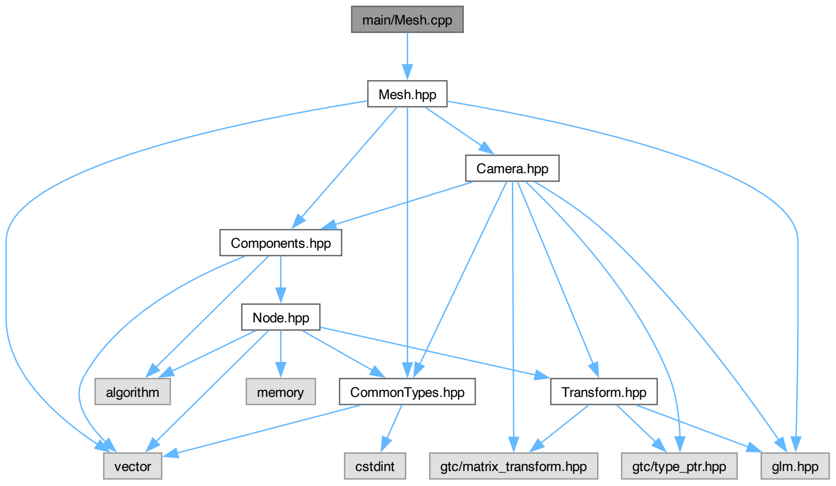
```
const char* MAIN_TAG = "Main"  [static]
```

7.34 main/Mesh.cpp File Reference

This file implements the Mesh class, which represents a 3D mesh in the Ragot engine.

```
#include "Mesh.hpp"
```

Include dependency graph for Mesh.cpp:



Namespaces

- namespace Ragot

7.34.1 Detailed Description

This file implements the Mesh class, which represents a 3D mesh in the Ragot engine.

Author

Andrés Ragot (github.com/andresragot)

The Mesh class is a base class for creating 3D meshes with vertices and faces. It provides methods to generate vertices and faces, apply transformations, and manage mesh information. The class also includes methods for setting and getting the color of the mesh.

Version

1.0

Date

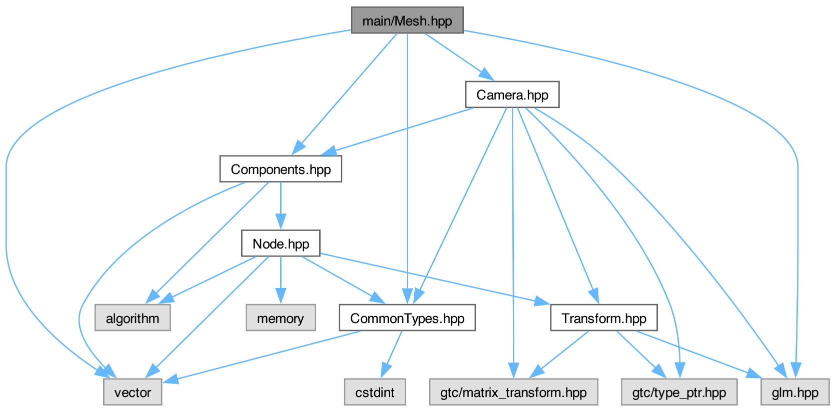
2025-06-01

7.35 main/Mesh.hpp File Reference

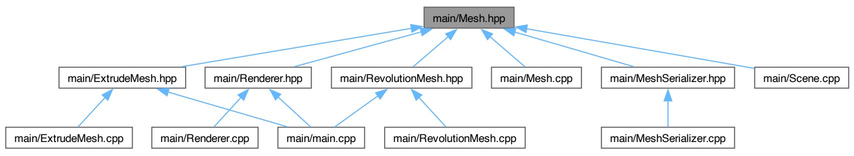
This file implements the Mesh class, which represents a 3D mesh in the Ragot engine.

```
#include "CommonTypes.hpp"
#include "Camera.hpp"
#include <vector>
#include "Components.hpp"
#include <glm.hpp>
```

Include dependency graph for Mesh.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Ragot::Mesh

  Represents a 3D mesh in the Ragot engine.

Namespaces

- namespace Ragot

7.35.1 Detailed Description

This file implements the Mesh class, which represents a 3D mesh in the Ragot engine.

Author

Andrés Ragot (github.com/andresragot)

The Mesh class is a base class for creating 3D meshes with vertices and faces. It provides methods to generate vertices and faces, apply transformations, and manage mesh information. The class also includes methods for setting and getting the color of the mesh.

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

7.36 Mesh.hpp

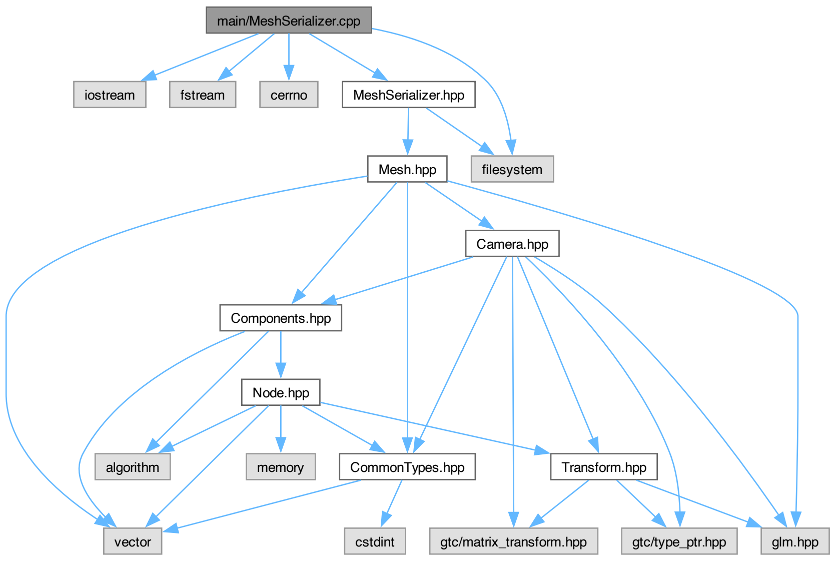
Go to the documentation of this file.

```
#pragma once

#include "CommonTypes.hpp"
#include "Camera.hpp"
#include <vector>
#include "Components.hpp"
#include <glm.hpp>

namespace Ragot
{
    class Mesh : public Component
    {
    protected:

        mesh_info_t mesh_info;
        uint16_t color = 0xFFFF;

        std::vector < glm::fvec4 > vertices;
        std::vector <   face_t > faces;

        int slices = 16;

    public:
        Mesh() = delete;

        virtual ~Mesh() = default;

        Mesh(mesh_info_t & mesh_info);

        virtual void generate_vertices() = 0;

        virtual void generate_faces() = 0;

        const std::vector < glm::fvec4 > & get_vertices() const { return vertices; }

        const std::vector <   face_t > & get_faces()    const { return faces;    }


        const size_t get_total_vertices() const { return vertices.size(); }

              size_t get_total_vertices()       { return vertices.size(); }

        void recalculate()
        {
            vertices.clear();
            faces.clear();

            generate_vertices();
            generate_faces();

            apply_transform_to_vertices();
        }

        void apply_transform_to_vertices()
        {
            glm::mat4 M = get_transform_matrix();
            for (auto & v : vertices)
            {
                v = M * v;
            }
        }

        void set_color(uint16_t new_color)
        {
            color = new_color;
        }

        uint16_t get_color() const
        {
            return color;
        }
    };
}
```

7.37 main/MeshSerializer.cpp File Reference

This file implements the MeshSerializer class, which provides methods to serialize a Mesh object to an OBJ file format.

```
#include <iostream>
#include <fstream>
#include <cerrno>
#include "MeshSerializer.hpp"
#include <filesystem>
```
Include dependency graph for MeshSerializer.cpp:



Namespaces

- namespace Ragot

Variables

- MeshSerializer & Ragot::serializer = MeshSerializer::instance()

7.37.1 Detailed Description

This file implements the MeshSerializer class, which provides methods to serialize a Mesh object to an OBJ file format.

Author

Andrés Ragot (github.com/andresragot)

The MeshSerializer class allows saving a Mesh object to an OBJ file, which is a common format for 3D models. It handles the serialization of vertices and faces, ensuring that the data is written in a format compatible with OBJ files. The class is designed to be used in graphics applications where exporting 3D models is required.

Version

1.0

Date

2025-06-01

Copyright

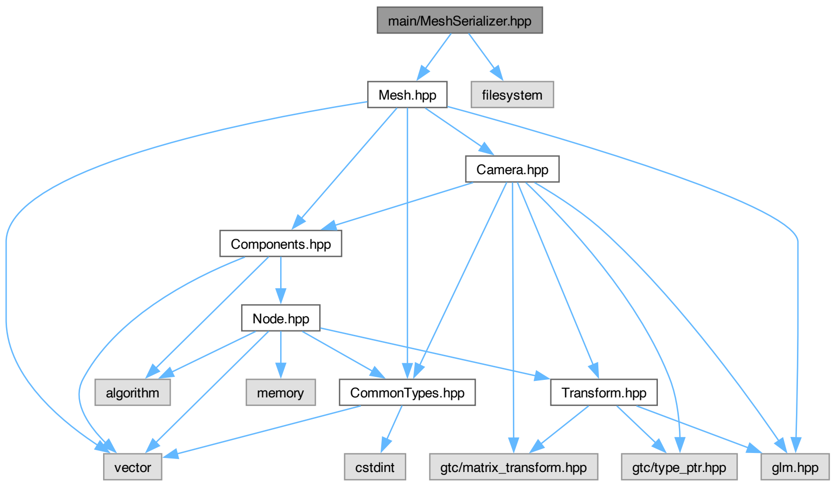
Copyright (c) 2025 MIT License

7.38 main/MeshSerializer.hpp File Reference

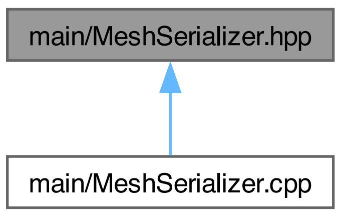
This file implements the MeshSerializer class, which provides methods to serialize a Mesh object to an OBJ file format.

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

Include dependency graph for MeshSerializer.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Ragot::MeshSerializer

  Singleton class to serialize Mesh objects to OBJ file format.

Namespaces

- namespace Ragot

7.38.1 Detailed Description

This file implements the MeshSerializer class, which provides methods to serialize a Mesh object to an OBJ file format.

Author

Andrés Ragot (github.com/andresragot)

The MeshSerializer class allows saving a Mesh object to an OBJ file, which is a common format for 3D models. It handles the serialization of vertices and faces, ensuring that the data is written in a format compatible with OBJ files. The class is designed to be used in graphics applications where exporting 3D models is required.

Version

1.0

Date

2025-06-01

7.39 MeshSerializer.hpp

Go to the documentation of this file.

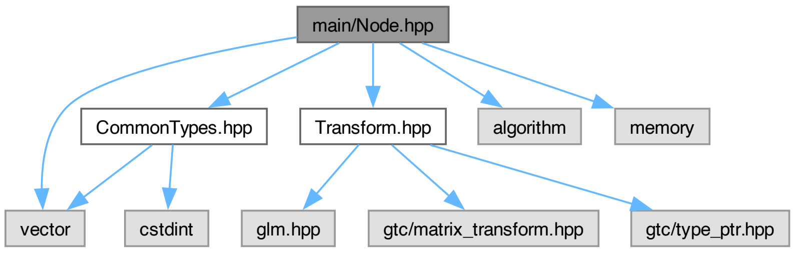
```

#pragma once

#include "Mesh.hpp"
#include <filesystem>

namespace Ragot
{
    // Vamos a hacer un Singleton.
    class MeshSerializer
    {
    public:
        static MeshSerializer & instance()
        {
            static MeshSerializer serializer;
            return serializer;
        }

    private:

        MeshSerializer () = default;

        MeshSerializer (const MeshSerializer &)  = delete;

        MeshSerializer (const MeshSerializer &&) = delete;

        MeshSerializer & operator = (const MeshSerializer &)  = delete;

        MeshSerializer & operator = (const MeshSerializer &&) = delete;

    public:

        bool save_to_obj (const Mesh & mesh, const std::filesystem::path & path);

    };

    extern MeshSerializer & serializer;
}
```

7.40 main/Node.hpp File Reference

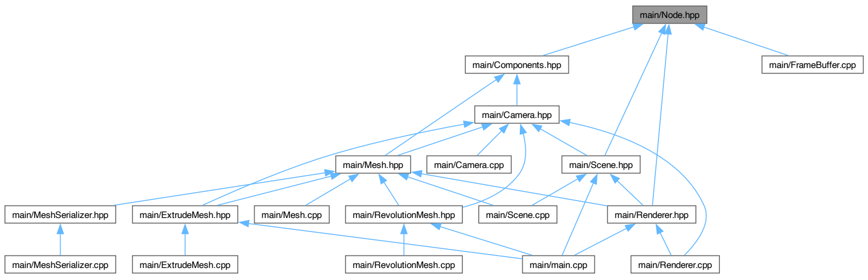
This file implements the Node class, which represents a node in a scene graph for 3D rendering.

```
#include "CommonTypes.hpp"
#include <vector>
#include "Transform.hpp"
#include <algorithm>
#include <memory>
```

Include dependency graph for Node.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Ragot::Node

  Represents a node in a scene graph for 3D rendering.

Namespaces

- namespace Ragot

7.40.1 Detailed Description

This file implements the Node class, which represents a node in a scene graph for 3D rendering.

Author

Andrés Ragot (github.com/andresragot)

The Node class is a part of the Ragot engine and extends the Transform class to include child nodes. It allows for hierarchical transformations and management of child nodes, enabling complex scene structures. The class provides methods to add and remove child nodes, retrieve the list of children, and compute the transformation matrix.

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

7.41 Node.hpp

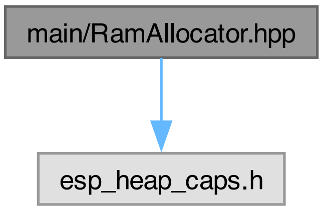
Go to the documentation of this file.

```
#pragma once
#include "CommonTypes.hpp"
#include <vector>
#include "Transform.hpp"
#include <algorithm>
#include <memory>

namespace Ragot
{
    using glm::mat4;

    class Node : public Transform
    {

    protected:

        std::vector < std::shared_ptr < Node > > children;
        Node * parent = nullptr;

    public:
        Node () = default;

        virtual ~Node () = default;

        Node (const Node &) = delete;

        Node (const Node &&) = delete;

        Node & operator = (const Node &) = delete;

        Node & operator = (const Node &&) = delete;
    public:

        void add_child (std::shared_ptr < Node > child)
        {
            if (child)
            {
                children.emplace_back(child);
                child->parent = this;
                child->dirty = true;
                dirty = true;
            }
        }

        void remove_child (std::shared_ptr < Node > child)
        {
            if (child)
            {
                auto it = std::remove(children.begin(), children.end(), child);
                if (it != children.end())
                {
                    children.erase(it, children.end());
                    child->parent = nullptr;
                    child->dirty = true;
                }
            }
        }

        const std::vector< std::shared_ptr < Node > >& get_children() const { return children; }

        mat4 get_transform_matrix() override
        {
            mat4 transform_matrix = Transform::get_transform_matrix();
            if (parent)
                transform_matrix = parent->get_transform_matrix() * transform_matrix;

            return transform_matrix;
        }

    };
}
```

7.42 main/RamAllocator.hpp File Reference

This file implements a custom memory allocator for PSRAM in the Ragot engine.

`#include "esp_heap_caps.h"`
Include dependency graph for RamAllocator.hpp:



Classes

- class Ragot::PSRAMAllocator< T, Flag >

  Custom memory allocator for PSRAM.
- struct Ragot::PSRAMAllocator< T, Flag >::rebind< U >

  Rebinds the allocator to a different type. This struct allows the PSRAMAllocator to be used with different types while maintaining the same allocation flags.

Namespaces

- namespace Ragot

Functions

- template<typename T, uint16_t F1, typename U, uint16_t F2>
  bool Ragot::operator== (const PSRAMAllocator< T, F1 > &, const PSRAMAllocator< U, F2 > &)

  Equality operator for PSRAMAllocator.
- template<typename T, uint16_t F1, typename U, uint16_t F2>
  bool Ragot::operator!= (const PSRAMAllocator< T, F1 > &a, const PSRAMAllocator< U, F2 > &b)

  Inequality operator for PSRAMAllocator.

7.42.1 Detailed Description

This file implements a custom memory allocator for PSRAM in the Ragot engine.

Author

Andrés Ragot (github.com/andresragot)

The PSRAMAllocator class provides a way to allocate and deallocate memory in PSRAM with specific flags. It is designed to be used with standard containers like std::vector, allowing for efficient memory management in embedded systems. The allocator uses ESP-IDF's heap_caps_malloc and heap_caps_free functions to manage memory.

Version

1.0

Date

2025-06-01

Copyright

7.43 RamAllocator.hpp

Go to the documentation of this file.

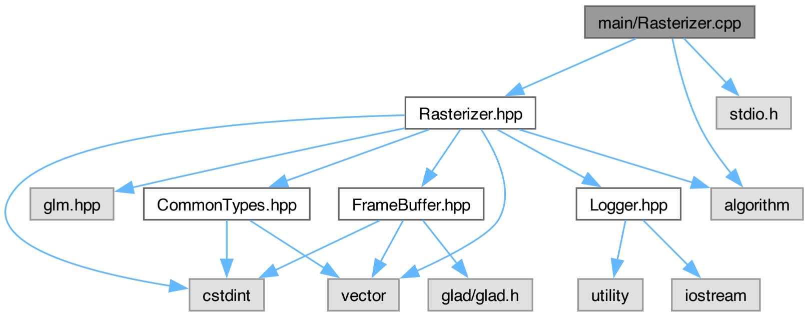
```

#pragma once
#include "esp_heap_caps.h"

namespace Ragot
{

    template <typename T, uint16_t Flag>
    class PSRAMAllocator
    {
    public:
        using value_type    = T;
        using pointer       = T*;
        using size_type     = std::size_t;

        template <typename U>
        struct rebind { using other = PSRAMAllocator<U, Flag>; };

        PSRAMAllocator() noexcept {}

        template <typename U, uint16_t F2>
        PSRAMAllocator(const PSRAMAllocator<U, F2>&) noexcept {}

        T* allocate(size_type n)
        {
            T* p = static_cast<T*>(heap_caps_malloc(n * sizeof(T), Flag));
            if (not p) throw std::bad_alloc();
            return p;
        }

        void deallocate(T* p, size_type) noexcept
        {
            heap_caps_free(p);
        }
    };

    template <typename T, uint16_t F1, typename U, uint16_t F2>
    bool operator==(const PSRAMAllocator<T, F1>&, const PSRAMAllocator<U, F2>&)
    {
      return F1 == F2;
    }

    template <typename T, uint16_t F1, typename U, uint16_t F2>
    bool operator!=(const PSRAMAllocator<T, F1>& a, const PSRAMAllocator<U, F2>& b)
    {
      return !(a == b);
    }

} // namespace Ragot
```

7.44 main/Rasterizer.cpp File Reference

Implementation of the Rasterizer class for rendering polygons in a frame buffer.

```
#include "Rasterizer.hpp"
#include <stdio.h>
#include <algorithm>
```

Include dependency graph for Rasterizer.cpp:



Namespaces

- namespace Ragot

Variables

- template<class COLOR_BUFFER_TYPE>
  int Ragot::Rasterizer< COLOR_BUFFER_TYPE >::offset_cache0 [1024]
- template<class COLOR_BUFFER_TYPE>
  int Ragot::Rasterizer< COLOR_BUFFER_TYPE >::offset_cache1 [1024]
- template<class COLOR_BUFFER_TYPE>
  int Ragot::Rasterizer< COLOR_BUFFER_TYPE >::z_cache0 [1024]
- template<class COLOR_BUFFER_TYPE>
  int Ragot::Rasterizer< COLOR_BUFFER_TYPE >::z_cache1 [1024]

7.44.1 Detailed Description

Implementation of the Rasterizer class for rendering polygons in a frame buffer.

Author

Andrés Ragot (github.com/andresragot)

Version

1.0

Date

2025-06-01

Copyright

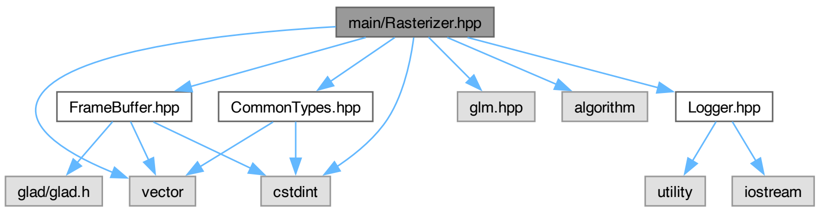
7.45 main/Rasterizer.hpp File Reference

Implementation of the Rasterizer class for rendering polygons in a frame buffer.

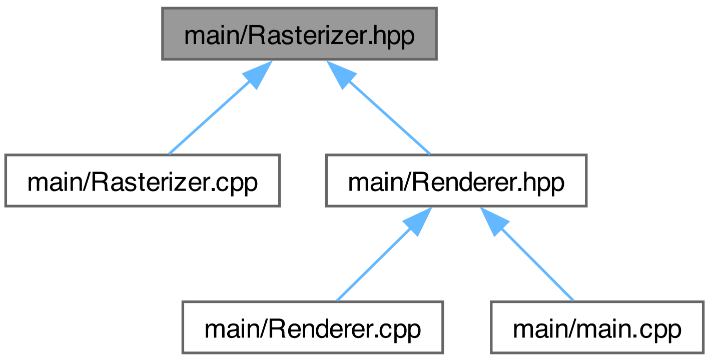
```
#include "FrameBuffer.hpp"
#include "CommonTypes.hpp"
#include <glm.hpp>
#include <cstdint>
#include <vector>
#include <algorithm>
```

```
#include "Logger.hpp"
```

Include dependency graph for Rasterizer.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Ragot::Rasterizer< Color >

  Class for rasterizing polygons in a frame buffer.

Namespaces

- namespace Ragot

7.45.1 Detailed Description

Implementation of the Rasterizer class for rendering polygons in a frame buffer.

Author

Andrés Ragot (github.com/andresragot)

Version

1.0

Date

2025-06-01

Copyright

7.46 Rasterizer.hpp

Go to the documentation of this file.

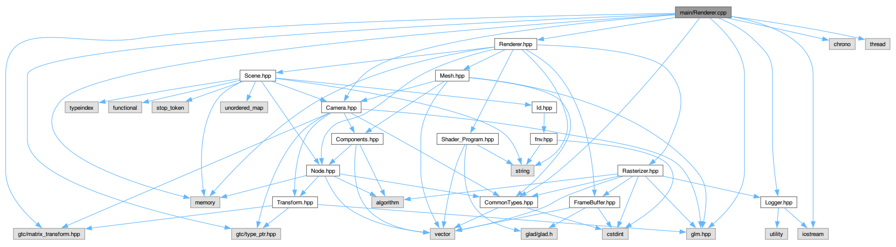
```


#pragma once
#include "FrameBuffer.hpp"
#include "CommonTypes.hpp"
#include <glm.hpp>
#include <cstdint>
#include <vector>
#include <algorithm>
#include "Logger.hpp"

namespace Ragot
{
    template <typename Color>
    class Rasterizer
    {

    private:
        FrameBuffer < Color > & frame_buffer;

        static int offset_cache0 [1024];
        static int offset_cache1 [1024];
#ifndef CONFIG_GRAPHICS_PAINTER_ALGO_ENABLED

        static int z_cache0 [1024];
        static int z_cache1 [1024];

        std::vector < int > z_buffer;
#endif // CONFIG_GRAPHICS_PAINTER_ALGO_ENABLED


        Color color;
        Color clear_color;

        static constexpr const char * RASTER_TAG = "Rasterizer";

    public:

        Rasterizer (FrameBuffer < Color > & frame) : frame_buffer (frame)
#ifndef CONFIG_GRAPHICS_PAINTER_ALGO_ENABLED
        , z_buffer(frame.get_width () * frame.get_height ())
#endif // CONFIG_GRAPHICS_PAINTER_ALGO_ENABLED
        {
        }

        Rasterizer () = default;

       ~Rasterizer () = default;

    public:
        const FrameBuffer < Color > & get_frame_buffer() const
        {
            return (frame_buffer);
        }

        void set_color (const Color & new_color)
        {
            color = new_color;
            // frame_buffer.set_color (new_color);
        }

        void clear ()
        {
            logger.Log (RASTER_TAG, 3, "Limpiando framebuffer");
            frame_buffer.clear_buffer();

#ifndef CONFIG_GRAPHICS_PAINTER_ALGO_ENABLED
            for (int * z = z_buffer.data(), * end = z + z_buffer.size(); z != end; ++z)
            {
                * z = std::numeric_limits< int >::max ();
            }
#endif // CONFIG_GRAPHICS_PAINTER_ALGO_ENABLED
        }

        void fill_convex_polygon (  const glm::ivec4 * const vertices,
                                    const int        * const indices_begin,
                                    const int        * const indices_end
                                 );

        void fill_convex_polygon ( const glm::ivec4 * const vertices,
                                      const face_t     * const face
                                 );

#ifndef CONFIG_GRAPHICS_PAINTER_ALGO_ENABLED

        void fill_convex_polygon_z_buffer ( const glm::ivec4 * const vertices,
                                            const face_t     * const face
                                          );

        void fill_convex_polygon_z_buffer (
                                            const glm::ivec4 * const vertices,
                                            const int     * const indices_begin,
                                            const int     * const indices_end
                                          );
#endif // CONFIG_GRAPHICS_PAINTER_ALGO_ENABLED

        // Logs debug para rasterizado
        bool debug_enabled = true;

    private:

        template < typename VALUE_TYPE, size_t SHIFT >
        void interpolate (int * cache, int v0, int v1, int y_min, int y_max);

        template < unsigned COLOR_SIZE >
        void fill_row (Color * start, unsigned left_offset, unsigned right_offset, const Color & color)
        {
            std::fill_n (start + left_offset, right_offset - left_offset, color);
        }

        // dentro de Rasterizer<COLOR> o en un header común
        template <unsigned COLOR_SIZE>
        void fill_row_zbuffer (
            Color *       start,        // puntero al primer píxel de la scanline
            int   *       zbuffer,      // puntero al primer elemento del Z-buffer
            unsigned      left_offset,  // offset inicial (inclusive)
            unsigned      right_offset, // offset final   (exclusive)
            int           z_start,      // profundidad en left_offset
            int           dz,           // incremento de z por píxel
            const Color & color         // color a pintar
                                    )
        {
            unsigned length = right_offset - left_offset;
            Color * pix = start   + left_offset;
            int   * zb  = zbuffer + left_offset;

            // Recorremos de 0 a length-1
            for (unsigned i = 0; i < length; ++i, ++pix, ++zb, z_start += dz)
            {
                if (z_start < *zb)
                {
                    *pix = color;
                    *zb  = z_start;
                }
            }
        }
    };

    template class Rasterizer<RGB565>;
}
```

7.47 main/Renderer.cpp File Reference

Implementation of the Renderer class for rendering scenes in the Ragot engine.

```
#include "Renderer.hpp"
#include "Camera.hpp"
#include <iostream>
#include <glm.hpp>
#include <gtc/matrix_transform.hpp>
#include <gtc/type_ptr.hpp>
#include "CommonTypes.hpp"
#include "Logger.hpp"
#include <memory>
#include <chrono>
#include <thread>
```

Include dependency graph for Renderer.cpp:



Namespaces

- namespace Ragot

Typedefs

- using Ragot::Matrix4x4 = glm::mat4

Functions

- template<typename Inside, typename Intersect>
  static std::vector< glm::fvec4 > Ragot::clipAgainstPlane (const std::vector< glm::fvec4 > &in, Inside inside, Intersect intersect)

Variables

- static const char ∗ Ragot::RENDERER_TAG = "Renderer"
- static const char ∗ Ragot::MAIN_TAG = "Main"

7.47.1 Detailed Description

Implementation of the Renderer class for rendering scenes in the Ragot engine.

Author

Andrés Ragot (github.com/andresragot)

The Renderer class is responsible for rendering 3D scenes using a rasterization approach.

Version

1.0

Date

2025-06-01

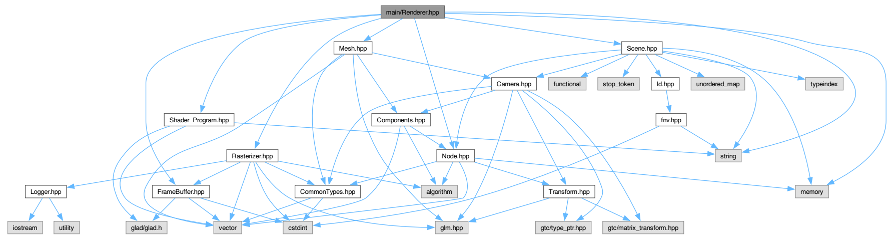
Copyright

Copyright (c) 2025 MIT License

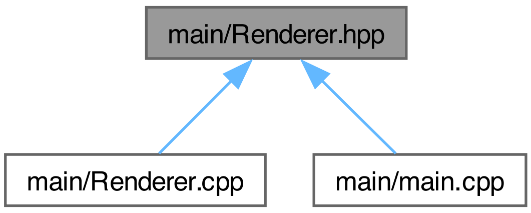
7.48 main/Renderer.hpp File Reference

Implementation of the Renderer class for rendering scenes in the Ragot engine.

```
#include "Mesh.hpp"
#include "FrameBuffer.hpp"
#include "Rasterizer.hpp"
#include "Node.hpp"
#include "Scene.hpp"
#include <memory>
#include <string>
#include "Shader_Program.hpp"
```
Include dependency graph for Renderer.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Ragot::Renderer

  Class for rendering scenes in the Ragot engine.

Namespaces

- namespace Ragot

7.48.1 Detailed Description

Implementation of the Renderer class for rendering scenes in the Ragot engine.

Author

Andrés Ragot (github.com/andresragot)

The Renderer class is responsible for rendering 3D scenes using a rasterization approach.

Version

1.0

Date

2025-06-01

Copyright

7.49 Renderer.hpp

Go to the documentation of this file.

```


#pragma once

#include "Mesh.hpp"
#include "FrameBuffer.hpp"
#include "Rasterizer.hpp"
#include "Node.hpp"
#include "Scene.hpp"
#include <memory>
#include <string>

#if ESP_PLATFORM == 1
#ifdef CONFIG_IDF_TARGET_ESP32P4
#include "driver_ek79007.hpp"
```

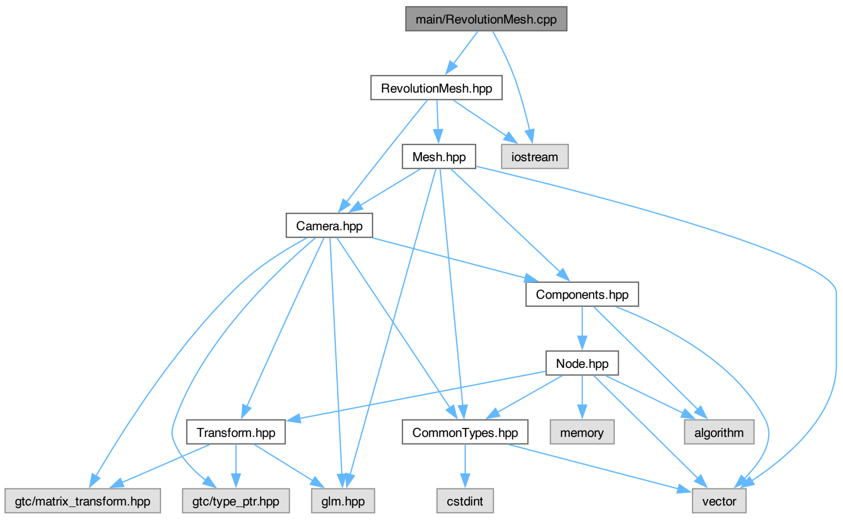
```
#elif CONFIG_IDF_TARGET_ESP32S3
#include "Driver_ST7789.hpp"
#endif
#else
#include "Shader_Program.hpp"
#endif

namespace Ragot
{
    class Renderer
    {
    private:

        float accumulated_time = 0.f;
        size_t iterations = 0;
        static constexpr size_t number_of_iterations = 10000000000000000;

        #if ESP_PLATFORM == 1
        #ifdef CONFIG_IDF_TARGET_ESP32P4
        DriverEK79007 driver;
        #elif CONFIG_IDF_TARGET_ESP32S3
        Driver_ST7789 driver;
        #endif
        #else
        std::unique_ptr < Shader_Program > quadShader       = nullptr;
        GLuint           quadVAO         = 0;
        GLuint           quadVBO         = 0;
        GLuint           quadEBO         = 0;

        void initFullScreenQuad();

        static const std::string   vertex_shader_code;
        static const std::string fragment_shader_code;
        #endif
        FrameBuffer < RGB565 > frame_buffer;
        Scene * current_scene = nullptr;
        Rasterizer < RGB565 > rasterizer;

        unsigned  width;
        unsigned height;

        bool initialized = false;

        std::atomic<bool> running = false;

    public:
        Renderer () = delete;

        Renderer (unsigned width, unsigned height);

       ~Renderer () = default;

        std::vector < glm::fvec4 > transformed_vertices;
        std::vector < glm::ivec4 > display_vertices;

        void set_scene (Scene * scene) { current_scene = scene; }

        void init ();

        void render ();

        void task_render (std::stop_token stop_token);

        bool is_frontface (const glm::fvec4 * const projected_vertices, const face_t * const indices);

        void start() { running = true;  }

        void  stop() { running = false; }
    };
}


```

7.50 main/RevolutionMesh.cpp File Reference

Implementation of the RevolutionMesh class for generating revolution meshes.

```
#include "RevolutionMesh.hpp"
#include <iostream>
```

Include dependency graph for RevolutionMesh.cpp:



Namespaces

- namespace Ragot

7.50.1 Detailed Description

Implementation of the RevolutionMesh class for generating revolution meshes.

Author

Andrés Ragot (github.com/andresragot)

The RevolutionMesh class generates a mesh by revolving a 2D profile around an axis.

Version

1.0

Date

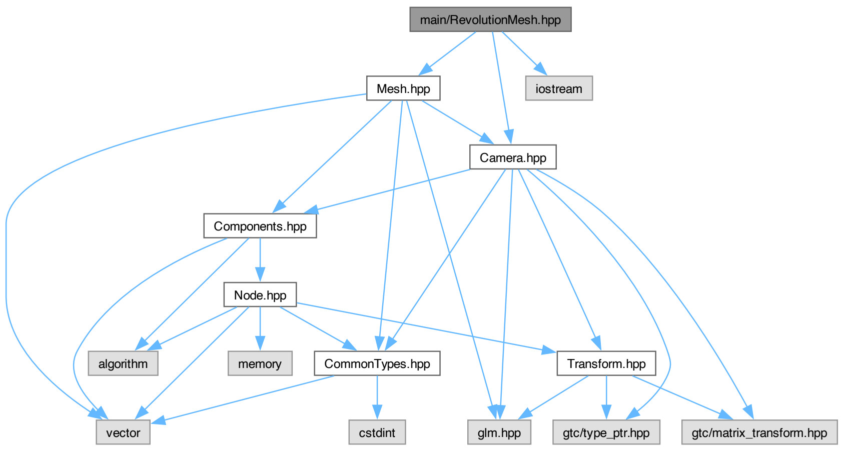
2025-06-01

7.51 main/RevolutionMesh.hpp File Reference

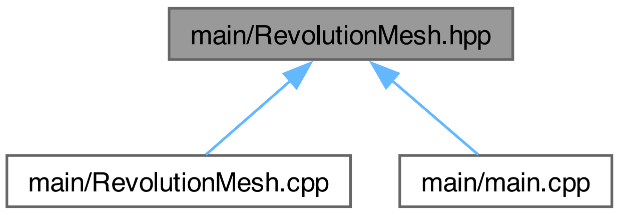
Implementation of the RevolutionMesh class for generating revolution meshes.

```
#include "Mesh.hpp"
#include "Camera.hpp"
#include <iostream>
```

Include dependency graph for RevolutionMesh.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Ragot::RevolutionMesh

  Class for generating revolution meshes.

Namespaces

- namespace Ragot

7.51.1 Detailed Description

Implementation of the RevolutionMesh class for generating revolution meshes.

Author

Andrés Ragot (github.com/andresragot)

The RevolutionMesh class generates a mesh by revolving a 2D profile around an axis.

Version

1.0

Date

2025-06-01

Copyright

7.52 RevolutionMesh.hpp

Go to the documentation of this file.

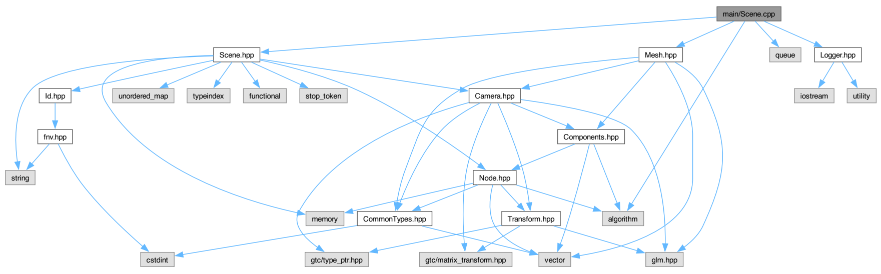
```

#pragma once

#include "Mesh.hpp"
#include "Camera.hpp"
#include <iostream>

namespace Ragot
{
    class RevolutionMesh : public Mesh
    {
    protected:
        const Camera & cam;
        bool faces_can_be_quads;
        static constexpr float PI = 3.14159265358979323846f;
    public:
        RevolutionMesh (mesh_info_t & mesh_info, const Camera & cam) : Mesh (mesh_info), cam (cam)
        {
            faces_can_be_quads = (mesh_info.vertex_amount % 8 == 0 || mesh_info.vertex_amount == 4);
            vertices.reserve (mesh_info.coordinates.size() * (slices + 1));
               faces.reserve (mesh_info.coordinates.size() *  slices);

            generate_vertices();
            generate_faces();

            std::cout « "Revolution Vertices: " « vertices.size() « std::endl;
            std::cout « "Revolution Faces: " « faces.size() « std::endl;
        }

       ~RevolutionMesh() = default;

        void generate_vertices () override;

        void generate_faces    () override;
    };
}
```

7.53 main/Scene.cpp File Reference

Implementation of the Scene class for managing 3D scenes.

```
#include "Scene.hpp"
#include "Mesh.hpp"
#include <queue>
#include <algorithm>
#include "Logger.hpp"
```

Include dependency graph for Scene.cpp:



Namespaces

- namespace Ragot

7.53.1 Detailed Description

Implementation of the Scene class for managing 3D scenes.

Author

Andrés Ragot (github.com/andresragot)

The Scene class provides methods to manage nodes, cameras, and scene traversal.

Version

1.0

Date

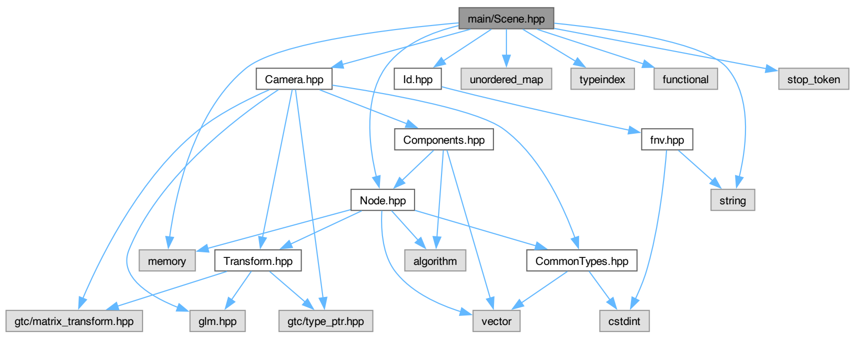
2025-06-01

7.54 main/Scene.hpp File Reference

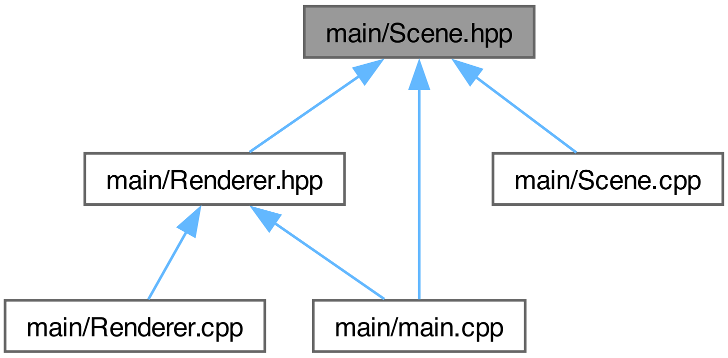
Implementation of the Scene class for managing 3D scenes.

```
#include "Node.hpp"
#include "Camera.hpp"
#include <string>
#include <unordered_map>
#include <typeindex>
#include <functional>
#include <memory>
#include "Id.hpp"
#include <stop_token>
```

Include dependency graph for Scene.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Ragot::Scene

  Class for managing a 3D scene.

Namespaces

- namespace Ragot

7.54.1 Detailed Description

Implementation of the Scene class for managing 3D scenes.

Author

Andrés Ragot (github.com/andresragot)

The Scene class provides methods to manage nodes, cameras, and scene traversal.

Version

1.0

Date

2025-06-01

7.55 Scene.hpp

Go to the documentation of this file.

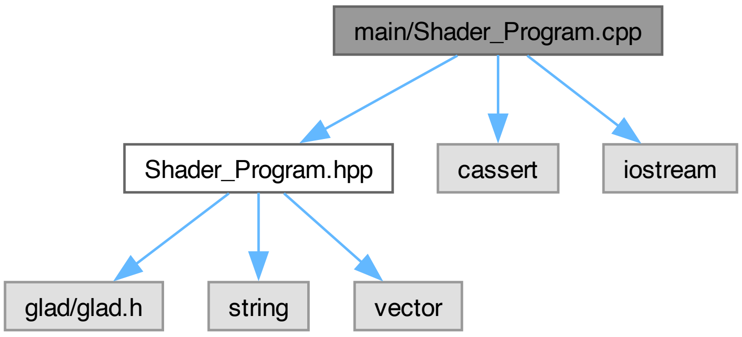
```


#pragma once

#include "Node.hpp"
#include "Camera.hpp"
#include <string>
#include <unordered_map>
#include <typeindex>
#include <functional>
#include <memory>
#include "Id.hpp"
#include <stop_token>

namespace Ragot
{
    class Scene
    {
    private:
        Camera * main_camera = nullptr;
        std::shared_ptr < Node > root_node;
        std::unordered_map<basics::Id, std::shared_ptr < Node > > named_nodes;
        std::atomic<bool> running = false;

    public:
        Scene();

       ~Scene() = default;

        Scene (Camera * camera);

        void add_node(std::shared_ptr < Node > node, const basics::Id name);

        void remove_node(std::shared_ptr < Node > node);

        std::shared_ptr < Node > find_node(const basics::Id name);

        void set_main_camera(Camera * camera);

        Camera * get_main_camera() const { return main_camera; }

        void traverse(const std::function<void(std::shared_ptr < Node >) >& callback);

        template<typename T>
        std::vector<std::shared_ptr < T > > collect_components();

        void update(float delta_time);
```

```

              std::shared_ptr < Node > get_root()       { return root_node; }
        const std::shared_ptr < Node > get_root() const { return root_node; }

        void start() { running = true; }

        void stop() { running = false; }

    private:
        void task_update (std::stop_token, float delta_time);
    };
}
```

7.56 main/Shader_Program.cpp File Reference

```
#include "Shader_Program.hpp"
#include <cassert>
#include <iostream>
```

Include dependency graph for Shader_Program.cpp:



Namespaces

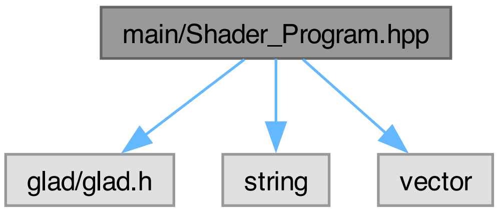
- namespace Ragot

7.57 main/Shader_Program.hpp File Reference

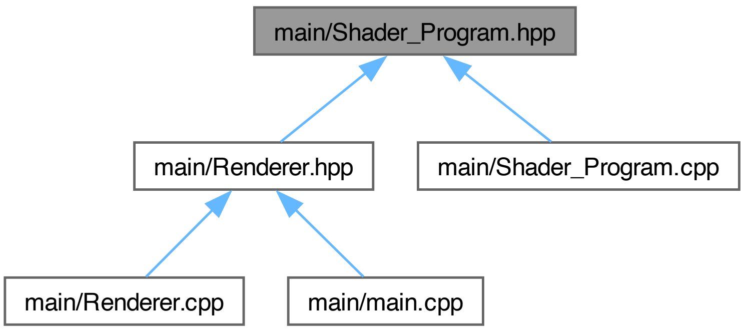
```
#include <glad/glad.h>
#include <string>
```

```
#include <vector>
```

Include dependency graph for Shader_Program.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Ragot::Shader

  Class for managing an OpenGL shader.
- class Ragot::Vertex_Shader

  Class for managing an OpenGL vertex shader.
- class Ragot::Fragment_Shader

  Class for managing an OpenGL fragment shader.
- class Ragot::Shader_Program

  Class for managing an OpenGL shader program.

Namespaces

- namespace Ragot

7.58 Shader_Program.hpp

Go to the documentation of this file.

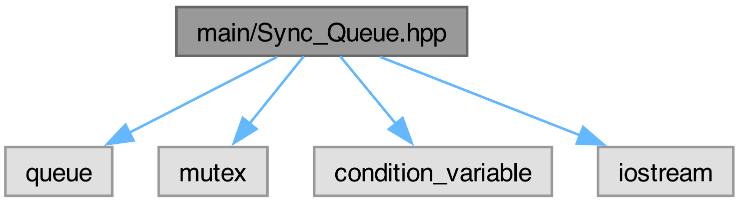
```
/*
 *  This file is part of OpenGL-FinalProject
 *
 *  Developed by Andrés Ragot - github.com/andresragot
 *
 *  MIT License
 *
 *  Copyright (c) 2025 Andrés Ragot
 *
 *  Permission is hereby granted, free of charge, to any person obtaining a copy
 *  of this software and associated documentation files (the "Software"), to deal
 *  in the Software without restriction, including without limitation the rights
 *  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 *  copies of the Software, and to permit persons to whom the Software is
 *  furnished to do so, subject to the following conditions:
 *
 *  The above copyright notice and this permission notice shall be included in all
 *  copies or substantial portions of the Software.
 *
 *  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 *  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 *  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 *  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 *  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 *  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
 *  SOFTWARE.a
 */

#pragma once

#include <glad/glad.h>

#include <string>
#include <vector>

namespace Ragot
{
    using namespace std;

    class Shader
    {
    private:
        GLuint id;
        string error;
        bool compilation_succeeded;

    protected:
        Shader(const vector<string>& source_code, GLenum type);

        GLuint compile_shader();

        void show_compilation_error();

    public:
        Shader() = delete;

        ~Shader()
        {
            glDeleteShader(id);
        }

        GLuint get_id() const
        {
            return id;
        }

        string* get_error()
        {
            return error.empty() ? nullptr : &error;
        }

        bool is_ok() const
        {
            return compilation_succeeded;
        }
    };

    class Vertex_Shader : public Shader
    {
    public:
        Vertex_Shader(const vector<string>& source_code) : Shader(source_code, GL_VERTEX_SHADER)
        {
        }
    };

    class Fragment_Shader : public Shader
    {
    public:
        Fragment_Shader(const vector<string>& source_code) : Shader(source_code, GL_FRAGMENT_SHADER)
        {
        }
    };

    class Shader_Program
    {
    private:
        GLuint program_id;

    public:
        Shader_Program(const vector<string>& source_code_vertex, const vector<string>&
source_code_fragment);

        Shader_Program() = delete;

        ~Shader_Program()
        {
            glDeleteProgram(program_id);
        }

        void use() const
        {
            glUseProgram(program_id);
        }

        GLuint get_id() const
        {
            return program_id;
        }

        GLuint get_uniform_location(string uniform_name) const
        {
            return glGetUniformLocation(program_id, uniform_name.c_str());
        }

    private:
        Shader_Program(const Shader_Program&) = delete;
        Shader_Program& operator=(const Shader_Program&) = delete;

        void initialize(GLuint vertex_shader_id, GLuint fragment_shader_id);

        void show_linkage_error();
    };
}
```

7.59 main/Sync_Queue.hpp File Reference

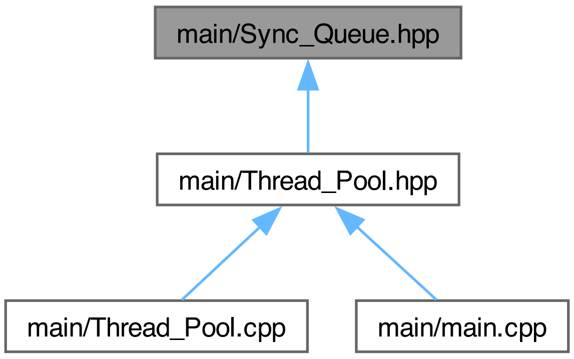
Implementation of a synchronized queue for thread-safe operations.

```
#include <queue>
#include <mutex>
#include <condition_variable>
#include <iostream>
```

Include dependency graph for Sync_Queue.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Ragot::Sync_Queue< T >

    A thread-safe queue implementation.

Namespaces

- namespace Ragot

7.59.1 Detailed Description

Implementation of a synchronized queue for thread-safe operations.

Author

Andrés Ragot (github.com/andresragot)

The Sync_Queue class provides a thread-safe queue implementation using mutexes and condition variables.

Version

1.0

Date

2025-06-02

Copyright

Copyright (c) 2025 MIT License

7.60 Sync_Queue.hpp

Go to the documentation of this file.

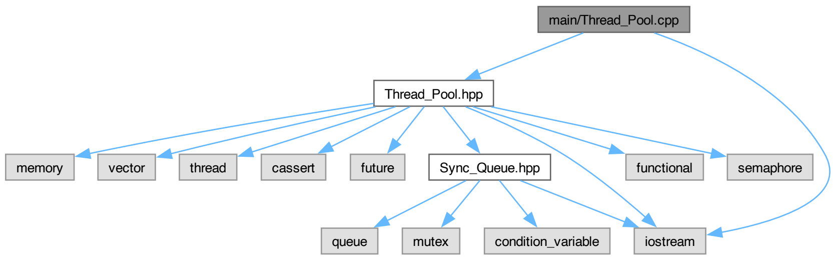
```

#pragma once

#include <queue>
#include <mutex>
#include <condition_variable>
#include <iostream>

namespace Ragot
{
    template<typename T>
    class Sync_Queue
    {
    public:
        using value_type = T;

    public:
        Sync_Queue () = default;

       ~Sync_Queue ()
        {
            close ();
        }

        Sync_Queue (const Sync_Queue &) = delete;

        Sync_Queue & operator=(const Sync_Queue &) = delete;

    private:
        std::queue<T> queue;
        mutable std::mutex mutex;
        std::condition_variable condition;
        bool closed = false;

    public:
        std::optional < value_type > get()
        {
            std::unique_lock<std::mutex> lock(mutex);
            condition.wait(lock, [this] { return closed || !queue.empty(); });
            if (not closed)
            {
                value_type value = queue.front();
                queue.pop();
                return value;
            }
            return std::nullopt;
        }

        void push (const value_type & value)
        {
            std::lock_guard<std::mutex> lock(mutex);
            if (not closed)
            {
                queue.push(value);
                condition.notify_one ();
            }
        }

        template < typename ...ARGUMENTS >
        void push (ARGUMENTS && ...arguments)
        {
            std::lock_guard<std::mutex> lock(mutex);
            if (not closed)
            {
                queue.push (std::forward < ARGUMENTS > (arguments)...);
                condition.notify_one ();
            }
        }

        template < typename ...ARGUMENTS >
        void emplace (ARGUMENTS && ...arguments)
        {
            std::lock_guard<std::mutex> lock(mutex);
            if (not closed)
            {
                queue.emplace(std::forward<ARGUMENTS>(arguments)...);
                condition.notify_one ();
            }
        }

        value_type & back ()
        {
            std::lock_guard<std::mutex> lock(mutex);
            return queue.back();
        }

        void close ()
        {
            {
                std::lock_guard<std::mutex> lock(mutex);
                closed = true;
            }
            condition.notify_all();
        }

        void clear ()
        {
            std::queue < value_type >  empty;
            std::lock_guard < std::mutex > lock(mutex);
            queue.swap(empty);
        }

        void swap (Sync_Queue & other)
        {
            std::lock(mutex, other.mutex);
            std::lock_guard<std::mutex> lock1(this->mutex, std::adopt_lock);
            std::lock_guard<std::mutex> lock2(other.mutex, std::adopt_lock);
            queue.swap(other.queue);
        }
        bool empty () const
        {
            std::lock_guard<std::mutex> lock(mutex);
            return queue.empty ();
        }

        size_t size () const
        {
            std::lock_guard<std::mutex> lock(mutex);
            return queue.size ();
        }
    };
}
```

7.61 main/Thread_Pool.cpp File Reference

Implementation of the Thread_Pool class for managing a pool of threads.

```
#include "Thread_Pool.hpp"
#include <iostream>
```

Include dependency graph for Thread_Pool.cpp:



Namespaces

- namespace Ragot

Variables

- Thread_Pool & Ragot::thread_pool = Thread_Pool::instance ()

7.61.1 Detailed Description

Implementation of the Thread_Pool class for managing a pool of threads.

Author

Andrés Ragot (github.com/andresragot)

The Thread_Pool class provides a way to manage a pool of threads that can execute tasks concurrently.

Version

1.0

Date

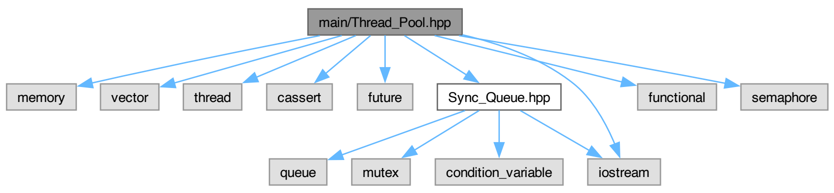
2025-06-02

Copyright

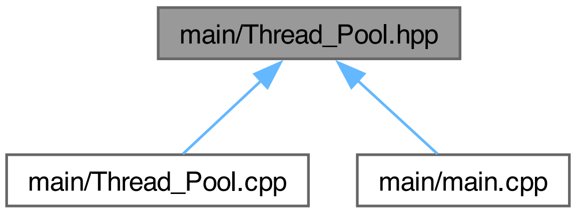
7.62 main/Thread_Pool.hpp File Reference

```
#include <memory>
#include <vector>
#include <thread>
#include <cassert>
#include <future>
#include "Sync_Queue.hpp"
#include <iostream>
#include <functional>
#include <semaphore>
```

Include dependency graph for Thread_Pool.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Ragot::Thread_Pool

  A thread pool for managing concurrent tasks.

Namespaces

- namespace Ragot

7.63 Thread_Pool.hpp

Go to the documentation of this file.

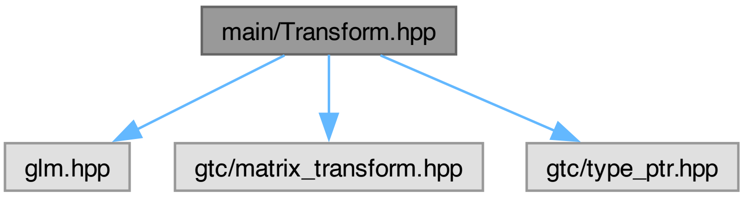
```
#pragma once

#include <memory>
#include <vector>
#include <thread>
#include <cassert>
#include <future>
#include "Sync_Queue.hpp"
#include <iostream>
#include <functional>
#include <semaphore>

namespace Ragot
{
    class Thread_Pool
    {
    public:
        using Task = std::function <  void (std::stop_token) >;
        std::binary_semaphore sem_mesh_ready {0};
        std::binary_semaphore sem_render_done {0};

    private:
        Sync_Queue < Task > tasks;

        std::vector < std::unique_ptr < std::jthread > > threads;

        std::atomic < bool > started;

    public:
        static Thread_Pool & instance ()
        {
            static Thread_Pool instance;
            return instance;
        }

    private:
        Thread_Pool()
        {
            auto cores = std::thread::hardware_concurrency();

            std::cout « "Thread pool initialized with " « cores « " threads." « std::endl;

            if (cores == 0)
            {
                cores = 2; // Default to 2 threads if hardware concurrency is not available
            }

            threads.resize (cores);

            started = false;
        }

        ~Thread_Pool()
        {
            sem_mesh_ready.release();
            sem_render_done.release();

            if (started)
            {
                stop();
            }

        }

        Thread_Pool (Thread_Pool & ) = delete;

        Thread_Pool (Thread_Pool &&) = delete;

        Thread_Pool & operator = (Thread_Pool & ) = delete;

        Thread_Pool & operator = (Thread_Pool &&) = delete;

    public:

        void start ()
        {
            assert (not started);

            std::cout « "Starting thread pool..." « std::endl;

            for (auto & thread : threads)
            {
                thread = std::make_unique < std::jthread > ( std::bind (&Thread_Pool::thread_function,
      this, std::placeholders::_1) );
            }

            started = true;
        }

        void stop ()
        {
            assert (started == true);
            started = false;

            tasks.close ();
            for (auto & thread : threads)
            {
                thread->request_stop();
            }


            threads.clear();

        }

        template<typename F, typename... Args>
        std::future < std::invoke_result_t < F, Args... > > submit (F && f, Args && ... args)
        {
            std::cout « "submit" « std::endl;

            using ReturnType = std::invoke_result_t < F, Args... >;

            auto task_ptr = std::make_shared < std::packaged_task < ReturnType() > > (
                std::bind (std::forward < F > (f), std::forward < Args > (args)...)
            );

            std::future<ReturnType> res = task_ptr->get_future();

            tasks.push ([task_ptr](std::stop_token) {
                (*task_ptr)();
            });

            return res;
        }

        template<typename F, typename... Args>
        std::future<std::invoke_result_t<F, std::stop_token, Args...»
        submit_with_stop(F&& f, Args&&... args)
        {
            std::cout « "submit_with_stop" « std::endl;

            using ReturnType = std::invoke_result_t<F, std::stop_token, Args...>;

            auto task_ptr = std::make_shared<std::packaged_task<ReturnType(std::stop_token)»(
                std::bind(std::forward<F>(f), std::placeholders::_1, std::forward<Args>(args)...)
            );
            std::future<ReturnType> res = task_ptr->get_future();

            tasks.push([task_ptr](std::stop_token tok) {
                (*task_ptr)(tok);
            });

            return res;
        }


    private:
        void thread_function (std::stop_token);

    };

    extern Thread_Pool & thread_pool;
}
```

7.64 main/Transform.hpp File Reference

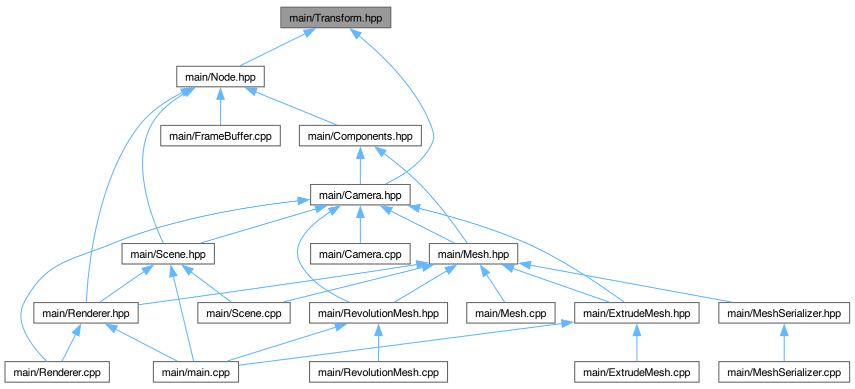
Implementation of the Transform class for 3D transformations.

```
#include <glm.hpp>
#include <gtc/matrix_transform.hpp>
#include <gtc/type_ptr.hpp>
```

Include dependency graph for Transform.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Ragot::Transform

  A class representing a 3D transformation with position, rotation, and scale.

Namespaces

- namespace Ragot

7.64.1 Detailed Description

Implementation of the Transform class for 3D transformations.

Author

Andrés Ragot (github.com/andresragot)

The Transform class provides methods to manage position, rotation, and scale of 3D objects.

Version

1.0

Date

2025-06-01

Copyright

7.65 Transform.hpp

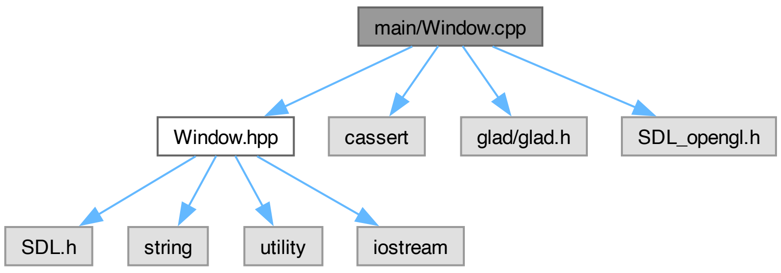
Go to the documentation of this file.

```


#pragma once

#include <glm.hpp>
#include <gtc/matrix_transform.hpp>         // translate, rotate, scale, perspective
#include <gtc/type_ptr.hpp>                 // value_ptr, quat

namespace Ragot
{
    using glm::vec3;
    using glm::mat4;

    class Transform
    {
    protected:
        vec3 position;
        vec3 rotation;
        vec3 scale;
        bool dirty = true;

    private:
        mat4 transform_matrix;

    public:
        Transform () : position (0.f), rotation (0.f), scale (1.f) {}

        virtual ~Transform () = default;

    public:
        void set_position (const vec3 & pos)
        {
            position = pos;
            dirty = true;
        }

        vec3 get_position () const { return position; }

        void set_rotation (const vec3 & rot)
        {
            rotation = rot;
            dirty = true;
        }

        vec3 get_rotation () const { return rotation; }

        void rotate (const float angle, const vec3 & axis)
        {
            rotation += angle * axis;
            if (rotation.x > 360.f) rotation.x -= 360.f;
            dirty = true;
        }

        void set_scale (const vec3 & scale)
        {
            this->scale = scale;
            dirty = true;
        }

        vec3 get_scale () const { return scale; }

        bool is_dirty () const { return dirty; }

    public:

        virtual mat4 get_transform_matrix ()
        {
            if (dirty)
            {
                dirty = false;

                mat4 identity(1);
                identity = glm::translate(identity, position);
                identity = glm::scale (identity, scale);

                glm::quat quaternion_rotation = glm::quat (glm::radians (rotation));
                identity *= glm::mat4_cast (quaternion_rotation);

                transform_matrix = identity;
            }

            return transform_matrix;
        }
    };
}
```

7.66 main/Window.cpp File Reference

```
#include "Window.hpp"
#include <cassert>
#include <glad/glad.h>
#include <SDL_opengl.h>
```

Include dependency graph for Window.cpp:



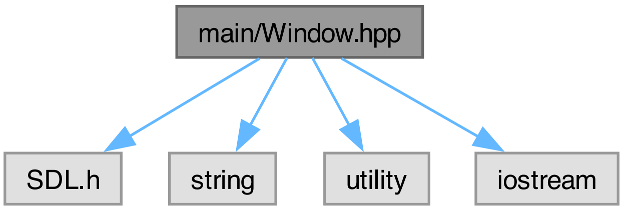
Namespaces

- namespace Ragot

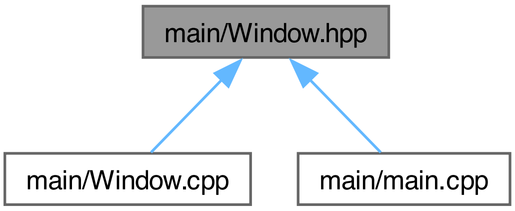
7.67 main/Window.hpp File Reference

```
#include <SDL.h>
#include <string>
#include <utility>
#include <iostream>
```

Include dependency graph for Window.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Ragot::Window

  Class for managing an SDL window with OpenGL context.
- struct Ragot::Window::OpenGL_Context_Settings

  Struct for OpenGL context settings.

Namespaces

- namespace Ragot

7.68 Window.hpp

Go to the documentation of this file.

```
/*
 *  This file is part of OpenGL-FinalProject
 *
 *  Developed by Andrés Ragot - github.com/andresragot
 *
 *  MIT License
 *
 *  Copyright (c) 2025 Andrés Ragot
 *
 *  Permission is hereby granted, free of charge, to any person obtaining a copy
 *  of this software and associated documentation files (the "Software"), to deal
 *  in the Software without restriction, including without limitation the rights
 *  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 *  copies of the Software, and to permit persons to whom the Software is
 *  furnished to do so, subject to the following conditions:
 *
 *  The above copyright notice and this permission notice shall be included in all
 *  copies or substantial portions of the Software.
 *
 *  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 *  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 *  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 *  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 *  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 *  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
 *  SOFTWARE.
 */

#pragma once

#include <SDL.h>
#include <string>
#include <utility>

#include <iostream>

namespace Ragot
{
    class Window
    {
    public:
        enum Position
        {
            UNDEFINED = SDL_WINDOWPOS_UNDEFINED,
            CENTERED  = SDL_WINDOWPOS_CENTERED,
        };

        struct OpenGL_Context_Settings
        {
            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;
        };

    private:
        SDL_Window* window_handle;
        SDL_GLContext opengl_context;

        unsigned width;
        unsigned height;

    public:
        Window(const std::string& title, int left_x, int top_y, unsigned width, unsigned height, const
OpenGL_Context_Settings& context_details)
            : Window(title.c_str(), left_x, top_y, width, height, context_details)
        {
        }

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

        ~Window();

    public:
        Window(const Window&) = delete;
        Window& operator=(const Window&) = delete;

        Window(Window&& other) noexcept
        {
            this->window_handle  = std::exchange(other.window_handle, nullptr);
            this->opengl_context = std::exchange(other.opengl_context, nullptr);
        }

        Window& operator=(Window&& other) noexcept
        {
            this->window_handle  = std::exchange(other.window_handle, nullptr);
            this->opengl_context = std::exchange(other.opengl_context, nullptr);

            return *this;
        }

        void swap_buffers()
        {
            SDL_GL_SwapWindow(window_handle);
        }

        unsigned get_width() { return width; }

        unsigned get_height() { return height; }
    };
}
```