OpenGL-FinalProject

Generated by Doxygen 1.13.2

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

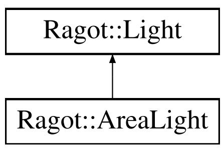
Class Documentation

4.1 Ragot::AreaLight Class Reference

Class for area light.

`#include <Ambient.hpp>`

Inheritance diagram for Ragot::AreaLight:



Public Member Functions

- AreaLight (const glm::vec3 &color, const glm::vec3 &position, const glm::vec3 &size)

  Constructor for the AreaLight class.

Public Member Functions inherited from Ragot::Light

- Light (const glm::vec3 &color)

  Constructor for the Light class.
- virtual ∼**Light** ()=default

  Virtual destructor for the Light class.

Public Attributes

- glm::vec3 **position**

  Position of the light.
- glm::vec3 **size**

  Size of the area light.

Public Attributes inherited from Ragot::Light

- glm::vec3 **color**

  Color of the light.

4.1.1 Detailed Description

Class for area light.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 AreaLight()

```
Ragot::AreaLight::AreaLight (
            const glm::vec3 & color,
            const glm::vec3 & position,
            const glm::vec3 & size) [inline]
```

Constructor for the AreaLight class.

Parameters

color	Color of the light.
position	Position of the light.
size	Size of the area light.

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

- Ambient.hpp

4.2 Ragot::Camera Class Reference

Class for managing a camera in OpenGL.

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

Public Member Functions

- Camera (float ratio=1.f)

  Constructor with default ratio.
- Camera (float near_z, float far_z, float ratio=1.f)

  Constructor with near and far clipping planes.
- Camera (float fov_degrees, float near_z, float far_z, float ratio)

  Constructor with field of view, near and far clipping planes, and aspect ratio.
- float get_fov () const

  Gets the field of view.
- float get_near_z () const

  Gets the near clipping plane.
- float get_far_z () const

  Gets the far clipping plane.
- float get_ratio () const

  Gets the aspect ratio.
- const Point & get_location () const

  Gets the location of the camera.
- const Point & get_target () const

  Gets the target point the camera is looking at.
- void set_fov (float new_fov)

  Sets the field of view and recalculates the projection matrix.
- void set_near_z (float new_near_z)

  Sets the near clipping plane and recalculates the projection matrix.
- void set_far_z (float new_far_z)

  Sets the far clipping plane and recalculates the projection matrix.
- void set_ratio (float new_ratio)

  Sets the aspect ratio and recalculates the projection matrix.
- void set_location (float x, float y, float z)

  Sets the location of the camera.
- void set_target (float x, float y, float z)

  Sets the target point the camera is looking at.
- void reset (float new_fov, float new_near_z, float new_far_z, float new_ratio)

  Resets the camera with new parameters and recalculates the projection matrix.
- void move (const glm::vec3 &translation)

  Moves the camera by a given translation vector.
- void rotate (const glm::mat4 &rotation)

  Rotates the camera by a given rotation matrix.
- const glm::mat4 & get_projection_matrix () const

  Gets the projection matrix of the camera.
- glm::mat4 get_transform_matrix_inverse () const

  Gets the inverse of the transformation matrix for the camera.

4.2.1 Detailed Description

Class for managing a camera in OpenGL.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Camera() [1/3]

```
Ragot::Camera::Camera (
            float ratio = 1.f)  [inline]
```

Constructor with default ratio.

Parameters

ratio	Aspect ratio (default is 1.0f).

4.2.2.2 Camera() [2/3]

```
Ragot::Camera::Camera (
            float near_z,
            float far_z,
            float ratio = 1.f)  [inline]
```

Constructor with near and far clipping planes.

Parameters

near↩_z	Near clipping plane.
far_z	Far clipping plane.
ratio	Aspect ratio (default is 1.0f).

4.2.2.3 Camera() [3/3]

```
Ragot::Camera::Camera (
            float fov_degrees,
            float near_z,
            float far_z,
            float ratio)  [inline]
```

Constructor with field of view, near and far clipping planes, and aspect ratio.

Parameters

fov_degrees	Field of view in degrees.
near_z	Near clipping plane.
far_z	Far clipping plane.
ratio	Aspect ratio.

4.2.3 Member Function Documentation

4.2.3.1 get_far_z()

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

Gets the far clipping plane.

Returns

Far clipping plane.

4.2.3.2 get_fov()

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

Gets the field of view.

Returns

Field of view in degrees.

4.2.3.3 get_location()

```
const Point & Ragot::Camera::get_location () const  [inline]
```

Gets the location of the camera.

Returns

Location of the camera.

4.2.3.4 get_near_z()

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

Gets the near clipping plane.

Returns

Near clipping plane.

4.2.3.5 get_projection_matrix()

```
const glm::mat4 & Ragot::Camera::get_projection_matrix () const  [inline]
```

Gets the projection matrix of the camera.

Returns

Projection matrix.

4.2.3.6 get_ratio()

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

Gets the aspect ratio.

Returns

Aspect ratio.

4.2.3.7 get_target()

```
const Point & Ragot::Camera::get_target () const  [inline]
```

Gets the target point the camera is looking at.

Returns

Target point.

4.2.3.8 get_transform_matrix_inverse()

```
glm::mat4 Ragot::Camera::get_transform_matrix_inverse () const  [inline]
```

Gets the inverse of the transformation matrix for the camera.

Returns

Inverse of the transformation matrix.

4.2.3.9 move()

```
void Ragot::Camera::move (
             const glm::vec3 & translation)  [inline]
```

Moves the camera by a given translation vector.

Parameters

translation	Translation vector.

4.2.3.10 reset()

```
void Ragot::Camera::reset (
             float new_fov,
             float new_near_z,
             float new_far_z,
             float new_ratio)  [inline]
```

Resets the camera with new parameters and recalculates the projection matrix.

Parameters

new_fov	New field of view in degrees.
new_near↩_z	New near clipping plane.
new_far_z	New far clipping plane.
new_ratio	New aspect ratio.

4.2.3.11 rotate()

```
void Ragot::Camera::rotate (
             const glm::mat4 & rotation)  [inline]
```

Rotates the camera by a given rotation matrix.

Parameters

rotation	Rotation matrix.

4.2.3.12 set_far_z()

```
void Ragot::Camera::set_far_z (
             float new_far_z)  [inline]
```

Sets the far clipping plane and recalculates the projection matrix.

Parameters

new_far↩ _z	New far clipping plane.

4.2.3.13 set_fov()

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

Sets the field of view and recalculates the projection matrix.

Parameters

new_fov	New field of view in degrees.

4.2.3.14 set_location()

```
void Ragot::Camera::set_location (
             float x,
             float y,
             float z)  [inline]
```

Sets the location of the camera.

Parameters

x	X-coordinate of the location.
y	Y-coordinate of the location.
z	Z-coordinate of the location.

4.2.3.15 set_near_z()

```
void Ragot::Camera::set_near_z (
             float new_near_z)  [inline]
```

Sets the near clipping plane and recalculates the projection matrix.

Parameters

new_near↩ _z	New near clipping plane.

4.2.3.16 set_ratio()

```
void Ragot::Camera::set_ratio (
             float new_ratio)  [inline]
```

Sets the aspect ratio and recalculates the projection matrix.

Parameters

new_ratio	New aspect ratio.

4.2.3.17 set_target()

```
void Ragot::Camera::set_target (
             float x,
             float y,
             float z)  [inline]
```

Sets the target point the camera is looking at.

Parameters

x	X-coordinate of the target.
y	Y-coordinate of the target.
z	Z-coordinate of the target.

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

- Camera.hpp

4.3 Ragot::Color_Buffer< COLOR > Class Template Reference

Template class for managing a color buffer.

```
#include <Color_Buffer.hpp>
```

Public Types

- using **Color** = COLOR
  Type alias for the color format.

Public Member Functions

- Color_Buffer (unsigned width, unsigned height)

  Constructor for the Color_Buffer class.
- unsigned get_width () const

  Gets the width of the buffer.
- unsigned get_height () const

  Gets the height of the buffer.
- Color ∗ colors ()

  Gets a pointer to the color data.
- const Color ∗ colors () const

  Gets a constant pointer to the color data.
- Color & get (unsigned offset)

  Gets the color at a specific offset.
- const Color & get (unsigned offset) const

  Gets the color at a specific offset (constant version).
- void set (unsigned offset, const Color &color)

  Sets the color at a specific offset.

4.3.1 Detailed Description

template<typename COLOR>
class Ragot::Color_Buffer< COLOR >

Template class for managing a color buffer.

Template Parameters

COLOR	The color format for the buffer.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Color_Buffer()

```
template<typename COLOR>
Ragot::Color_Buffer< COLOR >::Color_Buffer (
             unsigned width,
             unsigned height)  [inline]
```

Constructor for the Color_Buffer class.

Parameters

width	Width of the buffer.
height	Height of the buffer.

4.3.3 Member Function Documentation

4.3.3.1 colors() [1/2]

```
template<typename COLOR>
Color * Ragot::Color_Buffer< COLOR >::colors ()  [inline]
```

Gets a pointer to the color data.

Returns

Pointer to the color data.

4.3.3.2 colors() [2/2]

```
template<typename COLOR>
const Color * Ragot::Color_Buffer< COLOR >::colors () const  [inline]
```

Gets a constant pointer to the color data.

Returns

Constant pointer to the color data.

4.3.3.3 get() [1/2]

```
template<typename COLOR>
Color & Ragot::Color_Buffer< COLOR >::get (
            unsigned offset)  [inline]
```

Gets the color at a specific offset.

Parameters

offset	The offset in the buffer.

Returns

Reference to the color at the specified offset.

4.3.3.4 get() [2/2]

```
template<typename COLOR>
const Color & Ragot::Color_Buffer< COLOR >::get (
            unsigned offset) const  [inline]
```

Gets the color at a specific offset (constant version).

Parameters

offset	The offset in the buffer.

Returns

Constant reference to the color at the specified offset.

4.3.3.5 get_height()

```
template<typename COLOR>
unsigned Ragot::Color_Buffer< COLOR >::get_height () const  [inline]
```

Gets the height of the buffer.

Returns

Height of the buffer.

4.3.3.6 get_width()

```
template<typename COLOR>
unsigned Ragot::Color_Buffer< COLOR >::get_width () const  [inline]
```

Gets the width of the buffer.

Returns

Width of the buffer.

4.3.3.7 set()

```
template<typename COLOR>
void Ragot::Color_Buffer< COLOR >::set (
             unsigned offset,
             const Color & color)  [inline]
```

Sets the color at a specific offset.

Parameters

offset	The offset in the buffer.
color	The color to set.

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

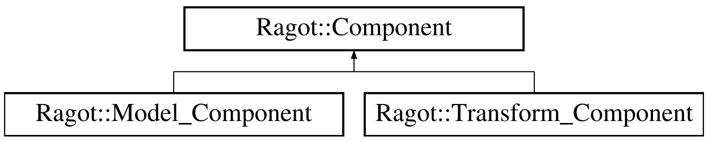
- Color_Buffer.hpp

4.4 Ragot::Component Class Reference

Base class for components.

```
#include <Component.hpp>
```

Inheritance diagram for Ragot::Component:



Public Member Functions

- virtual ∼**Component** ()=default

  Virtual destructor for the Component class.
- std::shared_ptr< Entity > get_entity () const

  Gets the entity associated with this component.
- void set_entity (std::shared_ptr< Entity > ent)

  Sets the entity associated with this component.
- bool get_has_task () const

  Checks if the component has a task.

Protected Attributes

- bool **has_task** = false

  Indicates whether the component has a task.

4.4.1 Detailed Description

Base class for components.

4.4.2 Member Function Documentation

4.4.2.1 get_entity()

```
std::shared_ptr< Entity > Ragot::Component::get_entity () const  [inline]
```

Gets the entity associated with this component.

Returns

Shared pointer to the associated entity.

4.4.2.2 get_has_task()

```
bool Ragot::Component::get_has_task () const  [inline]
```

Checks if the component has a task.

Returns

True if the component has a task, false otherwise.

4.4.2.3 set_entity()

```
void Ragot::Component::set_entity (
            std::shared_ptr< Entity > ent)  [inline]
```

Sets the entity associated with this component.

Parameters

ent	Shared pointer to the entity to associate.

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

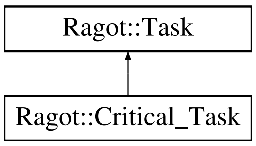
- Component.hpp

4.5 Ragot::Critical_Task Class Reference

Class to execute critical tasks such as rendering, which the Kernel can pause other tasks to execute. These tasks run in the main thread.

```
#include <Task.hpp>
```

Inheritance diagram for Ragot::Critical_Task:



Public Member Functions

- Critical_Task (function< void()> task_func)

  Constructor that calls the base class constructor.
- void execute () override

  Specific execution function for critical tasks.

Public Member Functions inherited from Ragot::Task

- Task (function< void()> task_func)

  Constructor that accepts the function to run for this task.
- virtual ~**Task** ()=default

  Default destructor.
- void **stop_execution** ()

  Stops execution of all tasks, even if executed by one thread.
- void **stop** ()

  Stops execution temporarily for critical sections of code.
- void **resume** ()

  Resumes execution after a stop.

Additional Inherited Members

Protected Member Functions inherited from Ragot::Task

- bool shouldStop ()

  Checks if the task should stop execution.
- bool shouldFinish ()

  Checks if the task should finish execution.
- void **wait_for_resume** ()

  Waits for resume signal to continue execution.

Protected Attributes inherited from Ragot::Task

- function< void()> **task_func**

  Function to run for this task.

4.5.1 Detailed Description

Class to execute critical tasks such as rendering, which the Kernel can pause other tasks to execute. These tasks run in the main thread.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 Critical_Task()

```
Ragot::Critical_Task::Critical_Task (
              function< void()> task_func)  [inline]
```

Constructor that calls the base class constructor.

Parameters

task_func	Function to run for this task.

4.5.3 Member Function Documentation

4.5.3.1 execute()

```
void Ragot::Critical_Task::execute ()  [override], [virtual]
```

Specific execution function for critical tasks.

Implements Ragot::Task.

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

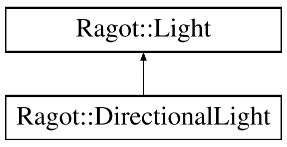
- Task.hpp
- Task.cpp

4.6 Ragot::DirectionalLight Class Reference

Class for directional light.

```
#include <Ambient.hpp>
```

Inheritance diagram for Ragot::DirectionalLight:



Public Member Functions

- DirectionalLight (const glm::vec3 &color, const glm::vec3 direction)

  Constructor for the DirectionalLight class.

Public Member Functions inherited from Ragot::Light

- Light (const glm::vec3 &color)

  Constructor for the Light class.
- virtual ∼**Light** ()=default

  Virtual destructor for the Light class.

Public Attributes

- glm::vec3 **direction**

  Direction of the light.

Public Attributes inherited from Ragot::Light

- glm::vec3 **color**

  Color of the light.

4.6.1 Detailed Description

Class for directional light.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 DirectionalLight()

```
Ragot::DirectionalLight::DirectionalLight (
            const glm::vec3 & color,
            const glm::vec3 direction)  [inline]
```

Constructor for the DirectionalLight class.

Parameters

color	Color of the light.
direction	Direction of the light.

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

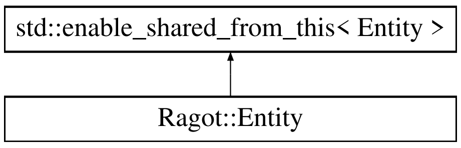
- Ambient.hpp

4.7 Ragot::Entity Class Reference

Class for managing entities in a scene.

```
#include <Entity.hpp>
```

Inheritance diagram for Ragot::Entity:



Public Member Functions

- void set_scene (Scene ∗scene)

  Sets the scene for this entity.
- const Scene ∗ get_scene ()

  Gets the scene this entity belongs to.
- const Scene ∗ get_scene () const

  Gets the scene this entity belongs to (const version).
- void add_component (shared_ptr< Component > component, const string &name)

  Adds a component to the entity.
- void remove_component (const string &name)

  Removes a component from the entity.
- void add_child (shared_ptr< Entity > child)

  Adds a child entity.
- void remove_child (shared_ptr< Entity > child)

  Removes a child entity.
- void set_transform_parent (Transform_Component ∗parent)

  Sets the parent transform component.
- const map< string, shared_ptr< Component > > & get_components () const

  Gets the components associated with this entity.

Public Attributes

- Transform_Component **transform**

  Transform component of the entity.

4.7.1 Detailed Description

Class for managing entities in a scene.

4.7.2 Member Function Documentation

4.7.2.1 add_child()

```
void Ragot::Entity::add_child (
            shared_ptr< Entity > child)  [inline]
```

Adds a child entity.

Parameters

child	Shared pointer to the child entity.

4.7.2.2 add_component()

```
void Ragot::Entity::add_component (
            shared_ptr< Component > component,
            const string & name)
```

Adds a component to the entity.

Parameters

component	Shared pointer to the component.
name	Name of the component.

4.7.2.3 get_components()

```
const map< string, shared_ptr< Component > > & Ragot::Entity::get_components () const  [inline]
```

Gets the components associated with this entity.

Returns

Map of components.

4.7.2.4 get_scene() [1/2]

```
const Scene * Ragot::Entity::get_scene ()  [inline]
```

Gets the scene this entity belongs to.

Returns

Pointer to the scene.

4.7.2.5 get_scene() [2/2]

```
const Scene * Ragot::Entity::get_scene () const  [inline]
```

Gets the scene this entity belongs to (const version).

Returns

Pointer to the scene.

4.7.2.6 remove_child()

```
void Ragot::Entity::remove_child (
             shared_ptr< Entity > child)  [inline]
```

Removes a child entity.

Parameters

child	Shared pointer to the child entity.

4.7.2.7 remove_component()

```
void Ragot::Entity::remove_component (
             const string & name)
```

Removes a component from the entity.

Parameters

name	Name of the component.

4.7.2.8 set_scene()

```
void Ragot::Entity::set_scene (
            Scene * scene)  [inline]
```

Sets the scene for this entity.

Parameters

scene	Pointer to the scene.

4.7.2.9 set_transform_parent()

```
void Ragot::Entity::set_transform_parent (
            Transform_Component * parent)  [inline]
```

Sets the parent transform component.

Parameters

parent	Pointer to the parent transform component.

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

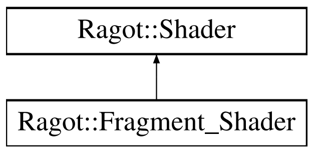
- Entity.hpp
- Entity.cpp

4.8 Ragot::Fragment_Shader Class Reference

Class for managing an OpenGL fragment shader.

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

Inheritance 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.

4.8.1 Detailed Description

Class for managing an OpenGL fragment shader.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 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.

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

- Shader_Program.hpp

4.9 Ragot::Frame_Buffer Class Reference

Class for managing a frame buffer in OpenGL.

```
#include <Postprocess.hpp>
```

Public Member Functions

- Frame_Buffer (unsigned width, unsigned height)

  Constructor for the Frame_Buffer class.
- **Frame_Buffer** ()=delete

  Default constructor is deleted.
- ∼**Frame_Buffer** ()

  Destructor for the Frame_Buffer class.
- void **bind_frame_buffer** () const

  Binds the frame buffer.
- void **unbind_frame_buffer** () const

  Unbinds the frame buffer.
- void **bind_texture** () const

  Binds the texture.
- void **unbind_texture** () const

  Unbinds the texture.
- void **render** ()

  Renders the frame buffer.

4.9.1 Detailed Description

Class for managing a frame buffer in OpenGL.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 Frame_Buffer()

```
Ragot::Frame_Buffer::Frame_Buffer (
                unsigned width,
                unsigned height)
```

Constructor for the Frame_Buffer class.

Parameters

width	Width of the frame buffer.
height	Height of the frame buffer.

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

- Postprocess.hpp
- Postprocess.cpp

4.10 Ragot::Kernel Class Reference

Class for managing the kernel that executes tasks.

`#include <MyKernel.hpp>`

Public Member Functions

- void add (std::shared_ptr< Task > new_task)

  Adds a new task to the kernel.
- void **run** ()

  Runs the kernel, executing all tasks.
- void **stop** ()

  Stops the kernel and all tasks.
- void **execute_critical** ()

  Executes all critical functions at once.

4.10.1 Detailed Description

Class for managing the kernel that executes tasks.

4.10.2 Member Function Documentation

4.10.2.1 add()

```
void Ragot::Kernel::add (
            std::shared_ptr< Task > new_task)
```

Adds a new task to the kernel.

Parameters

new_task	Shared pointer to the new task to add.

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

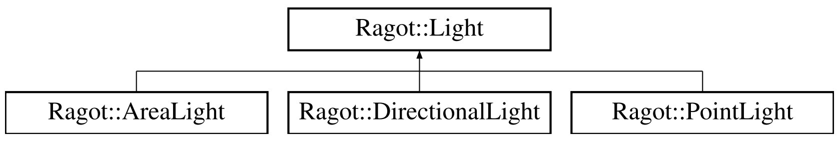
- MyKernel.hpp
- MyKernel.cpp

4.11 Ragot::Light Class Reference

Base class for different types of lights.

`#include <Ambient.hpp>`

Inheritance diagram for Ragot::Light:



Public Member Functions

- Light (const glm::vec3 &color)

  Constructor for the Light class.
- virtual ∼**Light** ()=default

  Virtual destructor for the Light class.

Public Attributes

- glm::vec3 **color**

  Color of the light.

4.11.1 Detailed Description

Base class for different types of lights.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 Light()

```
Ragot::Light::Light (
            const glm::vec3 & color)  [inline]
```

Constructor for the Light class.

Parameters

color	Color of the light.

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

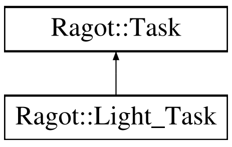
- Ambient.hpp

4.12 Ragot::Light_Task Class Reference

Class to execute cyclic tasks such as Update or Input.

```
#include <Task.hpp>
```

Inheritance diagram for Ragot::Light_Task:



Public Member Functions

- Light_Task (function< void()> task_func)

  Constructor that calls the base class constructor.
- void execute () override

  Specific execution function for light tasks.

Public Member Functions inherited from Ragot::Task

- Task (function< void()> task_func)

  Constructor that accepts the function to run for this task.
- virtual ~**Task** ()=default

  Default destructor.
- void **stop_execution** ()

  Stops execution of all tasks, even if executed by one thread.
- void **stop** ()

  Stops execution temporarily for critical sections of code.
- void **resume** ()

  Resumes execution after a stop.

Additional Inherited Members

Protected Member Functions inherited from Ragot::Task

- bool shouldStop ()

  Checks if the task should stop execution.
- bool shouldFinish ()

  Checks if the task should finish execution.
- void **wait_for_resume** ()

  Waits for resume signal to continue execution.

Protected Attributes inherited from Ragot::Task

- function< void()> **task_func**

  Function to run for this task.

4.12.1 Detailed Description

Class to execute cyclic tasks such as Update or Input.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 Light_Task()

```
Ragot::Light_Task::Light_Task (
            function< void()> task_func)  [inline]
```

Constructor that calls the base class constructor.

Parameters

task_func	Function to run for this task.

4.12.3 Member Function Documentation

4.12.3.1 execute()

```
void Ragot::Light_Task::execute ()  [override], [virtual]
```

Specific execution function for light tasks.

Implements Ragot::Task.

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

- Task.hpp
- Task.cpp

4.13 Ragot::Material Class Reference

Class for managing a material.

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

Public Member Functions

- **Material** ()=delete

  Deleted default constructor.
- Material (const vector< string > &source_code_vertex, const vector< string > &source_code_fragment, const string &texture_base_path)

  Constructor for the Material class.
- ∼**Material** ()=default

  Default destructor for the Material class.
- void **use_shader_program** ()

  Uses the shader program.
- GLint get_shader_program_uniform_location (const string &uniform)

  Gets the uniform location in the shader program.
- GLuint get_shader_program_id () const

  Gets the shader program ID.
- const bool bind_texture () const

  Binds the texture.
- const glm::vec3 get_color ()

  Gets the color of the material.
- const float get_shininess ()

  Gets the shininess of the material.

4.13.1 Detailed Description

Class for managing a material.

4.13.2 Constructor & Destructor Documentation

4.13.2.1 Material()

```
Ragot::Material::Material (
             const vector< string > & source_code_vertex,
             const vector< string > & source_code_fragment,
             const string & texture_base_path)
```

Constructor for the Material class.

Parameters

source_code_vertex	Vector of vertex shader source code.
source_code_fragment	Vector of fragment shader source code.
texture_base_path	Path to the base texture file.

4.13.3 Member Function Documentation

4.13.3.1 bind_texture()

```
const bool Ragot::Material::bind_texture () const  [inline]
```

Binds the texture.

Returns

True if the texture is successfully bound, false otherwise.

4.13.3.2 get_color()

```
const glm::vec3 Ragot::Material::get_color ()  [inline]
```

Gets the color of the material.

Returns

Color of the material.

4.13.3.3 get_shader_program_id()

```
GLuint Ragot::Material::get_shader_program_id () const  [inline]
```

Gets the shader program ID.

Returns

Shader program ID.

4.13.3.4 get_shader_program_uniform_location()

```
GLint Ragot::Material::get_shader_program_uniform_location (
             const string & uniform)  [inline]
```

Gets the uniform location in the shader program.

Parameters

uniform	Name of the uniform.

Returns

Uniform location.

4.13.3.5 get_shininess()

```
const float Ragot::Material::get_shininess ()  [inline]
```

Gets the shininess of the material.

Returns

Shininess of the material.

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

- Mesh.hpp
- Mesh.cpp

4.14 Ragot::Mesh Class Reference

Class for managing a 3D mesh.

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

Public Member Functions

- **Mesh** ()=default

  Default constructor for the Mesh class.
- Mesh (const std::string &mesh_file_path)

  Constructor for the Mesh class.
- ∼**Mesh** ()

  Destructor for the Mesh class.
- const vector< glm::vec3 > & get_coordinates () const

  Gets the vertex coordinates.
- const vector< glm::vec3 > & get_normals () const

  Gets the vertex normals.
- const vector< glm::vec2 > & get_textures_uv () const

  Gets the texture coordinates.
- const vector< GLuint > & get_indices () const

  Gets the indices.
- const GLuint get_vao_id () const

  Gets the Vertex Array Object ID.
- const GLsizei get_number_of_indices () const

  Gets the number of indices.

Protected Types

- enum {
  COORDINATES_VBO , NORMALS_VBO , TEXTURE_UVS_VBO , INDICES_EBO ,
  VBO_COUNT }

  Enum for VBO indices.

Protected Attributes

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

  Vector of vertex coordinates.
- vector< glm::vec3 > **normals**

  Vector of vertex normals.
- vector< glm::vec2 > **texture_coords**

  Vector of texture coordinates.
- vector< GLuint > **indices**

  Vector of indices.

4.14.1 Detailed Description

Class for managing a 3D mesh.

4.14.2 Member Enumeration Documentation

4.14.2.1 anonymous enum

```
anonymous enum  [protected]
```

Enum for VBO indices.

Enumerator

COORDINATES_VBO	VBO index for coordinates.
NORMALS_VBO	VBO index for normals.
TEXTURE_UVS_VBO	VBO index for texture UVs.
INDICES_EBO	VBO index for indices.
VBO_COUNT	Total number of VBOs.

4.14.3 Constructor & Destructor Documentation

4.14.3.1 Mesh()

```
Ragot::Mesh::Mesh (
             const std::string & mesh_file_path)
```

Constructor for the Mesh class.

Parameters

mesh_file_path	Path to the mesh file.

4.14.4 Member Function Documentation

4.14.4.1 get_coordinates()

```
const vector< glm::vec3 > & Ragot::Mesh::get_coordinates () const  [inline]
```

Gets the vertex coordinates.

Returns

Vector of vertex coordinates.

4.14.4.2 get_indices()

```
const vector< GLuint > & Ragot::Mesh::get_indices () const  [inline]
```

Gets the indices.

Returns

Vector of indices.

4.14.4.3 get_normals()

```
const vector< glm::vec3 > & Ragot::Mesh::get_normals () const  [inline]
```

Gets the vertex normals.

Returns

Vector of vertex normals.

4.14.4.4 get_number_of_indices()

```
const GLsizei Ragot::Mesh::get_number_of_indices () const  [inline]
```

Gets the number of indices.

Returns

Number of indices.

4.14.4.5 get_textures_uv()

```
const vector< glm::vec2 > & Ragot::Mesh::get_textures_uv () const  [inline]
```

Gets the texture coordinates.

Returns

Vector of texture coordinates.

4.14.4.6 get_vao_id()

```
const GLuint Ragot::Mesh::get_vao_id () const  [inline]
```

Gets the Vertex Array Object ID.

Returns

VAO ID.

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

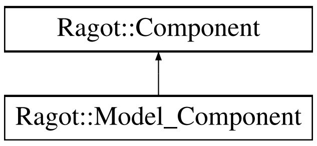
- Mesh.hpp
- Mesh.cpp

4.15 Ragot::Model_Component Class Reference

Component for managing models.

```
#include <Component.hpp>
```

Inheritance diagram for Ragot::Model_Component:



Public Member Functions

- **Model_Component** ()=delete

  Deleted default constructor.
- Model_Component (const string &model_file_path, const string &texture_file_path)

  Constructor for the Model_Component class.
- const GLuint get_shader_program_id () const

  Gets the shader program ID.
- void set_transparency (bool trans)

  Sets the transparency of the model.

Public Member Functions inherited from Ragot::Component

- virtual ∼**Component** ()=default

    Virtual destructor for the Component class.
- std::shared_ptr< Entity > get_entity () const

    Gets the entity associated with this component.
- void set_entity (std::shared_ptr< Entity > ent)

    Sets the entity associated with this component.
- bool get_has_task () const

    Checks if the component has a task.

Public Attributes

- Critical_Task **render_task**

    Task for rendering the model.
- Light_Task **update_task**

Additional Inherited Members

Protected Attributes inherited from Ragot::Component

- bool **has_task** = false

    Indicates whether the component has a task.

4.15.1 Detailed Description

Component for managing models.

4.15.2 Constructor & Destructor Documentation

4.15.2.1 Model_Component()

```
Ragot::Model_Component::Model_Component (
            const string & model_file_path,
            const string & texture_file_path)
```

Constructor for the Model_Component class.

Parameters

model_file_path	Path to the model file.
texture_file_path	Path to the texture file.

4.15.3 Member Function Documentation

4.15.3.1 get_shader_program_id()

```
const GLuint Ragot::Model_Component::get_shader_program_id () const  [inline]
```

Gets the shader program ID.

Returns

Shader program ID.

4.15.3.2 set_transparency()

```
void Ragot::Model_Component::set_transparency (
             bool trans)  [inline]
```

Sets the transparency of the model.

Parameters

trans	True to set the model as transparent, false otherwise.

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

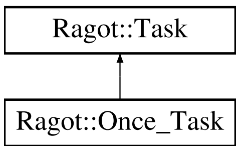
- Component.hpp
- Component.cpp

4.16 Ragot::Once_Task Class Reference

Class for tasks that are executed only once.

```
#include <Task.hpp>
```

Inheritance diagram for Ragot::Once_Task:



Public Member Functions

- Once_Task (function< void()> task_func)

  Constructor that calls the base class constructor.
- void execute () override

  Specific execution function for once-only tasks.

Public Member Functions inherited from Ragot::Task

- Task (function< void()> task_func)

  Constructor that accepts the function to run for this task.
- virtual ∼**Task** ()=default

  Default destructor.
- void **stop_execution** ()

  Stops execution of all tasks, even if executed by one thread.
- void **stop** ()

  Stops execution temporarily for critical sections of code.
- void **resume** ()

  Resumes execution after a stop.

Additional Inherited Members

Protected Member Functions inherited from Ragot::Task

- bool shouldStop ()

  Checks if the task should stop execution.
- bool shouldFinish ()

  Checks if the task should finish execution.
- void **wait_for_resume** ()

  Waits for resume signal to continue execution.

Protected Attributes inherited from Ragot::Task

- function< void()> **task_func**

  Function to run for this task.

4.16.1 Detailed Description

Class for tasks that are executed only once.

4.16.2 Constructor & Destructor Documentation

4.16.2.1 Once_Task()

```
Ragot::Once_Task::Once_Task (
            function< void()> task_func)  [inline]
```

Constructor that calls the base class constructor.

Parameters

task_func	Function to run for this task.

4.16.3 Member Function Documentation

4.16.3.1 execute()

```
void Ragot::Once_Task::execute ()  [override], [virtual]
```

Specific execution function for once-only tasks.

Implements Ragot::Task.

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

- Task.hpp
- Task.cpp

4.17 Ragot::Window::OpenGL_Context_Settings Struct Reference

Struct for OpenGL context settings.

```
#include <Window.hpp>
```

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.

4.17.1 Detailed Description

Struct for OpenGL context settings.

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

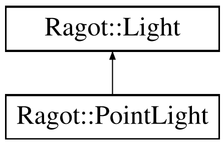
- Window.hpp

4.18 Ragot::PointLight Class Reference

Class for point light.

```
#include <Ambient.hpp>
```

Inheritance diagram for Ragot::PointLight:



Public Member Functions

- PointLight (const glm::vec3 &color, const glm::vec3 &position)

    Constructor for the PointLight class.

Public Member Functions inherited from Ragot::Light

- Light (const glm::vec3 &color)

    Constructor for the Light class.
- virtual ∼**Light** ()=default

    Virtual destructor for the Light class.

Public Attributes

- glm::vec3 **position**

    Position of the light.

Public Attributes inherited from Ragot::Light

- glm::vec3 **color**

    Color of the light.

4.18.1 Detailed Description

Class for point light.

4.18.2 Constructor & Destructor Documentation

4.18.2.1 PointLight()

```
Ragot::PointLight::PointLight (
            const glm::vec3 & color,
            const glm::vec3 & position)  [inline]
```

Constructor for the PointLight class.

Parameters

color	Color of the light.
position	Position of the light.

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

- Ambient.hpp

4.19 Ragot::Rgba8888 Union Reference

Union for managing RGBA color values.

```
#include <Color.hpp>
```

Public Types

- enum { **RED** , **GREEN** , **BLUE** , **ALPHA** }
  Enum for the RGBA component indices.

Public Attributes

- uint32_t **value**
  32-bit RGBA color value.
- uint8_t **components** [4]
  Array of RGBA components.

4.19.1 Detailed Description

Union for managing RGBA color values.

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

- Color.hpp

4.20 Ragot::Scene Class Reference

Class for managing a scene in OpenGL.

```
#include <MySystem.hpp>
```

Public Member Functions

- void resize (int width, int height)

  Resizes the scene.
- void on_drag (int pointer_x, int pointer_y)

  Handles pointer drag events.
- void on_click (int pointer_x, int pointer_y, bool down)

  Handles pointer click events.
- void on_translation (glm::vec3 translation)

  Handles translation events.
- void on_shift_pressed (bool down)

  Handles shift key press events.
- void **update** ()

  Updates the scene.
- void **render** ()

  Renders the scene.
- void **postproccess** ()

  Post-processes the scene.
- **Scene** ()

  Constructor for the Scene class.
- void add_entities (shared_ptr< Entity > entity, const string &name)

  Adds an entity to the scene.
- void remove_entities (const string &name)

  Removes an entity from the scene.
- shared_ptr< Entity > get_entity (const string &name) const

  Gets an entity from the scene.
- shared_ptr< Camera > get_camera () const

  Gets the camera of the scene.

4.20.1 Detailed Description

Class for managing a scene in OpenGL.

4.20.2 Member Function Documentation

4.20.2.1 add_entities()

```
void Ragot::Scene::add_entities (
            shared_ptr< Entity > entity,
            const string & name)
```

Adds an entity to the scene.

Parameters

entity	Shared pointer to the entity.
name	Name of the entity.

4.20.2.2 get_camera()

```
shared_ptr< Camera > Ragot::Scene::get_camera () const  [inline]
```

Gets the camera of the scene.

Returns

Shared pointer to the camera.

4.20.2.3 get_entity()

```
shared_ptr< Entity > Ragot::Scene::get_entity (
             const string & name) const
```

Gets an entity from the scene.

Parameters

name	Name of the entity.

Returns

Shared pointer to the entity.

4.20.2.4 on_click()

```
void Ragot::Scene::on_click (
             int pointer_x,
             int pointer_y,
             bool down)
```

Handles pointer click events.

Parameters

pointer←_x	X-coordinate of the pointer.
pointer←_y	Y-coordinate of the pointer.
down	Indicates if the pointer is pressed down.

4.20.2.5 on_drag()

```
void Ragot::Scene::on_drag (
             int pointer_x,
             int pointer_y)
```

Handles pointer drag events.

Parameters

pointer↩ _x	X-coordinate of the pointer.
pointer↩ _y	Y-coordinate of the pointer.

4.20.2.6 on_shift_pressed()

```
void Ragot::Scene::on_shift_pressed (
            bool down)
```

Handles shift key press events.

Parameters

down	Indicates if the shift key is pressed down.

4.20.2.7 on_translation()

```
void Ragot::Scene::on_translation (
            glm::vec3 translation)
```

Handles translation events.

Parameters

translation	Translation vector.

4.20.2.8 remove_entities()

```
void Ragot::Scene::remove_entities (
            const string & name)
```

Removes an entity from the scene.

Parameters

name	Name of the entity.

4.20.2.9 resize()

```
void Ragot::Scene::resize (
            int width,
            int height)
```

Resizes the scene.

Parameters

width	New width of the scene.
height	New height of the scene.

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

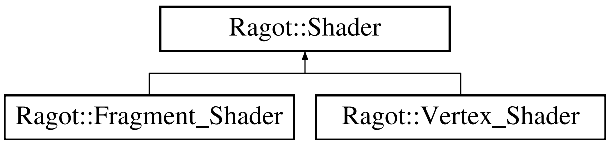
- MySystem.hpp
- MySystem.cpp

4.21 Ragot::Shader Class Reference

Class for managing an OpenGL shader.

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

Inheritance 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.

4.21.1 Detailed Description

Class for managing an OpenGL shader.

4.21.2 Constructor & Destructor Documentation

4.21.2.1 Shader()

```
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).

4.21.3 Member Function Documentation

4.21.3.1 compile_shader()

```
GLuint Ragot::Shader::compile_shader () [protected]
```

Compiles the shader.

Returns

Shader ID.

4.21.3.2 get_error()

```
string * Ragot::Shader::get_error () [inline]
```

Gets the compilation error message.

Returns

Pointer to the error message string.

4.21.3.3 get_id()

```
GLuint Ragot::Shader::get_id () const [inline]
```

Gets the shader ID.

Returns

Shader ID.

4.21.3.4 is_ok()

```
bool Ragot::Shader::is_ok () const  [inline]
```

Checks if the shader is compiled successfully.

Returns

True if compilation succeeded, false otherwise.

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

- Shader_Program.hpp
- Shader_Program.cpp

4.22 Ragot::Shader_Program Class Reference

Class for managing an OpenGL shader program.

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

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.

4.22.1 Detailed Description

Class for managing an OpenGL shader program.

4.22.2 Constructor & Destructor Documentation

4.22.2.1 Shader_Program()

```
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.

4.22.3 Member Function Documentation

4.22.3.1 get_id()

```
GLuint Ragot::Shader_Program::get_id () const  [inline]
```

Gets the shader program ID.

Returns

Shader program ID.

4.22.3.2 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.

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

- Shader_Program.hpp
- Shader_Program.cpp

4.23 Ragot::Skybox Class Reference

Class for rendering a skybox in the scene.

```
#include <Ambient.hpp>
```

Public Member Functions

- Skybox (const string &texture_path)

  Constructor for the Skybox class.
- ∼**Skybox** ()

  Destructor for the Skybox class.
- void set_camera (shared_ptr< Camera > cam)

  Sets the camera for the skybox.
- void **render** ()

  Renders the skybox.

4.23.1 Detailed Description

Class for rendering a skybox in the scene.

4.23.2 Constructor & Destructor Documentation

4.23.2.1 Skybox()

```
Ragot::Skybox::Skybox (
            const string & texture_path)
```

Constructor for the Skybox class.

Parameters

texture_path	Path to the texture for the skybox.

4.23.3 Member Function Documentation

4.23.3.1 set_camera()

```
void Ragot::Skybox::set_camera (
            shared_ptr< Camera > cam)  [inline]
```

Sets the camera for the skybox.

Parameters

cam	Shared pointer to the camera.

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

- Ambient.hpp
- Ambient.cpp

4.24 Ragot::System Class Reference

Class for managing the system in OpenGL.

```
#include <MySystem.hpp>
```

Public Member Functions

- System (const string &Window_Name, const int width, const int height)

  Constructor for the System class.
- **System** ()

  Default constructor for the System class.
- ∼**System** ()

  Destructor for the System class.
- void add_entities (shared_ptr< Entity > entity, const string &name)

  Adds an entity to the system.
- void **run** ()

  Runs the system.
- void **stop** ()

  Stops the system.

4.24.1 Detailed Description

Class for managing the system in OpenGL.

4.24.2 Constructor & Destructor Documentation

4.24.2.1 System()

```
Ragot::System::System (
            const string & Window_Name,
            const int width,
            const int height)
```

Constructor for the System class.

Parameters

Window_Name	Name of the window.
width	Width of the window.
height	Height of the window.

4.24.3 Member Function Documentation

4.24.3.1 add_entities()

```
void Ragot::System::add_entities (
            shared_ptr< Entity > entity,
            const string & name)
```

Adds an entity to the system.

Parameters

entity	Shared pointer to the entity.
name	Name of the entity.

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

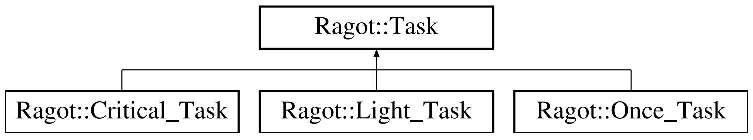
- MySystem.hpp
- MySystem.cpp

4.25 Ragot::Task Class Reference

Base class for managing tasks.

```
#include <Task.hpp>
```

Inheritance diagram for Ragot::Task:



Public Member Functions

- Task (function< void()> task_func)

  Constructor that accepts the function to run for this task.
- virtual ∼**Task** ()=default

  Default destructor.
- virtual void execute ()=0

  Virtual function to execute the task, to be overridden by derived classes.
- void **stop_execution** ()

  Stops execution of all tasks, even if executed by one thread.
- void **stop** ()

  Stops execution temporarily for critical sections of code.
- void **resume** ()

  Resumes execution after a stop.

Protected Member Functions

- bool shouldStop ()

  Checks if the task should stop execution.
- bool shouldFinish ()

  Checks if the task should finish execution.
- void **wait_for_resume** ()

  Waits for resume signal to continue execution.

Protected Attributes

- function< void()> **task_func**

    Function to run for this task.

4.25.1 Detailed Description

Base class for managing tasks.

4.25.2 Constructor & Destructor Documentation

4.25.2.1 Task()

```
Ragot::Task::Task (
            function< void()> task_func)  [inline]
```

Constructor that accepts the function to run for this task.

Parameters

task_func	Function to run for this task.

4.25.3 Member Function Documentation

4.25.3.1 execute()

```
virtual void Ragot::Task::execute ()  [pure virtual]
```

Virtual function to execute the task, to be overridden by derived classes.

Implemented in Ragot::Critical_Task, Ragot::Light_Task, and Ragot::Once_Task.

4.25.3.2 shouldFinish()

```
bool Ragot::Task::shouldFinish ()  [inline], [protected]
```

Checks if the task should finish execution.

Returns

True if the task should finish, false otherwise.

4.25.3.3 shouldStop()

```
bool Ragot::Task::shouldStop ()  [inline], [protected]
```

Checks if the task should stop execution.

Returns

True if the task should stop, false otherwise.

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

- Task.hpp
- Task.cpp

4.26 Ragot::Terrain Class Reference

Class for rendering a terrain in the scene.

```
#include <Ambient.hpp>
```

Public Member Functions

- Terrain (float width, float depth, unsigned x_slices, unsigned z_slices)

  Constructor for the Terrain class.
- ∼**Terrain** ()

  Destructor for the Terrain class.
- void set_camera (shared_ptr< Camera > cam)

  Sets the camera for the terrain.
- void **render** ()

  Renders the terrain.

4.26.1 Detailed Description

Class for rendering a terrain in the scene.

4.26.2 Constructor & Destructor Documentation

4.26.2.1 Terrain()

```
Ragot::Terrain::Terrain (
             float width,
             float depth,
             unsigned x_slices,
             unsigned z_slices)
```

Constructor for the Terrain class.

Parameters

width	Width of the terrain.
depth	Depth of the terrain.
x_slices	Number of slices along the x-axis.
z_slices	Number of slices along the z-axis.

4.26.3 Member Function Documentation

4.26.3.1 set_camera()

```
void Ragot::Terrain::set_camera (
            shared_ptr< Camera > cam)  [inline]
```

Sets the camera for the terrain.

Parameters

cam	Shared pointer to the camera.

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

- Ambient.hpp
- Ambient.cpp

4.27 Ragot::Texture2D< COLOR_FORMAT > Class Template Reference

Template class for managing a 2D texture.

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

Public Member Functions

- Texture2D (const string &texture_base_path)

  Constructor for the Texture2D class.
- ∼**Texture2D** ()

  Destructor for the Texture2D class.
- bool is_ok () const

  Checks if the texture is loaded.
- virtual bool bind () const

  Binds the texture.

Protected Types

- typedef Color_Buffer< COLOR_FORMAT > **Color_Buffer**

  Type alias for color buffer.

Protected Member Functions

- **Texture2D** ()

  Default constructor for the Texture2D class.
- GLint create_texture_2d (const string &texture_path)

  Creates a 2D texture from a file.
- unique_ptr< Color_Buffer > load_image (const string &texture_path)

  Loads an image from a file.

Protected Attributes

- GLuint **texture_id**

  Texture ID.
- bool **texture_is_loaded**

  Indicates if the texture is loaded.

4.27.1 Detailed Description

template<typename COLOR_FORMAT>
class Ragot::Texture2D< COLOR_FORMAT >

Template class for managing a 2D texture.

Template Parameters

COLOR_FORMAT	Color format of the texture.

4.27.2 Constructor & Destructor Documentation

4.27.2.1 Texture2D()

```
template<typename COLOR_FORMAT>
Ragot::Texture2D< COLOR_FORMAT >::Texture2D (
            const string & texture_base_path)
```

Constructor for the Texture2D class.

Parameters

texture_base_path	Path to the base texture file.

4.27.3 Member Function Documentation

4.27.3.1 bind()

```
template<typename COLOR_FORMAT>
virtual bool Ragot::Texture2D< COLOR_FORMAT >::bind () const  [inline], [virtual]
```

Binds the texture.

Returns

True if the texture is successfully bound, false otherwise.

Reimplemented in Ragot::Texture_Cube.

4.27.3.2 create_texture_2d()

```
template<typename COLOR_FORMAT>
GLint Ragot::Texture2D< COLOR_FORMAT >::create_texture_2d (
            const string & texture_path)  [protected]
```

Creates a 2D texture from a file.

Parameters

texture_path	Path to the texture file.

Returns

The texture ID.

4.27.3.3 is_ok()

```
template<typename COLOR_FORMAT>
bool Ragot::Texture2D< COLOR_FORMAT >::is_ok () const  [inline]
```

Checks if the texture is loaded.

Returns

True if the texture is loaded, false otherwise.

4.27.3.4 load_image()

```
template<typename COLOR_FORMAT>
unique_ptr< Color_Buffer< COLOR_FORMAT > > Ragot::Texture2D< COLOR_FORMAT >::load_image (
            const string & texture_path)  [protected]
```

Loads an image from a file.

Parameters

texture_path	Path to the texture file.

Returns

Unique pointer to the color buffer.

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

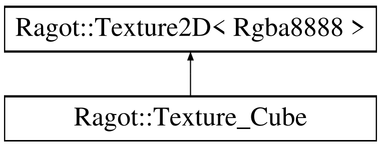
- Mesh.hpp
- Mesh.cpp

4.28 Ragot::Texture_Cube Class Reference

Class for managing a cube texture.

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

Inheritance diagram for Ragot::Texture_Cube:



Public Member Functions

- Texture_Cube (const string &texture_base_path)

    Constructor for the Texture_Cube class.
- bool bind () const override

    Binds the cube texture.

Public Member Functions inherited from Ragot::Texture2D< Rgba8888 >

- Texture2D (const string &texture_base_path)

    Constructor for the Texture2D class.
- ∼**Texture2D** ()

    Destructor for the Texture2D class.
- bool is_ok () const

    Checks if the texture is loaded.

Additional Inherited Members

Protected Types inherited from Ragot::Texture2D< Rgba8888 >

- typedef Color_Buffer< Rgba8888 > **Color_Buffer**

    Type alias for color buffer.

Protected Member Functions inherited from Ragot::Texture2D< Rgba8888 >

- **Texture2D** ()

  Default constructor for the Texture2D class.
- GLint create_texture_2d (const string &texture_path)

  Creates a 2D texture from a file.
- unique_ptr< Color_Buffer > load_image (const string &texture_path)

  Loads an image from a file.

Protected Attributes inherited from Ragot::Texture2D< Rgba8888 >

- GLuint **texture_id**

  Texture ID.
- bool **texture_is_loaded**

  Indicates if the texture is loaded.

4.28.1 Detailed Description

Class for managing a cube texture.

4.28.2 Constructor & Destructor Documentation

4.28.2.1 Texture_Cube()

```
Ragot::Texture_Cube::Texture_Cube (
             const string & texture_base_path)
```

Constructor for the Texture_Cube class.

Parameters

texture_base_path	Path to the base texture file.

4.28.3 Member Function Documentation

4.28.3.1 bind()

```
bool Ragot::Texture_Cube::bind () const  [inline], [override], [virtual]
```

Binds the cube texture.

Returns

True if the texture is successfully bound, false otherwise.

Reimplemented from Ragot::Texture2D< Rgba8888 >.

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

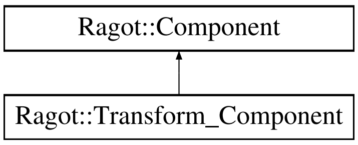
- Mesh.hpp
- Mesh.cpp

4.29 Ragot::Transform_Component Class Reference

Component for managing transformations.

```
#include <Component.hpp>
```

Inheritance diagram for Ragot::Transform_Component:



Public Member Functions

- **Transform_Component** ()

  Constructor for the Transform_Component class.
- mat4 get_transform_matrix ()

  Gets the transformation matrix.
- void set_position (const vec3 &pos)

  Sets the position.
- vec3 get_position () const

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

  Sets the rotation.
- vec3 get_rotation () const

  Gets the rotation.
- void set_scale (const vec3 &scal)

  Sets the scale.
- vec3 get_scale () const

  Gets the scale.
- void set_parent (Transform_Component ∗par)

  Sets the parent transformation component.
- Transform_Component ∗ get_parent () const

  Gets the parent transformation component.

Public Member Functions inherited from Ragot::Component

- virtual ∼**Component** ()=default

  Virtual destructor for the Component class.
- std::shared_ptr< Entity > get_entity () const

  Gets the entity associated with this component.
- void set_entity (std::shared_ptr< Entity > ent)

  Sets the entity associated with this component.
- bool get_has_task () const

  Checks if the component has a task.

Additional Inherited Members

Protected Attributes inherited from Ragot::Component

- bool **has_task** = false

  Indicates whether the component has a task.

4.29.1 Detailed Description

Component for managing transformations.

4.29.2 Member Function Documentation

4.29.2.1 get_parent()

```
Transform_Component * Ragot::Transform_Component::get_parent () const  [inline]
```

Gets the parent transformation component.

Returns

Pointer to the parent transformation component.

4.29.2.2 get_position()

```
vec3 Ragot::Transform_Component::get_position () const  [inline]
```

Gets the position.

Returns

Current position.

4.29.2.3 get_rotation()

```
vec3 Ragot::Transform_Component::get_rotation () const  [inline]
```

Gets the rotation.

Returns

Current rotation.

4.29.2.4 get_scale()

```
vec3 Ragot::Transform_Component::get_scale () const  [inline]
```

Gets the scale.

Returns

Current scale.

4.29.2.5 get_transform_matrix()

```
mat4 Ragot::Transform_Component::get_transform_matrix ()  [inline]
```

Gets the transformation matrix.

Returns

Transformation matrix.

4.29.2.6 set_parent()

```
void Ragot::Transform_Component::set_parent (
             Transform_Component * par)  [inline]
```

Sets the parent transformation component.

Parameters

par	Pointer to the parent transformation component.

4.29.2.7 set_position()

```
void Ragot::Transform_Component::set_position (
             const vec3 & pos)  [inline]
```

Sets the position.

Parameters

pos	New position.

4.29.2.8 set_rotation()

```
void Ragot::Transform_Component::set_rotation (
             const vec3 & rot)  [inline]
```

Sets the rotation.

Parameters

rot	New rotation.

4.29.2.9 set_scale()

```
void Ragot::Transform_Component::set_scale (
             const vec3 & scal)  [inline]
```

Sets the scale.

Parameters

scal	New scale.

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

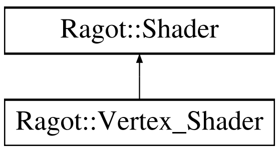
- Component.hpp

4.30 Ragot::Vertex_Shader Class Reference

Class for managing an OpenGL vertex shader.

`#include <Shader_Program.hpp>`

Inheritance 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.

4.30.1 Detailed Description

Class for managing an OpenGL vertex shader.

4.30.2 Constructor & Destructor Documentation

4.30.2.1 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.

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

- Shader_Program.hpp

4.31 Ragot::Window Class Reference

Class for managing an SDL window with OpenGL context.

```
#include <Window.hpp>
```

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.

4.31.1 Detailed Description

Class for managing an SDL window with OpenGL context.

4.31.2 Member Enumeration Documentation

4.31.2.1 Position

```
enum Ragot::Window::Position
```

Enum for window position.

Enumerator

UNDEFINED	Undefined position.
CENTERED	Centered position.

4.31.3 Constructor & Destructor Documentation

4.31.3.1 Window() [1/3]

```
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.

4.31.3.2 Window() [2/3]

```
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.

4.31.3.3 Window() [3/3]

```
Ragot::Window::Window (
            Window && other)  [inline], [noexcept]
```

Move constructor for the Window class.

Parameters

other	Other window to move from.

4.31.4 Member Function Documentation

4.31.4.1 get_height()

```
unsigned Ragot::Window::get_height ()  [inline]
```

Gets the height of the window.

Returns

Height of the window.

4.31.4.2 get_width()

```
unsigned Ragot::Window::get_width ()  [inline]
```

Gets the width of the window.

Returns

Width of the window.

4.31.4.3 operator=()

```
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.

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

- Window.hpp
- Window.cpp

Chapter 5

File Documentation

5.1 Ambient.hpp

```
/*
 *  This file is part of OpenGL-FinalProject
 *
 *  Developed by Andrés Ragot - github.com/andresragot
 *
 *  MIT License
 *
 *  Copyright (c) 2024 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 "Camera.hpp"
#include "Shader_Program.hpp"
#include "Mesh.hpp"
#include "Color.hpp"

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

namespace Ragot
{
    using namespace std;

    class Skybox
    {
    private:
        Shader_Program shader_program;
        static const GLfloat coordinates[];
        static const string vertex_shader_code;
        static const string fragment_shader_code;
        GLuint vbo_id;
        GLuint vao_id;
        GLint model_view_matrix_id;
        GLint projection_matrix_id;
        shared_ptr<Camera> camera = nullptr;
        Texture_Cube texture_cube;

    public:
        Skybox(const string & texture_path);

        ~Skybox();

        void set_camera(shared_ptr<Camera> cam) { camera = cam; }

        void render();
    };

    class Terrain
    {
    private:
        enum
        {
            COORDINATES_VBO,
            TEXTURE_UVS_VBO,
            INDICES_EBO,
            VBO_COUNT
        };

        Shader_Program shader_program;
        static const string vertex_shader_code;
        static const string fragment_shader_code;
        GLsizei number_of_vertices;
        GLsizei number_of_indices;
        GLuint vbo_ids[VBO_COUNT];
        GLuint vao_id;
        GLint model_view_matrix_id;
        GLint projection_matrix_id;
        GLint view_position_id;
        GLint light_position_id;
        shared_ptr< Camera > camera = nullptr;
        Texture2D< Monochrome8 > texture;

    public:
        Terrain(float width, float depth, unsigned x_slices, unsigned z_slices);

        ~Terrain();

        void set_camera(shared_ptr<Camera> cam) { camera = cam; }

        void render();
    };

    class Light
    {
    public:
        glm::vec3 color;

        Light(const glm::vec3 & color) : color(color) {}

        virtual ~Light() = default;
    };

    class DirectionalLight : public Light
    {
    public:
        glm::vec3 direction;

        DirectionalLight(const glm::vec3 & color, const glm::vec3 direction)
            : Light(color), direction(direction) {}
    };

    class PointLight : public Light
    {
    public:
        glm::vec3 position;

        PointLight(const glm::vec3 & color, const glm::vec3 & position)
            : Light(color), position(position) {}
    };

    class AreaLight : public Light
    {
    public:
        glm::vec3 position;
        glm::vec3 size;

        AreaLight(const glm::vec3 & color, const glm::vec3 & position, const glm::vec3 & size)
            : Light(color), position(position), size(size) {}
    };
}
```

5.2 Camera.hpp

```
/*
 *  This file is part of OpenGL-FinalProject
 *
 *  Developed by Andrés Ragot - github.com/andresragot
 *
 *  MIT License
 *
 *  Copyright (c) 2024 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 <glm.hpp>                          // vec3, vec4, ivec4, mat4
#include <gtc/matrix_transform.hpp>         // translate, rotate, scale, perspective
#include <gtc/type_ptr.hpp>                 // value_ptr


namespace Ragot
{
    class Camera
    {
        using Point = glm::vec4;
        using Vector = glm::vec4;
        using Matrix44 = glm::mat4;

    private:
        float fov;
        float near_z;
        float far_z;
        float ratio;

        Point location;
        Point target;

        Matrix44 projection_matrix;

    public:
        Camera(float ratio = 1.f)
        {
            reset(60.f, 0.1f, 1000.f, ratio);
        }

        Camera(float near_z, float far_z, float ratio = 1.f)
        {
            reset(60.f, near_z, far_z, ratio);
        }

        Camera(float fov_degrees, float near_z, float far_z, float ratio)
        {
            reset(fov_degrees, near_z, far_z, ratio);
        }

        float get_fov() const { return fov; }

        float get_near_z() const { return near_z; }

        float get_far_z() const { return far_z; }

        float get_ratio() const { return ratio; }

        const Point & get_location() const { return location; }

        const Point & get_target() const { return target; }

        void set_fov(float new_fov) { fov = new_fov; calculate_projection_matrix(); }

        void set_near_z(float new_near_z) { near_z = new_near_z; calculate_projection_matrix(); }

        void set_far_z(float new_far_z) { far_z = new_far_z; calculate_projection_matrix(); }

        void set_ratio(float new_ratio) { ratio = new_ratio; calculate_projection_matrix(); }

        void set_location(float x, float y, float z) { location[0] = x; location[1] = y; location[2] = z; }

        void set_target(float x, float y, float z) { target[0] = x; target[1] = y; target[2] = z; }

        void reset(float new_fov, float new_near_z, float new_far_z, float new_ratio)
        {
            set_fov(new_fov);
            set_near_z(new_near_z);
            set_far_z(new_far_z);
            set_ratio(new_ratio);
            set_location(0.f, 0.f, 0.f);
            set_target(0.f, 0.f, -1.f);
            calculate_projection_matrix();
        }

        void move(const glm::vec3 & translation)
        {
            location += glm::vec4(translation, 0.f);
            target   += glm::vec4(translation, 1.f);
        }

        void rotate(const glm::mat4 & rotation)
        {
            target = location + rotation * (target - location);
        }

        const glm::mat4 & get_projection_matrix() const
        {
            return projection_matrix;
        }

        glm::mat4 get_transform_matrix_inverse() const
        {
            return glm::lookAt(
                glm::vec3(location[0], location[1], location[2]),
                glm::vec3(target[0], target[1], target[2]),
                glm::vec3(0.0f, 1.0f, 0.0f)
            );
        }

    private:
        void calculate_projection_matrix()
        {
            projection_matrix = glm::perspective(glm::radians(fov), ratio, near_z, far_z);
        }
    };
}
```

5.3 Color.hpp

```
/*
 *  This file is part of OpenGL-FinalProject
 *
 *  Developed by Andrs Ragot - github.com/andresragot
 *
 *  MIT License
 *
 *  Copyright (c) 2024 Andrs 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 <cstdint>

namespace Ragot
{
    using Monochrome8 = uint8_t;

    union Rgba8888
    {
        enum { RED, GREEN, BLUE, ALPHA };

        uint32_t value;
        uint8_t  components[4];
    };
}
```

5.4 Color_Buffer.hpp

```
/*
 *  This file is part of OpenGL-FinalProject
 *
 *  Developed by Andrs Ragot - github.com/andresragot
 *
 *  MIT License
 *
 *  Copyright (c) 2024 Andrs 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 <vector>

namespace Ragot
{
    template<typename COLOR>
    class Color_Buffer
    {
    public:
        using Color = COLOR;

    private:
        unsigned width;
        unsigned height;

        std::vector<Color> buffer;

    public:
        Color_Buffer(unsigned width, unsigned height)
        :
            width(width),
            height(height),
            buffer(width * height)
        {
        }

        unsigned get_width() const
        {
            return width;
        }

        unsigned get_height() const
        {
            return height;
        }

        Color* colors()
        {
            return buffer.data();
        }

        const Color* colors() const
        {
            return buffer.data();
        }

        Color& get(unsigned offset)
        {
            return buffer[offset];
        }

        const Color& get(unsigned offset) const
        {
            return buffer[offset];
        }

        void set(unsigned offset, const Color& color)
        {
            buffer[offset] = color;
        }
    };
}
```

5.5 Component.hpp

```
/*
 *  This file is part of OpenGL-FinalProject
 *
 *  Developed by Andrés Ragot - github.com/andresragot
 *
 *  MIT License
 *
 *  Copyright (c) 2024 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 "MySystem.hpp"
#include "Mesh.hpp"
#include <glm.hpp>
#include <gtc/matrix_transform.hpp>         // translate, rotate, scale, perspective
#include <gtc/type_ptr.hpp>                 // value_ptr, quat
#include "Camera.hpp"
#include "Shader_Program.hpp"

using glm::vec3;
using glm::mat4;

namespace Ragot
{
    class Component
    {
    public:
        virtual ~Component() = default;

        std::shared_ptr<Entity> get_entity() const { return entity.lock(); }

        void set_entity(std::shared_ptr<Entity> ent) { entity = ent; }
        bool get_has_task() const { return has_task; }

    private:
        std::weak_ptr<Entity> entity;

    protected:
        bool has_task = false;
    };

    class Transform_Component : public Component
    {
    public:
        Transform_Component() : position(0.f), rotation(0.f), scale(1.0f), parent(nullptr) {}

        mat4 get_transform_matrix()
        {
            mat4 transform_matrix(1);
            transform_matrix  = glm::translate(transform_matrix, position);
            transform_matrix  = glm::scale(transform_matrix, scale);

            glm::quat quaternion_rotation = glm::quat(glm::radians(rotation));
            transform_matrix *= glm::mat4_cast(quaternion_rotation);

            if (parent)
                transform_matrix = parent->get_transform_matrix() * transform_matrix;

            return transform_matrix;
        }

        void set_position(const vec3 &pos) { position = pos; }

        vec3 get_position() const { return position; }

        void set_rotation(const vec3 &rot) { rotation = rot; }

        vec3 get_rotation() const { return rotation; }

        void set_scale(const vec3 &scal) { scale = scal; }

        vec3 get_scale() const { return scale; }

        void set_parent(Transform_Component *par) { parent = par; }

        Transform_Component* get_parent() const { return parent; }

    private:
        vec3 position;
        vec3 rotation;
        vec3 scale;
        Transform_Component* parent;
    };

    class Model_Component : public Component
    {
    public:
        Model_Component() = delete;

        Model_Component(const string &model_file_path, const string &texture_file_path);

        Critical_Task render_task;
        Light_Task update_task;

        const GLuint get_shader_program_id() const { return material.get_shader_program_id(); }

        void set_transparency(bool trans) { is_transparent = trans; }

    private:
        Mesh mesh;
        Material material;
        bool is_transparent = false;
        float vertical_position;
        float vertical_speed = 0.0000005f;

        float orbit_angle;
        float orbit_speed = 0.00005f;

        static const string vertex_shader_code;
        static const string fragment_shader_code;

        GLint model_view_matrix_id;
        GLint projection_matrix_id;
        GLint normal_matrix_id;
        GLint view_pos_id;

    private:
        void configure_material();
```

```

        void render();

         void update();
    };
}
```

5.6 Entity.hpp

```
/*
 *  This file is part of OpenGL-FinalProject
 *
 *  Developed by Andrés Ragot - github.com/andresragot
 *
 *  MIT License
 *
 *  Copyright (c) 2024 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 "MySystem.hpp"
#include "Component.hpp"

namespace Ragot
{
    class Entity : public std::enable_shared_from_this<Entity>
    {
        Scene* scene;

    public:
        Transform_Component transform;

    private:
        map<string, shared_ptr<Component» components;
        vector<shared_ptr<Entity» children;

    public:
        void set_scene(Scene* scene) { this->scene = scene; }

        const Scene* get_scene() { return scene; }

        const Scene* get_scene() const { return scene; }

        void add_component(shared_ptr<Component> component, const string& name);

        void remove_component(const string& name);

        void add_child(shared_ptr<Entity> child)
        {
            child->set_transform_parent(&transform);
            children.push_back(child);
        }

        void remove_child(shared_ptr<Entity> child)
        {
            child->set_transform_parent(nullptr);
            children.erase(remove(children.begin(), children.end(), child), children.end());
        }

        void set_transform_parent(Transform_Component* parent)
        {
            transform.set_parent(parent);
        }

        const map<string, shared_ptr<Component»& get_components() const { return components; }
    };
}
```

5.7 Mesh.hpp

```
/*
 *  This file is part of OpenGL-FinalProject
 *
 *  Developed by Andrés Ragot - github.com/andresragot
 *
 *  MIT License
 *
 *  Copyright (c) 2024 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 <glad/glad.h>
#include <glm.hpp>
#include <vector>
#include <string>
#include "Camera.hpp"
#include "Shader_Program.hpp"
#include "Color.hpp"
#include "Color_Buffer.hpp"

namespace Ragot
{
    using std::vector;

    class Mesh
    {
    protected:
        enum
        {
            COORDINATES_VBO,
            NORMALS_VBO,
            TEXTURE_UVS_VBO,
            INDICES_EBO,
            VBO_COUNT
        };

        vector<glm::vec3> coordinates;
        vector<glm::vec3> normals;
        vector<glm::vec2> texture_coords;
        vector<GLuint> indices;

    private:
        GLuint vbo_ids[VBO_COUNT];
        GLuint vao_id;

        GLsizei number_of_indices;

        float angle;

        void load_mesh(const std::string& mesh_file_path);

    private:
        Mesh(const Mesh&) = delete;
        Mesh& operator=(const Mesh&) = delete;

    public:
        Mesh() = default;

        Mesh(const std::string& mesh_file_path);

        ~Mesh()
        {
            glDeleteVertexArrays(1, &vao_id);
            glDeleteBuffers(VBO_COUNT, vbo_ids);
        }

    public:
        const vector<glm::vec3>& get_coordinates() const { return coordinates; }

        const vector<glm::vec3>& get_normals() const { return normals; }

        const vector<glm::vec2>& get_textures_uv() const { return texture_coords; }

        const vector<GLuint>& get_indices() const { return indices; }

        const GLuint get_vao_id() const { return vao_id; }

        const GLsizei get_number_of_indices() const { return number_of_indices; }
    };

    template <typename COLOR_FORMAT>
    class Texture2D
    {
    protected:
        typedef Color_Buffer<COLOR_FORMAT> Color_Buffer;

        GLuint texture_id;
        bool texture_is_loaded;

    private:
        bool is_uint8 = false;

    public:
        Texture2D(const string& texture_base_path);

        ~Texture2D();

    private:
        Texture2D(const Texture2D&) = delete;
        Texture2D& operator=(const Texture2D&) = delete;

    protected:
        Texture2D() : texture_id(0), texture_is_loaded(false) {}

    public:
        bool is_ok() const
        {
            return texture_is_loaded;
        }

        virtual bool bind() const
        {
            return texture_is_loaded ? glBindTexture(GL_TEXTURE_2D, texture_id), true : false;
        }

    protected:
        GLint create_texture_2d(const string& texture_path);

        unique_ptr<Color_Buffer> load_image(const string& texture_path);
    };

    class Texture_Cube : public Texture2D<Rgba8888>
    {
    public:
        Texture_Cube(const string& texture_base_path);

        bool bind() const override
        {
            return texture_is_loaded ? glBindTexture(GL_TEXTURE_CUBE_MAP, texture_id), true : false;
        }
    };

    class Material
    {
    private:
        Shader_Program shader_program;
        Texture2D<Rgba8888> texture;
        glm::vec3 color;
        float shininess;

    public:
        Material() = delete;

        Material(const vector<string>& source_code_vertex, const vector<string>& source_code_fragment,
    const string& texture_base_path);
```

```
        ~Material() = default;

        void use_shader_program() { shader_program.use(); }

        GLint get_shader_program_uniform_location(const string& uniform) { return
shader_program.get_uniform_location(uniform); }

        GLuint get_shader_program_id() const { return shader_program.get_id(); }

        const bool bind_texture() const { return texture.bind(); }

        const glm::vec3 get_color() { return color; }

        const float get_shininess() { return shininess; }
    };
}
```

5.8 MyKernel.hpp

```
/*
 *  This file is part of OpenGL-FinalProject
 *
 *  Developed by Andrés Ragot - github.com/andresragot
 *
 *  MIT License
 *
 *  Copyright (c) 2024 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 "Task.hpp"
#include <thread>
#include <vector>
#include <memory>
#include <mutex>


namespace Ragot
{
    using std::vector;

    class Kernel
    {
        vector<std::shared_ptr<Task» tasks;
        vector<std::shared_ptr<Critical_Task» render_tasks;

        std::mutex tasks_mutex;

        std::atomic<bool> exit;
        std::atomic<bool> is_running;

        std::condition_variable cv;

    public:
        void add(std::shared_ptr<Task> new_task);

        void run();

        void stop()
        {
            tasks.front()->stop_execution();
            exit = true;
            cv.notify_all();
        }

        void execute_critical();
    };
}
```

5.9 MySystem.hpp

```
/*
 *  This file is part of OpenGL-FinalProject
 *
 *  Developed by Andrés Ragot - github.com/andresragot
 *
 *  MIT License
 *
 *  Copyright (c) 2024 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 "Window.hpp"
#include "Task.hpp"
#include "MyKernel.hpp"
#include "Camera.hpp"
#include "Ambient.hpp"
#include "Postprocess.hpp"
#include <string>
#include <memory>
#include <map>
#include <vector>

#include <mutex>
#include <condition_variable>
#include <queue>

namespace Ragot
{
    using namespace std;

    // Declaración adelantada de Entity
    class Entity;

    class Scene
    {
    private:
        shared_ptr<Camera> camera = make_shared<Camera>();
        map<string, shared_ptr<Entity» entities;
        Frame_Buffer framebuffer;
        Skybox skybox;
        Terrain terrain;
        vector<shared_ptr<Light» lights;

        mutex scene_mutex;

        int width;
        int height;

        float angle_around_x;
        float angle_around_y;
        float angle_delta_x;
        float angle_delta_y;

        float camera_speed = 0.025f;

        bool pointer_pressed;
```

```
            int last_pointer_x;
            int last_pointer_y;
            bool turbo;

            float camera_turbo_speed = 2.f;

            glm::vec3 camera_translation;

        public:
            void resize(int width, int height);

            void on_drag(int pointer_x, int pointer_y);

            void on_click(int pointer_x, int pointer_y, bool down);

            void on_translation(glm::vec3 translation);

            void on_shift_pressed(bool down);

        public:
            void update();

            void render();

            void postproccess();

        public:
            Scene();

        public:
            void add_entities(shared_ptr<Entity> entity, const string& name);

            void remove_entities(const string& name);

            shared_ptr<Entity> get_entity(const string& name) const;

            shared_ptr<Camera> get_camera() const { return camera; }

        private:
            void set_lights(GLuint shader_program_id);
        };

        class System
        {
            Critical_Task buffer_swap;
            Critical_Task handle_events;
            Critical_Task scene_render;
            Light_Task scene_update;
            Light_Task process_events;

            queue<SDL_Event> eventQueue;
            mutex queueMutex;
            condition_variable queueCondition;

            Window window;
            Scene scene;

            Kernel kernel;

        private:
            void input();

            void initialize();

            void sdl_events();

        public:
            System(const string& Window_Name, const int width, const int height);

            System();

            ~System() { SDL_Quit(); }

        public:
            void add_entities(shared_ptr<Entity> entity, const string& name);

            void run()
            {
                kernel.run();
            }

            void stop()
            {
                kernel.stop();
            }
        };
    }
```

5.10 Postprocess.hpp

```
/*
 *  This file is part of OpenGL-FinalProject
 *
 *  Developed by Andrés Ragot - github.com/andresragot
 *
 *  MIT License
 *
 *  Copyright (c) 2024 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 "Shader_Program.hpp"

namespace Ragot
{
    class Frame_Buffer
    {
    private:
        enum
        {
            COORDINATES_VBO,
            UV_COORDINATES_VBO,
            VBO_COUNT
        };

        static float vertices[];
        static float uv_coordenates[];

        static const string vertex_code_shader;
        static const string fragment_code_shader;

        GLuint frame_buffer_id;
        GLuint texture_id;
        GLuint depthbuffer_id;
        GLint current_time_id;

        Shader_Program shader_program;

        GLuint vbo_id[VBO_COUNT];
        GLuint vao_id;

    public:
        Frame_Buffer(unsigned width, unsigned height);

        Frame_Buffer() = delete;

        ~Frame_Buffer();

        void bind_frame_buffer() const { glBindFramebuffer(GL_FRAMEBUFFER, frame_buffer_id); }

        void unbind_frame_buffer() const { glBindFramebuffer(GL_FRAMEBUFFER, 0); }

        void bind_texture() const { glBindTexture(GL_TEXTURE_2D, texture_id); }

        void unbind_texture() const { glBindTexture(GL_TEXTURE_2D, 0); }

        void render();
    };
}
```

5.11 Shader_Program.hpp

```
/*
```

```
 *  This file is part of OpenGL-FinalProject
 *
 *  Developed by Andrés Ragot - github.com/andresragot
 *
 *  MIT License
 *
 *  Copyright (c) 2024 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();
    };
}
```

5.12 Task.hpp

```
/*
 *  This file is part of OpenGL-FinalProject
 *
 *  Developed by Andrés Ragot - github.com/andresragot
 *
 *  MIT License
 *
 *  Copyright (c) 2024 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 <mutex>
#include <condition_variable>
#include <atomic>
#include <functional>
```

```

namespace Ragot
{
    using std::condition_variable;
    using std::mutex;
    using std::atomic;
    using std::function;

    class Task
    {
        static condition_variable cv;
        static mutex mtx;
        static atomic<bool> is_stop;
        static atomic<bool> finish_execution;

    public:
        Task(function<void()> task_func) : task_func(task_func) {}

        virtual ~Task() = default;

        virtual void execute() = 0;

        void stop_execution()
        {
            std::lock_guard<mutex> lock(mtx);
            finish_execution = true;
            cv.notify_all();
        }

        void stop()
        {
            std::lock_guard<mutex> lock(mtx);
            is_stop = true;
            cv.notify_all();
        }

        void resume()
        {
            std::lock_guard<mutex> lock(mtx);
            is_stop = false;
            cv.notify_all();
        }

    protected:
        bool shouldStop()
        {
            return is_stop.load();
        }

        bool shouldFinish()
        {
            return finish_execution.load();
        }

        void wait_for_resume()
        {
            std::unique_lock<mutex> lock(mtx);
            cv.wait(lock, [this] {return !shouldStop() || shouldFinish(); });
        }

    protected:
        function<void()> task_func;
    };

    class Light_Task : public Task
    {
    public:
        Light_Task(function<void()> task_func) : Task(task_func) {}

        void execute() override;
    };

    class Critical_Task : public Task
    {
    public:
        Critical_Task(function<void()> task_func) : Task(task_func) {}

        void execute() override;
    };

    class Once_Task : public Task
    {
    public:
        Once_Task(function<void()> task_func) : Task(task_func) {}

        void execute() override;
    };
}

```

5.13 Window.hpp

```
/*
 *  This file is part of OpenGL-FinalProject
 *
 *  Developed by Andrés Ragot - github.com/andresragot
 *
 *  MIT License
 *
 *  Copyright (c) 2024 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; }
    };
}
```