raytracer

0.1.0

Generated by Doxygen 1.10.0

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

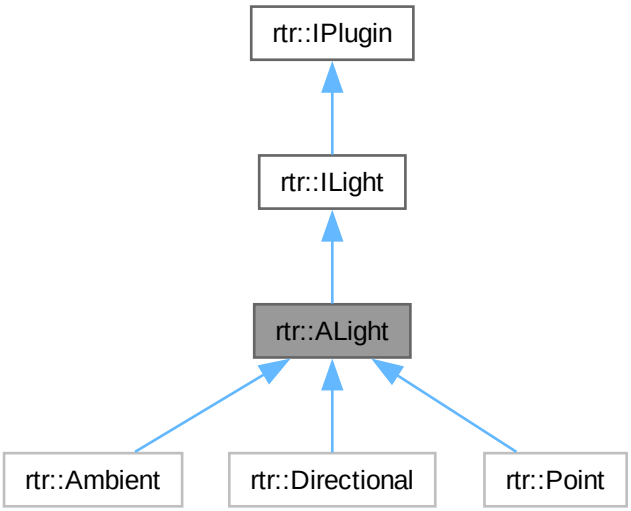
Class Documentation

4.1 rtr::ALight Class Reference

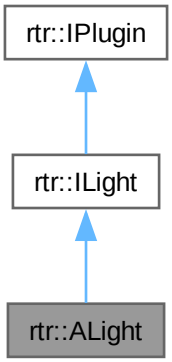
An abstract class for lights.

`#include <ALight.hpp>`

Inheritance diagram for rtr::ALight:



Collaboration diagram for rtr::ALight:



Public Member Functions

- void setType (const LightType &type) override
  Sets the type of the light (directional, ambient or point).
- void setIntensity (const float &intensity) override
  Sets the intensity of the light, based on the configuration file.
- const LightType & getType () const override
  Gets the type of the light based on the configuration file.
- Vector & getPosition () override
  Gets the position of the light based on the configuration file.
- Vector & getDirection () override
  Gets the direction of the light based on the configuration file.
- Color & getColor () override
  Gets the color of the light based on the configuration file.
- float & getIntensity () override
  Gets the intensity of the light based on the configuration file.

Public Member Functions inherited from rtr::ILight

- virtual Color LightColor (const Vector &normal, const Color &col)=0
  Creates light effects based on the light type.

Public Member Functions inherited from rtr::IPlugin

- virtual std::string getPluginName () const =0
  Gets the name of the plugin.

4.1.1 Detailed Description

An abstract class for lights.

4.1.2 Member Function Documentation

4.1.2.1 getColor()

```
Color & rtr::ALight::getColor ( )  [inline], [override], [virtual]
```

Gets the color of the light based on the configuration file.

Implements rtr::ILight.

4.1.2.2 getDirection()

```
Vector & rtr::ALight::getDirection ( )  [inline], [override], [virtual]
```

Gets the direction of the light based on the configuration file.

Implements rtr::ILight.

Reimplemented in rtr::Ambient, and rtr::Point.

4.1.2.3 getIntensity()

```
float & rtr::ALight::getIntensity ( )  [inline], [override], [virtual]
```

Gets the intensity of the light based on the configuration file.

Implements rtr::ILight.

4.1.2.4 getPosition()

```
Vector & rtr::ALight::getPosition ( )  [inline], [override], [virtual]
```

Gets the position of the light based on the configuration file.

Implements rtr::ILight.

4.1.2.5 getType()

```
const LightType & rtr::ALight::getType ( ) const  [inline], [override], [virtual]
```

Gets the type of the light based on the configuration file.

Implements rtr::ILight.

4.1.2.6 setIntensity()

```
void rtr::ALight::setIntensity (
            const float & intensity )  [inline], [override], [virtual]
```

Sets the intensity of the light, based on the configuration file.

Implements rtr::ILight.

4.1.2.7 setType()

```
void rtr::ALight::setType (
            const LightType & type )  [inline], [override], [virtual]
```

Sets the type of the light (directional, ambient or point).

Parameters

type	The type of the light (defined in the enum class LighType).

Implements rtr::ILight.

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

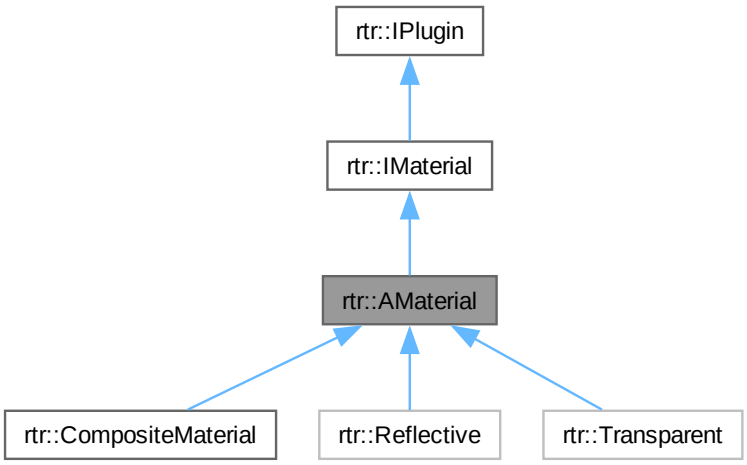
- App/include/RayTracer/Abstraction/ALight.hpp

4.2 rtr::AMaterial Class Reference

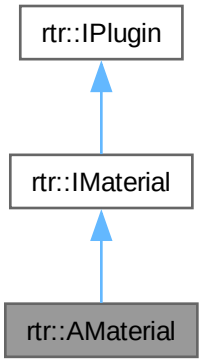
An abstract class for materials, based on the interface IMaterials.

`#include <AMaterial.hpp>`

Inheritance diagram for rtr::AMaterial:



Collaboration diagram for rtr::AMaterial:



Public Member Functions

- void setType (const MaterialType &type) override
  Sets the type of the material.
- void setReflectivity (const float &reflectivity) override
  Sets the color of the material.
- void setTransparency (const float &transparency) override
  Sets the transparency of the material.
- const MaterialType & getType () const override
  Gets the type of the material based on the configuration file.
- Color & getColor () override
  Gets the color of the material based on the configuration file.
- const float & getReflectivity () const override
  Gets the reflectiveness of the material based on the configuration file.
- const float & getTransparency () const override
  Gets the transparency of the material based on the configuration file.

Public Member Functions inherited from rtr::IMaterial

- virtual void applyMaterial (Color ∗color)=0
  Applies the material to the shape (transparency and reflectiveness).

Public Member Functions inherited from rtr::IPlugin

- virtual std::string getPluginName () const =0
  Gets the name of the plugin.

4.2.1 Detailed Description

An abstract class for materials, based on the interface IMaterials.

4.2.2 Member Function Documentation

4.2.2.1 getColor()

```
Color & rtr::AMaterial::getColor ( )  [inline], [override], [virtual]
```

Gets the color of the material based on the configuration file.

Implements rtr::IMaterial.

4.2.2.2 getReflectivity()

```
const float & rtr::AMaterial::getReflectivity ( ) const  [inline], [override], [virtual]
```

Gets the reflectiveness of the material based on the configuration file.

Implements rtr::IMaterial.

4.2.2.3 getTransparency()

```
const float & rtr::AMaterial::getTransparency ( ) const  [inline], [override], [virtual]
```

Gets the transparency of the material based on the configuration file.

Implements rtr::IMaterial.

4.2.2.4 getType()

```
const MaterialType & rtr::AMaterial::getType ( ) const  [inline], [override], [virtual]
```

Gets the type of the material based on the configuration file.

Implements rtr::IMaterial.

4.2.2.5 setReflectivity()

```
void rtr::AMaterial::setReflectivity (
            const float & reflectivity )  [inline], [override], [virtual]
```

Sets the color of the material.

Parameters

reflectivity	The reflectiveness of the material, based on the configuration file.

Implements rtr::IMaterial.

4.2.2.6 setTransparency()

```
void rtr::AMaterial::setTransparency (
            const float & transparency )  [inline], [override], [virtual]
```

Sets the transparency of the material.

Parameters

transparency	The transparency of the material, based on the configuration file.

Implements rtr::IMaterial.

4.2.2.7 setType()

```
void rtr::AMaterial::setType (
            const MaterialType & type )  [inline], [override], [virtual]
```

Sets the type of the material.

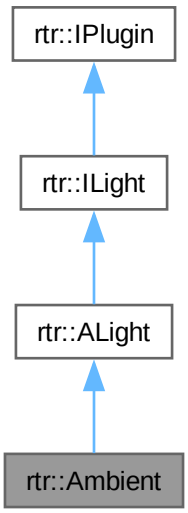
Implements rtr::IMaterial.

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

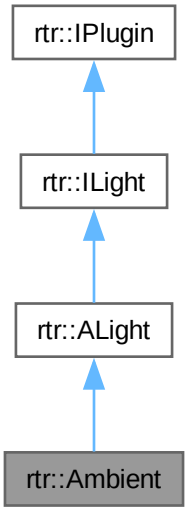
- App/include/RayTracer/Abstraction/AMaterial.hpp

4.3 rtr::Ambient Class Reference

Inheritance diagram for rtr::Ambient:



Collaboration diagram for rtr::Ambient:



Public Member Functions

- Color LightColor (const Vector &normal, const Color &col) override

  Creates light effects based on the light type.
- std::string getPluginName () const override

  Gets the name of the plugin.
- Vector & getDirection () override

  Gets the direction of the light based on the configuration file.

Public Member Functions inherited from rtr::ALight

- void setType (const LightType &type) override

  Sets the type of the light (directional, ambient or point).
- void setIntensity (const float &intensity) override

  Sets the intensity of the light, based on the configuration file.
- const LightType & getType () const override

  Gets the type of the light based on the configuration file.
- Vector & getPosition () override

  Gets the position of the light based on the configuration file.
- Color & getColor () override

  Gets the color of the light based on the configuration file.
- float & getIntensity () override

  Gets the intensity of the light based on the configuration file.

4.3.1 Member Function Documentation

4.3.1.1 getDirection()

```
Vector & rtr::Ambient::getDirection ( )  [inline], [override], [virtual]
```

Gets the direction of the light based on the configuration file.

Reimplemented from rtr::ALight.

4.3.1.2 getPluginName()

```
std::string rtr::Ambient::getPluginName ( ) const  [inline], [override], [virtual]
```

Gets the name of the plugin.

Returns

A string of the plugin's name, defined as const expressions.

Implements rtr::IPlugin.

4.3.1.3 LightColor()

```
Color rtr::Ambient::LightColor (
            const Vector & normal,
            const Color & col )  [override], [virtual]
```

Creates light effects based on the light type.

Parameters

normal	The normal of the shape.
col	The current color of the shape.

Returns

The new color of the shape with the light effects.

Implements rtr::ILight.

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

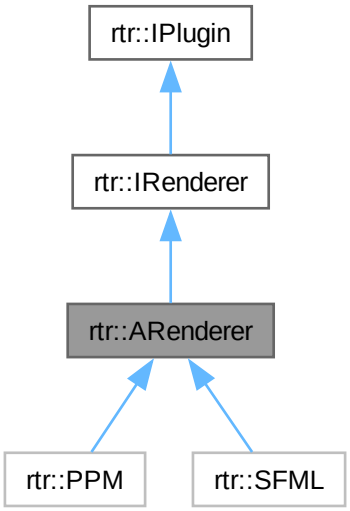
- App/plugins/Light/Ambient/include/RayTracer/Ambient.hpp

4.4 rtr::ARenderer Class Reference

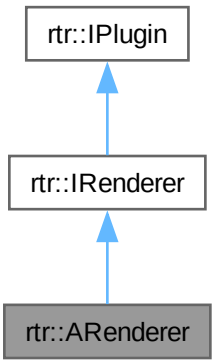
An abstract class for renderers, based on the interface IRenderer.

`#include <ARenderer.hpp>`

Inheritance diagram for rtr::ARenderer:



Collaboration diagram for rtr::ARenderer:



Public Member Functions

- void setType (const RendererType &rendererType) override

  Sets the type of the renderer.
- void setName (const std::string &name) override

  Sets the name of the renderer.
- void setPixels (const std::vector< std::vector< rtr::Color > > &pixels) override

  Sets the pixels of the renderer.
- const RendererType & getType () const override

  Gets the type of the renderer based on the configuration file.
- Resolution & getResolution () override

  Gets the resolution of the renderer based on the configuration file.
- Color & getBackgroundColor () override

  Gets the background color of the renderer based on the configuration file.
- const std::string & getName () const override

  Gets the name of the renderer based on the configuration file.
- std::vector< std::vector< rtr::Color > > & getPixels () override

  Gets the pixels of the renderer based on the configuration file.

Public Member Functions inherited from rtr::IRenderer

- virtual void render (const std::vector< std::unique_ptr< AShape > > &shapes, const std::vector< std::unique_ptr< ALight > > &lights, const Camera &camera)=0

  Renders the scene based on the shapes, lights and camera.

Public Member Functions inherited from rtr::IPlugin

- virtual std::string getPluginName () const =0

  Gets the name of the plugin.

4.4.1 Detailed Description

An abstract class for renderers, based on the interface IRenderer.

4.4.2 Member Function Documentation

4.4.2.1 getBackgroundColor()

```
Color & rtr::ARenderer::getBackgroundColor ( )  [inline], [override], [virtual]
```

Gets the background color of the renderer based on the configuration file.

Returns

The background color of the renderer.

Implements rtr::IRenderer.

4.4.2.2 getName()

```
const std::string & rtr::ARenderer::getName ( ) const  [inline], [override], [virtual]
```

Gets the name of the renderer based on the configuration file.

Returns

A string of the renderer's name.

Implements rtr::IRenderer.

4.4.2.3 getPixels()

```
std::vector< std::vector< rtr::Color > > & rtr::ARenderer::getPixels ( )  [inline], [override],
[virtual]
```

Gets the pixels of the renderer based on the configuration file.

Returns

Each pixels of the image.

Implements rtr::IRenderer.

4.4.2.4 getResolution()

```
Resolution & rtr::ARenderer::getResolution ( )  [inline], [override], [virtual]
```

Gets the resolution of the renderer based on the configuration file.

Returns

The resolution of the renderer using the Resolution class.

Implements rtr::IRenderer.

4.4.2.5 getType()

```
const RendererType & rtr::ARenderer::getType ( ) const  [inline], [override], [virtual]
```

Gets the type of the renderer based on the configuration file.

Returns

The type of the renderer, using the enum class RendererType.

Implements rtr::IRenderer.

4.4.2.6 setName()

```
void rtr::ARenderer::setName (
            const std::string & name )  [inline], [override], [virtual]
```

Sets the name of the renderer.

Parameters

name	The name of the renderer, based on the configuration file.

Implements rtr::IRenderer.

4.4.2.7 setPixels()

```
void rtr::ARenderer::setPixels (
            const std::vector< std::vector< rtr::Color > > & pixels )  [inline], [override],
[virtual]
```

Sets the pixels of the renderer.

Parameters

pixels	The pixels of the renderer.

Implements rtr::IRenderer.

4.4.2.8 setType()

```
void rtr::ARenderer::setType (
            const RendererType & rendererType )  [inline], [override], [virtual]
```

Sets the type of the renderer.

Parameters

rendererType	The type of the renderer (defined in the RendererType enum class).

Implements rtr::IRenderer.

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

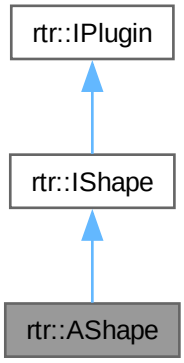
- App/include/RayTracer/Abstraction/ARenderer.hpp

4.5 rtr::AShape Class Reference

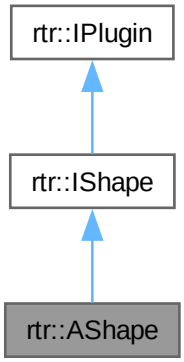
An abstract class for shapes, based on the interface IShape.

```
#include <AShape.hpp>
```

Inheritance diagram for rtr::AShape:



Collaboration diagram for rtr::AShape:



Public Member Functions

- void setType (const ShapeType &type) override

  Sets the type of the shape (sphere, plane, cone...).
- void setRadius (const double &radius) override

  Sets the radius of the shape.
- void setHeight (const double &height) override

  Sets the height of the shape.
- void setMaterial (std::unique_ptr< AMaterial > material) override

  Sets the material of the shape.
- const ShapeType & getType () const override

Gets the type of the shape.

- AMaterial & getMaterial () override

  Gets the material of the shape.
- Vector & getPosition () override

  Gets the position of the shape.
- Vector & getNormal () override

  Gets the normal of the shape.
- Vector & getRotation () override

  Gets the rotation of the shape, used to create the shape.
- const double & getRadius () const override

  Gets the radius of the shape.
- const double & getHeight () const override

  Gets the height of the shape.
- Vector getDistance (const Vector &point) override

  Gets the distance between the shape and a point.

Public Member Functions inherited from rtr::IShape

- virtual bool hits (std::pair< Vector, Vector > ray, RayHit &hit)=0

  Checks if the ray hits the shape, used to render the output file.

Public Member Functions inherited from rtr::IPlugin

- virtual std::string getPluginName () const =0

  Gets the name of the plugin.

4.5.1 Detailed Description

An abstract class for shapes, based on the interface IShape.

4.5.2 Member Function Documentation

4.5.2.1 getDistance()

```
Vector rtr::AShape::getDistance (
            const Vector & point )  [inline], [override], [virtual]
```

Gets the distance between the shape and a point.

Parameters

point	The point to check the distance with.

Returns

The distance between the shape and the point.

Implements rtr::IShape.

4.5.2.2 getHeight()

```
const double & rtr::AShape::getHeight ( ) const  [inline], [override], [virtual]
```

Gets the height of the shape.

Returns

The height of the shape as a double.

Implements rtr::IShape.

4.5.2.3 getMaterial()

```
AMaterial & rtr::AShape::getMaterial ( )  [inline], [override], [virtual]
```

Gets the material of the shape.

Returns

The material of the shape, using the AMaterial class.

Implements rtr::IShape.

4.5.2.4 getNormal()

```
Vector & rtr::AShape::getNormal ( )  [inline], [override], [virtual]
```

Gets the normal of the shape.

Returns

The normal of the shape, using the Vector class.

Implements rtr::IShape.

4.5.2.5 getPosition()

```
Vector & rtr::AShape::getPosition ( )  [inline], [override], [virtual]
```

Gets the position of the shape.

Returns

The position of the shape, using the Vector class.

Implements rtr::IShape.

4.5.2.6 getRadius()

```
const double & rtr::AShape::getRadius ( ) const  [inline], [override], [virtual]
```

Gets the radius of the shape.

Returns

The radius of the shape as a double, used to check the size of the shape.

Implements rtr::IShape.

4.5.2.7 getRotation()

```
Vector & rtr::AShape::getRotation ( )  [inline], [override], [virtual]
```

Gets the rotation of the shape, used to create the shape.

Returns

The rotation of the shape, using the Vector class.

Implements rtr::IShape.

4.5.2.8 getType()

```
const ShapeType & rtr::AShape::getType ( ) const  [inline], [override], [virtual]
```

Gets the type of the shape.

Returns

The type of the shape, using the ShapeType enum class.

Implements rtr::IShape.

4.5.2.9 setHeight()

```
void rtr::AShape::setHeight (
            const double & height )  [inline], [override], [virtual]
```

Sets the height of the shape.

Parameters

height	The height of the shape.

Implements rtr::IShape.

4.5.2.10 setMaterial()

```
void rtr::AShape::setMaterial (
            std::unique_ptr< AMaterial > material )  [inline], [override], [virtual]
```

Sets the material of the shape.

Parameters

material	The material of the shape (transparency, reflectivity...).

Implements rtr::IShape.

4.5.2.11 setRadius()

```
void rtr::AShape::setRadius (
            const double & radius )  [inline], [override], [virtual]
```

Sets the radius of the shape.

Parameters

radius	The radius of the shape.

Implements rtr::IShape.

4.5.2.12 setType()

```
void rtr::AShape::setType (
            const ShapeType & type )  [inline], [override], [virtual]
```

Sets the type of the shape (sphere, plane, cone...).

Parameters

type	The type of the shape.

Implements rtr::IShape.

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

- App/include/RayTracer/Abstraction/AShape.hpp

4.6 rtr::Camera Class Reference

A class to handle the camera.

`#include <Camera.hpp>`

Public Member Functions

- **Camera** (uint16_t fov, const Vector &origin, const Vector &direction)
- void setFov (const uint16_t fov)

  Sets the camera's field of view.
- uint16_t getFov () const

  Gets the camera's field of view.
- const Vector & **getOrigin** () const

  Gets the camera's origin.
- const Vector & **getDirection** () const

  Gets the camera's direction.
- const Vector & **getUp** () const

  Gets the camera's up vector.
- std::pair< Vector, Vector > ray (const double u, const double v) const

  The camera's ray, used to check if the ray intersects with an object.

4.6.1 Detailed Description

A class to handle the camera.

4.6.2 Member Function Documentation

4.6.2.1 getFov()

```
uint16_t rtr::Camera::getFov ( ) const  [inline]
```

Gets the camera's field of view.

Returns

The field of view, as a uint16, or a unsigned short.

4.6.2.2 ray()

```
std::pair< Vector, Vector > rtr::Camera::ray (
            const double u,
            const double v ) const  [inline]
```

The camera's ray, used to check if the ray intersects with an object.

Parameters

u	defines the horizontal position of the ray.
v	defines the vertical position of the ray.

Returns

A pair of vectors that defines the origin of the ray and its normal.

4.6.2.3 setFov()

```
void rtr::Camera::setFov (
            const uint16_t fov )  [inline]
```

Sets the camera's field of view.

Parameters

fov	The field of view, as a uint16, or a unsigned short.

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

- App/include/RayTracer/Scene/Camera.hpp

4.7 rtr::Color Class Reference

Class representing RGB colors.

`#include <Color.hpp>`

Public Member Functions

- **Color** (const uint8_t r, const uint8_t g, const uint8_t b)
- **Color** (const color_t &color)
- void setColor (const uint8_t r, const uint8_t g, const uint8_t b)
  Sets the color components.
- void setColor (const color_t &color)
  Sets the color components.
- void **setR** (const uint8_t r)
- void **setG** (const uint8_t g)
- void **setB** (const uint8_t b)
- color_t **getValue** () const
- uint8_t **getR** () const
- uint8_t **getG** () const
- uint8_t **getB** () const
- Color operator+ (const Color &other) const
  Adds two colors.
- Color operator∗ (const double &scalar) const
  Multiplies a color by a scalar.
- Color operator∗ (const Color &other) const
  Multiplies two colors.
- Color operator+= (const Color &other)
  Adds a color to the current color.
- Color operator∗= (const double &scalar)
  Multiplies the current color by a scalar.

Static Public Member Functions

- static constexpr color_t **getRed** ()
- static constexpr color_t **getGreen** ()
- static constexpr color_t **getBlue** ()
- static constexpr color_t **getWhite** ()
- static constexpr color_t **getBlack** ()
- static constexpr color_t **getYellow** ()
- static constexpr color_t **getMagenta** ()
- static constexpr color_t **getCyan** ()
- static constexpr color_t **getGray** ()
- static constexpr color_t **getOrange** ()
- static constexpr color_t **getBrown** ()
- static constexpr color_t **getLightBlue** ()
- static constexpr color_t **getLightGreen** ()
- static constexpr color_t **getLightPink** ()
- static constexpr color_t **getLightYellow** ()
- static constexpr color_t **getLightGray** ()
- static constexpr color_t **getDarkGray** ()
- static constexpr color_t **getDarkRed** ()
- static constexpr color_t **getDarkGreen** ()
- static constexpr color_t **getDarkBlue** ()
- static constexpr color_t **getDarkYellow** ()

4.7.1 Detailed Description

Class representing RGB colors.

4.7.2 Member Function Documentation

4.7.2.1 operator∗() [1/2]

```
Color rtr::Color::operator* (
            const Color & other ) const  [inline]
```

Multiplies two colors.

Parameters

other	The other color to multiply.

Returns

The product of the two colors.

4.7.2.2 operator∗() [2/2]

```
Color rtr::Color::operator* (
            const double & scalar ) const  [inline]
```

Multiplies a color by a scalar.

Parameters

scalar	The scalar to multiply by.

Returns

The product of the color and the scalar.

4.7.2.3 operator*=()

```
Color rtr::Color::operator*= (
            const double & scalar )  [inline]
```

Multiplies the current color by a scalar.

Parameters

scalar	The scalar to multiply by.

Returns

A reference to the current color.

4.7.2.4 operator+()

```
Color rtr::Color::operator+ (
            const Color & other ) const  [inline]
```

Adds two colors.

Parameters

other	The other color to add.

Returns

The sum of the two colors.

4.7.2.5 operator+=()

```
Color rtr::Color::operator+= (
            const Color & other )  [inline]
```

Adds a color to the current color.

Parameters

other	The other color to add.

Returns

A reference to the current color.

4.7.2.6 setColor() [1/2]

```
void rtr::Color::setColor (
            const color_t & color )  [inline]
```

Sets the color components.

Parameters

color	An RGB color.

4.7.2.7 setColor() [2/2]

```
void rtr::Color::setColor (
            const uint8_t r,
            const uint8_t g,
            const uint8_t b )  [inline]
```

Sets the color components.

Parameters

r	Red color component.
g	Green color component.
b	Blue color component.

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

- App/include/RayTracer/Utils/Color.hpp

4.8 color_s Struct Reference

A struct representing an RGB color.

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

4.8.1 Detailed Description

A struct representing an RGB color.

Type alias for an RGB color component.

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

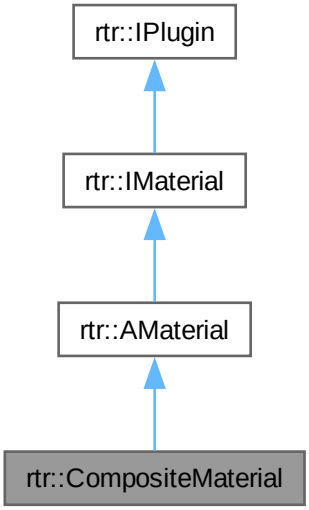
- App/include/RayTracer/Utils/Color.hpp

4.9 rtr::CompositeMaterial Class Reference

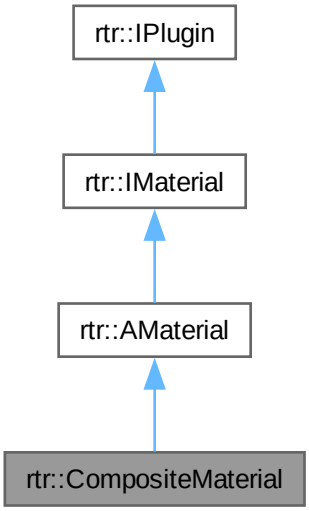
A class to create a composite material.

`#include <Material.hpp>`

Inheritance diagram for rtr::CompositeMaterial:



Collaboration diagram for rtr::CompositeMaterial:



Public Member Functions

- std::string getPluginName () const override

  Gets the name of the plugin.
- void addMaterial (std::unique_ptr< AMaterial > material)

  Adds a material to the composite material.
- void applyMaterial (Color *color) override

  Applies the material to the shape (transparency and reflectiveness).

Public Member Functions inherited from rtr::AMaterial

- void setType (const MaterialType &type) override

  Sets the type of the material.
- void setReflectivity (const float &reflectivity) override

  Sets the color of the material.
- void setTransparency (const float &transparency) override

  Sets the transparency of the material.
- const MaterialType & getType () const override

  Gets the type of the material based on the configuration file.
- Color & getColor () override

  Gets the color of the material based on the configuration file.
- const float & getReflectivity () const override

  Gets the reflectiveness of the material based on the configuration file.
- const float & getTransparency () const override

  Gets the transparency of the material based on the configuration file.

4.9.1 Detailed Description

A class to create a composite material.

4.9.2 Member Function Documentation

4.9.2.1 addMaterial()

```
void rtr::CompositeMaterial::addMaterial (
            std::unique_ptr< AMaterial > material )  [inline]
```

Adds a material to the composite material.

Parameters

material	The material to add.

4.9.2.2 applyMaterial()

```
void rtr::CompositeMaterial::applyMaterial (
            Color * color )  [inline], [override], [virtual]
```

Applies the material to the shape (transparency and reflectiveness).

Implements rtr::IMaterial.

4.9.2.3 getPluginName()

```
std::string rtr::CompositeMaterial::getPluginName ( ) const  [inline], [override], [virtual]
```

Gets the name of the plugin.

Returns

A string of the plugin's name, defined as const expressions.

Implements rtr::IPlugin.

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

- App/include/RayTracer/Composite/Material.hpp

4.10 rtr::Core Class Reference

A class representing the core functionality of the ray tracer.

```
#include <Core.hpp>
```

Classes

- class CoreException

  An exception class for core errors.

Static Public Member Functions

- static void runRayTracer (Scene &scene)

  Runs the ray tracer with a given Scene object.

4.10.1 Detailed Description

A class representing the core functionality of the ray tracer.

4.10.2 Member Function Documentation

4.10.2.1 runRayTracer()

```
static void rtr::Core::runRayTracer (
            Scene & scene )  [static]
```

Runs the ray tracer with a given Scene object.

Parameters

scene	The Scene object used for ray tracing.

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

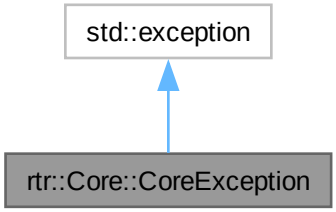
- App/include/RayTracer/Core.hpp

4.11 rtr::Core::CoreException Class Reference

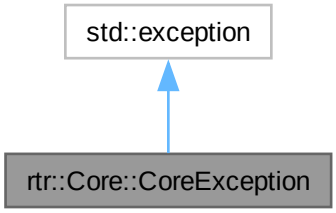
An exception class for core errors.

```
#include <Core.hpp>
```

Inheritance diagram for rtr::Core::CoreException:



Collaboration diagram for rtr::Core::CoreException:



Public Member Functions

- **CoreException** (std::string msg)
- **CoreException** (const CoreException &)=delete
- CoreException & **operator=** (const CoreException &)=delete
- **CoreException** (const CoreException &&)=delete
- CoreException & **operator=** (const CoreException &&)=delete
- const char * **what** () const noexcept override

  Returns the error message.

4.11.1 Detailed Description

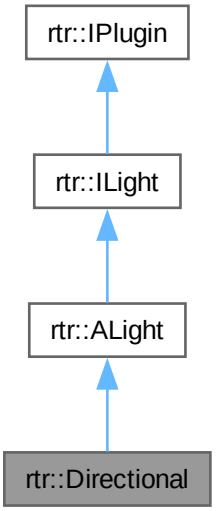
An exception class for core errors.

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

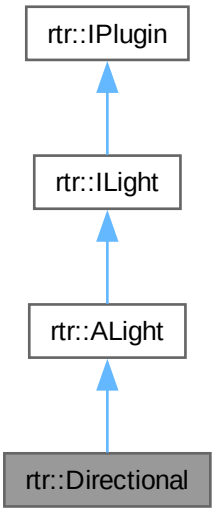
- App/include/RayTracer/Core.hpp

4.12 rtr::Directional Class Reference

Inheritance diagram for rtr::Directional:



Collaboration diagram for rtr::Directional:



Public Member Functions

- Color LightColor (const Vector &normal, const Color &col) override

  Creates light effects based on the light type.
- std::string getPluginName () const override

  Gets the name of the plugin.

Public Member Functions inherited from rtr::ALight

- void setType (const LightType &type) override

  Sets the type of the light (directional, ambient or point).
- void setIntensity (const float &intensity) override

  Sets the intensity of the light, based on the configuration file.
- const LightType & getType () const override

  Gets the type of the light based on the configuration file.
- Vector & getPosition () override

  Gets the position of the light based on the configuration file.
- Vector & getDirection () override

  Gets the direction of the light based on the configuration file.
- Color & getColor () override

  Gets the color of the light based on the configuration file.
- float & getIntensity () override

  Gets the intensity of the light based on the configuration file.

4.12.1 Member Function Documentation

4.12.1.1 getPluginName()

```
std::string rtr::Directional::getPluginName ( ) const  [inline], [override], [virtual]
```

Gets the name of the plugin.

Returns

A string of the plugin's name, defined as const expressions.

Implements rtr::IPlugin.

4.12.1.2 LightColor()

```
Color rtr::Directional::LightColor (
            const Vector & normal,
            const Color & col )  [override], [virtual]
```

Creates light effects based on the light type.

Parameters

normal	The normal of the shape.
col	The current color of the shape.

Returns

The new color of the shape with the light effects.

Implements rtr::ILight.

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

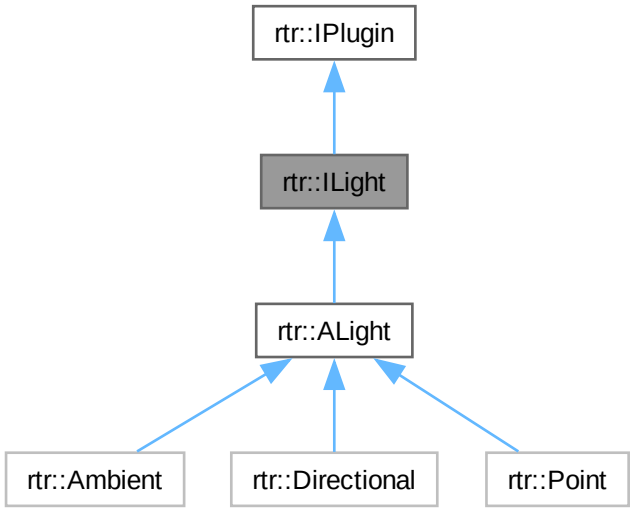
- App/plugins/Light/Directional/include/RayTracer/Directional.hpp

4.13 rtr::ILight Class Reference

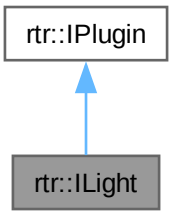
An interface for lights.

`#include <ILight.hpp>`

Inheritance diagram for rtr::ILight:



Collaboration diagram for rtr::ILight:



Public Member Functions

- virtual void setType (const LightType &type)=0

  Sets the type of the light (directional, ambient or point).
- virtual void setIntensity (const float &intensity)=0

  Sets the intensity of the light, based on the configuration file.
- virtual Color LightColor (const Vector &normal, const Color &col)=0

  Creates light effects based on the light type.
- virtual const LightType & getType () const =0

  Gets the type of the light based on the configuration file.
- virtual Vector & getPosition ()=0

  Gets the position of the light based on the configuration file.
- virtual Vector & getDirection ()=0

  Gets the direction of the light based on the configuration file.
- virtual Color & getColor ()=0

  Gets the color of the light based on the configuration file.
- virtual float & getIntensity ()=0

  Gets the intensity of the light based on the configuration file.

Public Member Functions inherited from rtr::IPlugin

- virtual std::string getPluginName () const =0

  Gets the name of the plugin.

4.13.1 Detailed Description

An interface for lights.

4.13.2 Member Function Documentation

4.13.2.1 getColor()

```
virtual Color & rtr::ILight::getColor ( )  [pure virtual]
```

Gets the color of the light based on the configuration file.

Implemented in rtr::ALight.

4.13.2.2 getDirection()

```
virtual Vector & rtr::ILight::getDirection ( )  [pure virtual]
```

Gets the direction of the light based on the configuration file.

Implemented in rtr::ALight, rtr::Ambient, and rtr::Point.

4.13.2.3 getIntensity()

```
virtual float & rtr::ILight::getIntensity ( )  [pure virtual]
```

Gets the intensity of the light based on the configuration file.

Implemented in rtr::ALight.

4.13.2.4 getPosition()

```
virtual Vector & rtr::ILight::getPosition ( )  [pure virtual]
```

Gets the position of the light based on the configuration file.

Implemented in rtr::ALight.

4.13.2.5 getType()

```
virtual const LightType & rtr::ILight::getType ( ) const  [pure virtual]
```

Gets the type of the light based on the configuration file.

Implemented in rtr::ALight.

4.13.2.6 LightColor()

```
virtual Color rtr::ILight::LightColor (
            const Vector & normal,
            const Color & col )  [pure virtual]
```

Creates light effects based on the light type.

Parameters

normal	The normal of the shape.
col	The current color of the shape.

Returns

The new color of the shape with the light effects.

Implemented in rtr::Ambient, rtr::Directional, and rtr::Point.

4.13.2.7 setIntensity()

```
virtual void rtr::ILight::setIntensity (
            const float & intensity )  [pure virtual]
```

Sets the intensity of the light, based on the configuration file.

Implemented in rtr::ALight.

4.13.2.8 setType()

```
virtual void rtr::ILight::setType (
            const LightType & type )  [pure virtual]
```

Sets the type of the light (directional, ambient or point).

Parameters

type	The type of the light (defined in the enum class LighType).

Implemented in rtr::ALight.

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

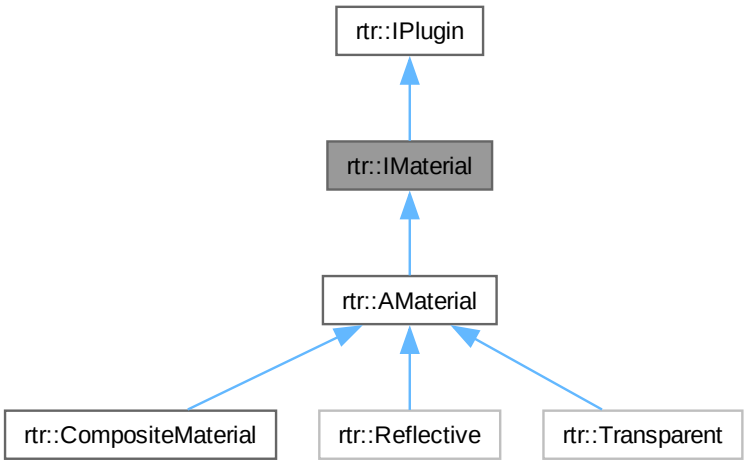
- App/include/RayTracer/Abstraction/ILight.hpp

4.14 rtr::IMaterial Class Reference

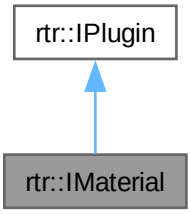
An interface for materials.

`#include <IMaterial.hpp>`

Inheritance diagram for rtr::IMaterial:



Collaboration diagram for rtr::IMaterial:



Public Member Functions

- virtual void applyMaterial (Color *color)=0
  Applies the material to the shape (transparency and reflectiveness).
- virtual void setType (const MaterialType &type)=0
  Sets the type of the material.
- virtual void setReflectivity (const float &reflectivity)=0
  Sets the color of the material.
- virtual void setTransparency (const float &transparency)=0
  Sets the transparency of the material.
- virtual const MaterialType & getType () const =0
  Gets the type of the material based on the configuration file.
- virtual Color & getColor ()=0
  Gets the color of the material based on the configuration file.
- virtual const float & getReflectivity () const =0
  Gets the reflectiveness of the material based on the configuration file.
- virtual const float & getTransparency () const =0
  Gets the transparency of the material based on the configuration file.

Public Member Functions inherited from rtr::IPlugin

- virtual std::string getPluginName () const =0
  Gets the name of the plugin.

4.14.1 Detailed Description

An interface for materials.

4.14.2 Member Function Documentation

4.14.2.1 applyMaterial()

```
virtual void rtr::IMaterial::applyMaterial (
            Color * color )  [pure virtual]
```

Applies the material to the shape (transparency and reflectiveness).

Implemented in rtr::CompositeMaterial, rtr::Reflective, and rtr::Transparent.

4.14.2.2 getColor()

```
virtual Color & rtr::IMaterial::getColor ( )  [pure virtual]
```

Gets the color of the material based on the configuration file.

Implemented in rtr::AMaterial.

4.14.2.3 getReflectivity()

```
virtual const float & rtr::IMaterial::getReflectivity ( ) const  [pure virtual]
```

Gets the reflectiveness of the material based on the configuration file.

Implemented in rtr::AMaterial.

4.14.2.4 getTransparency()

```
virtual const float & rtr::IMaterial::getTransparency ( ) const  [pure virtual]
```

Gets the transparency of the material based on the configuration file.

Implemented in rtr::AMaterial.

4.14.2.5 getType()

```
virtual const MaterialType & rtr::IMaterial::getType ( ) const  [pure virtual]
```

Gets the type of the material based on the configuration file.

Implemented in rtr::AMaterial.

4.14.2.6 setReflectivity()

```
virtual void rtr::IMaterial::setReflectivity (
            const float & reflectivity )  [pure virtual]
```

Sets the color of the material.

Parameters

reflectivity	The reflectiveness of the material, based on the configuration file.

Implemented in rtr::AMaterial.

4.14.2.7 setTransparency()

```
virtual void rtr::IMaterial::setTransparency (
```

```
            const float & transparency )  [pure virtual]
```

Sets the transparency of the material.

Parameters

transparency	The transparency of the material, based on the configuration file.

Implemented in rtr::AMaterial.

4.14.2.8 setType()

```
virtual void rtr::IMaterial::setType (
            const MaterialType & type )  [pure virtual]
```

Sets the type of the material.

Implemented in rtr::AMaterial.

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

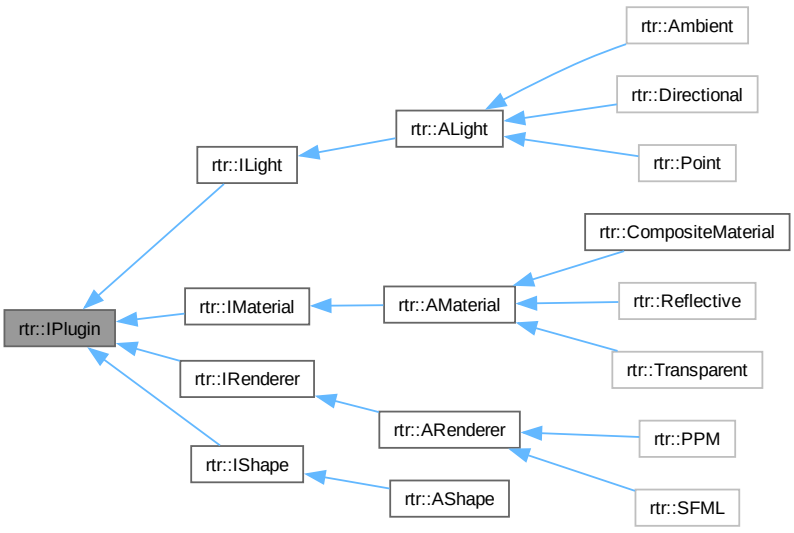
- App/include/RayTracer/Abstraction/IMaterial.hpp

4.15 rtr::IPlugin Class Reference

An interface for plugins.

`#include <IPlugin.hpp>`

Inheritance diagram for rtr::IPlugin:



Public Member Functions

- virtual std::string getPluginName () const =0

  Gets the name of the plugin.

4.15.1 Detailed Description

An interface for plugins.

4.15.2 Member Function Documentation

4.15.2.1 getPluginName()

```
virtual std::string rtr::IPlugin::getPluginName ( ) const  [pure virtual]
```

Gets the name of the plugin.

Returns

A string of the plugin's name, defined as const expressions.

Implemented in rtr::CompositeMaterial, rtr::Ambient, rtr::Directional, rtr::Point, rtr::Reflective, rtr::Transparent, rtr::PPM, and rtr::SFML.

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

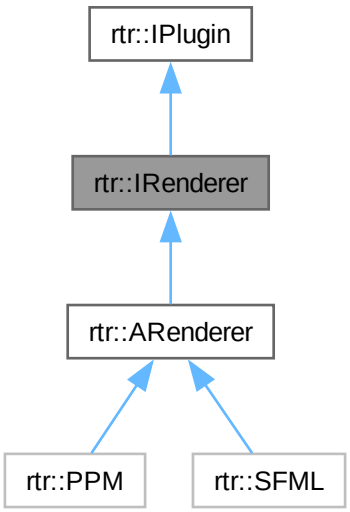
- App/include/RayTracer/Abstraction/IPlugin.hpp

4.16 rtr::IRenderer Class Reference

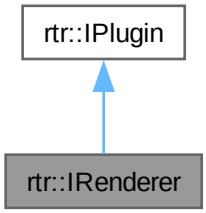
An interface for renderers.

`#include <IRenderer.hpp>`

Inheritance diagram for rtr::IRenderer:



Collaboration diagram for rtr::IRenderer:



Public Member Functions

- virtual void render (const std::vector< std::unique_ptr< AShape > > &shapes, const std::vector< std↩ ::unique_ptr< ALight > > &lights, const Camera &camera)=0

  Renders the scene based on the shapes, lights and camera.
- virtual void setType (const RendererType &rendererType)=0

  Sets the type of the renderer.
- virtual void setName (const std::string &name)=0

  Sets the name of the renderer.
- virtual void setPixels (const std::vector< std::vector< rtr::Color > > &pixels)=0

  Sets the pixels of the renderer.
- virtual const RendererType & getType () const =0

  Gets the type of the renderer based on the configuration file.
- virtual const std::string & getName () const =0

  Gets the name of the renderer based on the configuration file.
- virtual Resolution & getResolution ()=0

  Gets the resolution of the renderer based on the configuration file.
- virtual Color & getBackgroundColor ()=0

  Gets the background color of the renderer based on the configuration file.
- virtual std::vector< std::vector< rtr::Color > > & getPixels ()=0

  Gets the pixels of the renderer based on the configuration file.

Public Member Functions inherited from rtr::IPlugin

- virtual std::string getPluginName () const =0

  Gets the name of the plugin.

4.16.1 Detailed Description

An interface for renderers.

4.16.2 Member Function Documentation

4.16.2.1 getBackgroundColor()

```
virtual Color & rtr::IRenderer::getBackgroundColor ( )  [pure virtual]
```

Gets the background color of the renderer based on the configuration file.

Returns

The background color of the renderer.

Implemented in rtr::ARenderer.

4.16.2.2 getName()

```
virtual const std::string & rtr::IRenderer::getName ( ) const  [pure virtual]
```

Gets the name of the renderer based on the configuration file.

Returns

A string of the renderer's name.

Implemented in rtr::ARenderer.

4.16.2.3 getPixels()

```
virtual std::vector< std::vector< rtr::Color > > & rtr::IRenderer::getPixels ( )  [pure virtual]
```

Gets the pixels of the renderer based on the configuration file.

Returns

Each pixels of the image.

Implemented in rtr::ARenderer.

4.16.2.4 getResolution()

```
virtual Resolution & rtr::IRenderer::getResolution ( )  [pure virtual]
```

Gets the resolution of the renderer based on the configuration file.

Returns

The resolution of the renderer using the Resolution class.

Implemented in rtr::ARenderer.

4.16.2.5 getType()

```
virtual const RendererType & rtr::IRenderer::getType ( ) const  [pure virtual]
```

Gets the type of the renderer based on the configuration file.

Returns

The type of the renderer, using the enum class RendererType.

Implemented in rtr::ARenderer.

4.16.2.6 render()

```
virtual void rtr::IRenderer::render (
            const std::vector< std::unique_ptr< AShape > > & shapes,
            const std::vector< std::unique_ptr< ALight > > & lights,
            const Camera & camera )  [pure virtual]
```

Renders the scene based on the shapes, lights and camera.

Parameters

shapes	The shapes of the scene.
lights	The lights of the scene.
camera	The camera of the scene.

Implemented in rtr::PPM, and rtr::SFML.

4.16.2.7 setName()

```
virtual void rtr::IRenderer::setName (
            const std::string & name )  [pure virtual]
```

Sets the name of the renderer.

Parameters

name	The name of the renderer, based on the configuration file.

Implemented in rtr::ARenderer.

4.16.2.8 setPixels()

```
virtual void rtr::IRenderer::setPixels (
            const std::vector< std::vector< rtr::Color > > & pixels )  [pure virtual]
```

Sets the pixels of the renderer.

Parameters

pixels	The pixels of the renderer.

Implemented in rtr::ARenderer.

4.16.2.9 setType()

```
virtual void rtr::IRenderer::setType (
            const RendererType & rendererType )  [pure virtual]
```

Sets the type of the renderer.

Parameters

rendererType	The type of the renderer (defined in the RendererType enum class).

Implemented in rtr::ARenderer.

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

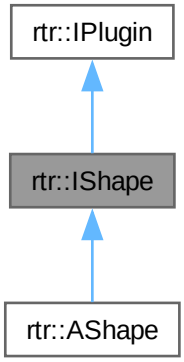
- App/include/RayTracer/Abstraction/IRenderer.hpp

4.17 rtr::IShape Class Reference

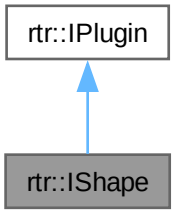
An interface used to get the shape's parameters based on the configuration file.

`#include <IShape.hpp>`

Inheritance diagram for rtr::IShape:



Collaboration diagram for rtr::IShape:



Public Member Functions

- virtual void setType (const ShapeType &type)=0

  Sets the type of the shape (sphere, plane, cone...).
- virtual void setMaterial (std::unique_ptr< AMaterial > material)=0

  Sets the material of the shape.
- virtual void setRadius (const double &radius)=0

  Sets the radius of the shape.
- virtual void setHeight (const double &height)=0

  Sets the height of the shape.
- virtual const ShapeType & getType () const =0

  Gets the type of the shape.
- virtual AMaterial & getMaterial ()=0

  Gets the material of the shape.
- virtual Vector & getPosition ()=0

  Gets the position of the shape.
- virtual Vector & getNormal ()=0

  Gets the normal of the shape.
- virtual Vector & getRotation ()=0

  Gets the rotation of the shape, used to create the shape.
- virtual const double & getRadius () const =0

  Gets the radius of the shape.
- virtual const double & getHeight () const =0

  Gets the height of the shape.
- virtual bool hits (std::pair< Vector, Vector > ray, RayHit &hit)=0

  Checks if the ray hits the shape, used to render the output file.
- virtual Vector getDistance (const Vector &point)=0

  Gets the distance between the shape and a point.

Public Member Functions inherited from rtr::IPlugin

- virtual std::string getPluginName () const =0

  Gets the name of the plugin.

4.17.1 Detailed Description

An interface used to get the shape's parameters based on the configuration file.

4.17.2 Member Function Documentation

4.17.2.1 getDistance()

```
virtual Vector rtr::IShape::getDistance (
            const Vector & point )  [pure virtual]
```

Gets the distance between the shape and a point.

Parameters

point	The point to check the distance with.

Returns

The distance between the shape and the point.

Implemented in rtr::AShape.

4.17.2.2 getHeight()

```
virtual const double & rtr::IShape::getHeight ( ) const  [pure virtual]
```

Gets the height of the shape.

Returns

The height of the shape as a double.

Implemented in rtr::AShape.

4.17.2.3 getMaterial()

```
virtual AMaterial & rtr::IShape::getMaterial ( )  [pure virtual]
```

Gets the material of the shape.

Returns

The material of the shape, using the AMaterial class.

Implemented in rtr::AShape.

4.17.2.4 getNormal()

```
virtual Vector & rtr::IShape::getNormal ( )  [pure virtual]
```

Gets the normal of the shape.

Returns

The normal of the shape, using the Vector class.

Implemented in rtr::AShape.

4.17.2.5 getPosition()

```
virtual Vector & rtr::IShape::getPosition ( )  [pure virtual]
```

Gets the position of the shape.

Returns

The position of the shape, using the Vector class.

Implemented in rtr::AShape.

4.17.2.6 getRadius()

```
virtual const double & rtr::IShape::getRadius ( ) const  [pure virtual]
```

Gets the radius of the shape.

Returns

The radius of the shape as a double, used to check the size of the shape.

Implemented in rtr::AShape.

4.17.2.7 getRotation()

```
virtual Vector & rtr::IShape::getRotation ( )  [pure virtual]
```

Gets the rotation of the shape, used to create the shape.

Returns

The rotation of the shape, using the Vector class.

Implemented in rtr::AShape.

4.17.2.8 getType()

```
virtual const ShapeType & rtr::IShape::getType ( ) const  [pure virtual]
```

Gets the type of the shape.

Returns

The type of the shape, using the ShapeType enum class.

Implemented in rtr::AShape.

4.17.2.9 hits()

```
virtual bool rtr::IShape::hits (
            std::pair< Vector, Vector > ray,
            RayHit & hit )  [pure virtual]
```

Checks if the ray hits the shape, used to render the output file.

Parameters

ray	A pair of vectors, to get the rays of the "camera".
hit	The hit of the ray, used to check if the ray hits the shape.

Returns

A boolean, true if the ray hits the shape, false otherwise.

4.17.2.10 setHeight()

```
virtual void rtr::IShape::setHeight (
            const double & height )  [pure virtual]
```

Sets the height of the shape.

Parameters

height	The height of the shape.

Implemented in rtr::AShape.

4.17.2.11 setMaterial()

```
virtual void rtr::IShape::setMaterial (
            std::unique_ptr< AMaterial > material )  [pure virtual]
```

Sets the material of the shape.

Parameters

material	The material of the shape (transparency, reflectivity...).

Implemented in rtr::AShape.

4.17.2.12 setRadius()

```
virtual void rtr::IShape::setRadius (
            const double & radius )  [pure virtual]
```

Sets the radius of the shape.

Parameters

radius	The radius of the shape.

Implemented in rtr::AShape.

4.17.2.13 setType()

```
virtual void rtr::IShape::setType (
            const ShapeType & type )  [pure virtual]
```

Sets the type of the shape (sphere, plane, cone...).

Parameters

type	The type of the shape.

Implemented in rtr::AShape.

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

- App/include/RayTracer/Abstraction/IShape.hpp

4.18 rtr::LightFactory Class Reference

A factory class for the lights.

`#include <Light.hpp>`

Static Public Member Functions

- static std::unique_ptr< ALight > createLight (const Color &color, const float &intensity)

  Creates a light based on the color and intensity (specific to the ambient light).
- static std::unique_ptr< ALight > createLight (const LightType &type, const Color &color, const float &intensity, const Vector &vector)

  Creates a light based on the type, color, intensity and a vector (used for directional and point lights).

4.18.1 Detailed Description

A factory class for the lights.

4.18.2 Member Function Documentation

4.18.2.1 createLight() [1/2]

```
static std::unique_ptr< ALight > rtr::LightFactory::createLight (
            const Color & color,
            const float & intensity )  [static]
```

Creates a light based on the color and intensity (specific to the ambient light).

Parameters

color	The color of the light.
intensity	The intensity of the light.

Returns

A unique pointer to the light.

4.18.2.2 createLight() [2/2]

```
static std::unique_ptr< ALight > rtr::LightFactory::createLight (
            const LightType & type,
            const Color & color,
            const float & intensity,
            const Vector & vector )  [static]
```

Creates a light based on the type, color, intensity and a vector (used for directional and point lights).

Parameters

type	The type of the light (defined in the LightType enum class).
color	The color of the light.
intensity	The intensity of the light.
vector	A vector to get the position of the light.

Returns

A unique pointer to the light.

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

- App/include/RayTracer/Factory/Light.hpp

4.19 rtr::MaterialFactory Class Reference

A factory class for the materials of the shapes.

`#include <Material.hpp>`

Static Public Member Functions

- static std::unique_ptr< AMaterial > createMaterial (const MaterialType &type, const float &floatValue)
  Creates a material based on the type and the color.

4.19.1 Detailed Description

A factory class for the materials of the shapes.

4.19.2 Member Function Documentation

4.19.2.1 createMaterial()

```
static std::unique_ptr< AMaterial > rtr::MaterialFactory::createMaterial (
            const MaterialType & type,
            const float & floatValue )  [static]
```

Creates a material based on the type and the color.

Parameters

type	The type of the material (defined in the MaterialType enum class).
floatValue	The value of the transparency and reflectiveness to set.

Returns

A unique pointer to the material.

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

- App/include/RayTracer/Factory/Material.hpp

4.20 rtr::Parser Class Reference

Class dedicated to the parsing of configuration files and command-line arguments.

`#include <Parser.hpp>`

Classes

- class ParserException

  Exception class for errors in the parsers.

Static Public Member Functions

- static int parseArgs (const std::string &filePath)

  Parses command-line arguments.
- static std::unique_ptr< rtr::Scene > parseFile (const std::string &filePath)

  Parses a configuration file and returns a Scene object.
- static void parseRenderer (const libconfig::Setting &renderer, Scene &scene)

  Parses the renderer settings from a configuration file.
- static void parseCamera (const libconfig::Setting &camera, Scene &scene)

  Parses the camera settings from a configuration file.
- static ShapeType parseShapeType (const std::string &type)

  Parses the shape type from a string.
- static void parseShapes (const libconfig::Setting &shapesSetting, Scene &scene)

  Parses the shapes settings from a configuration file.
- static std::unique_ptr< AMaterial > parseMaterial (const libconfig::Setting &materialSetting)

  Parses the material settings from a configuration file.
- static LightType parseLightType (const std::string &type)

  Parses the light settings from a configuration file.
- static void parseLights (const libconfig::Setting &lightsSetting, Scene &scene)

  Parses the light settings from a configuration file.
- template<typename T , typename ConversionFunc >
  static T getVector (const libconfig::Setting &setting, ConversionFunc convert)

  Templated function to get a vector from a configuration file setting.
- template<typename T >
  static T convertInt (const libconfig::Setting &setting)

  Templated function to convert an integer value from a configuration file setting.

4.20.1 Detailed Description

Class dedicated to the parsing of configuration files and command-line arguments.

4.20.2 Member Function Documentation

4.20.2.1 convertInt()

```
template<typename T >
static T rtr::Parser::convertInt (
            const libconfig::Setting & setting )  [static]
```

Templated function to convert an integer value from a configuration file setting.

Template Parameters

T	The type of the value.

Parameters

setting	The setting in the configuration file.

Returns

The parsed value.

4.20.2.2 getVector()

```
template<typename T , typename ConversionFunc >
static T rtr::Parser::getVector (
            const libconfig::Setting & setting,
            ConversionFunc convert )  [static]
```

Templated function to get a vector from a configuration file setting.

Template Parameters

T	The type of the vector.
ConversionFunc	The function to convert the vector elements.

Parameters

setting	The setting in the configuration file.
convert	The conversion function.

Returns

The parsed vector.

4.20.2.3 parseArgs()

```
static int rtr::Parser::parseArgs (
            const std::string & filePath )  [static]
```

Parses command-line arguments.

Parameters

filePath	The path to the configuration file.

Returns

0 on success, 1 on failure.

4.20.2.4 parseCamera()

```
static void rtr::Parser::parseCamera (
            const libconfig::Setting & camera,
            Scene & scene )  [static]
```

Parses the camera settings from a configuration file.

Parameters

camera	The camera settings in the configuration file.
scene	The Scene object to update with the parsed settings.

4.20.2.5 parseFile()

```
static std::unique_ptr< rtr::Scene > rtr::Parser::parseFile (
            const std::string & filePath )  [static]
```

Parses a configuration file and returns a Scene object.

Parameters

filePath	The path to the configuration file.

Returns

A unique_ptr to a Scene object.

4.20.2.6 parseLights()

```
static void rtr::Parser::parseLights (
            const libconfig::Setting & lightsSetting,
            Scene & scene )  [static]
```

Parses the light settings from a configuration file.

Parameters

lightsSetting	The lights settings in the configuration files.
scene	The Scene object to update with the parsed settings.

4.20.2.7 parseLightType()

```
static LightType rtr::Parser::parseLightType (
            const std::string & type )  [static]
```

Parses the light settings from a configuration file.

Parameters

type	The light type as a string.

Returns

The parsed light type.

4.20.2.8 parseMaterial()

```
static std::unique_ptr< AMaterial > rtr::Parser::parseMaterial (
            const libconfig::Setting & materialSetting )  [static]
```

Parses the material settings from a configuration file.

Parameters

materialSetting	The material settings in the configuration file.

Returns

A unique_ptr to a Material object.

4.20.2.9 parseRenderer()

```
static void rtr::Parser::parseRenderer (
            const libconfig::Setting & renderer,
            Scene & scene )  [static]
```

Parses the renderer settings from a configuration file.

Parameters

renderer	The renderer settings in the configuration file.
scene	The Scene object to update with the parsed settings.

4.20.2.10 parseShapes()

```
static void rtr::Parser::parseShapes (
            const libconfig::Setting & shapesSetting,
            Scene & scene )  [static]
```

Parses the shapes settings from a configuration file.

Parameters

shapesSetting	The shapes settings in the configuration file.
scene	The Scene object to update with the parsed settings.

4.20.2.11 parseShapeType()

```
static ShapeType rtr::Parser::parseShapeType (
            const std::string & type )  [static]
```

Parses the shape type from a string.

Parameters

type	The shape type as a string.

Returns

The parsed shape type.

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

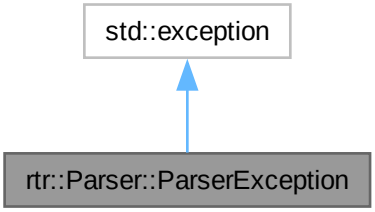
- App/include/RayTracer/Parser.hpp

4.21 rtr::Parser::ParserException Class Reference

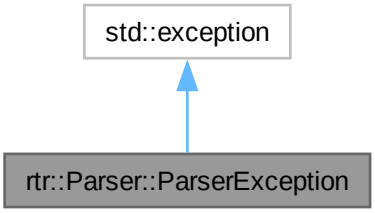
Exception class for errors in the parsers.

`#include <Parser.hpp>`

Inheritance diagram for rtr::Parser::ParserException:



Collaboration diagram for rtr::Parser::ParserException:



Public Member Functions

- **ParserException** (std::string msg)
- **ParserException** (const ParserException &)=delete
- ParserException & **operator=** (const ParserException &)=delete
- **ParserException** (const ParserException &&)=delete
- ParserException & **operator=** (const ParserException &&)=delete
- const char * **what** () const noexcept override

  Returns the error message.

4.21.1 Detailed Description

Exception class for errors in the parsers.

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

- App/include/RayTracer/Parser.hpp

4.22 rtr::PluginLoader Class Reference

A class to load the plugins.

`#include <Plugin.hpp>`

Public Types

- using **PluginCreator** = std::unique_ptr<IPlugin> (*)()

Public Member Functions

- template<typename T >
  std::unique_ptr< T > getPlugin (const std::string &pluginName)

  Gets the plugin based on the name.

Static Public Member Functions

- static PluginLoader & getInstance ()

  Gets the instance of the plugin loader.

4.22.1 Detailed Description

A class to load the plugins.

4.22.2 Member Function Documentation

4.22.2.1 getInstance()

```
static PluginLoader & rtr::PluginLoader::getInstance ( )  [inline], [static]
```

Gets the instance of the plugin loader.

Returns

A reference to the plugin loader.

4.22.2.2 getPlugin()

```
template<typename T >
std::unique_ptr< T > rtr::PluginLoader::getPlugin (
            const std::string & pluginName )
```

Gets the plugin based on the name.

Parameters

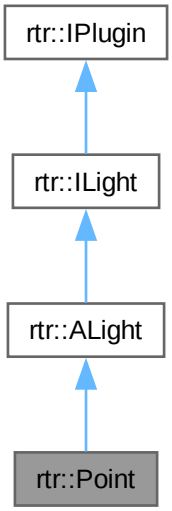
pluginName	The name of the plugin.

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

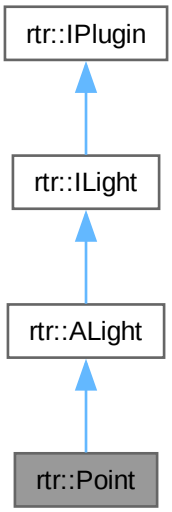
- App/include/RayTracer/Loader/Plugin.hpp

4.23 rtr::Point Class Reference

Inheritance diagram for rtr::Point:



Collaboration diagram for rtr::Point:



Public Member Functions

- std::string getPluginName () const override

  Gets the name of the plugin.
- Color LightColor (const Vector &normal, const Color &col) override

  Creates light effects based on the light type.
- Vector & getDirection () override

  Gets the direction of the light based on the configuration file.

Public Member Functions inherited from rtr::ALight

- void setType (const LightType &type) override

  Sets the type of the light (directional, ambient or point).
- void setIntensity (const float &intensity) override

  Sets the intensity of the light, based on the configuration file.
- const LightType & getType () const override

  Gets the type of the light based on the configuration file.
- Vector & getPosition () override

  Gets the position of the light based on the configuration file.
- Color & getColor () override

  Gets the color of the light based on the configuration file.
- float & getIntensity () override

  Gets the intensity of the light based on the configuration file.

4.23.1 Member Function Documentation

4.23.1.1 getDirection()

```
Vector & rtr::Point::getDirection ( )  [inline], [override], [virtual]
```

Gets the direction of the light based on the configuration file.

Reimplemented from rtr::ALight.

4.23.1.2 getPluginName()

```
std::string rtr::Point::getPluginName ( ) const  [inline], [override], [virtual]
```

Gets the name of the plugin.

Returns

A string of the plugin's name, defined as const expressions.

Implements rtr::IPlugin.

4.23.1.3 LightColor()

```
Color rtr::Point::LightColor (
            const Vector & normal,
            const Color & col )  [override], [virtual]
```

Creates light effects based on the light type.

Parameters

normal	The normal of the shape.
col	The current color of the shape.

Returns

The new color of the shape with the light effects.

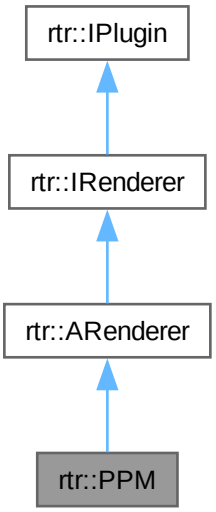
Implements rtr::ILight.

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

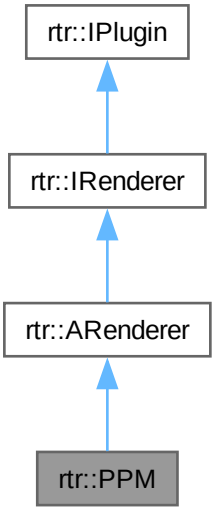
- App/plugins/Light/Point/include/RayTracer/Point.hpp

4.24 rtr::PPM Class Reference

Inheritance diagram for rtr::PPM:



Collaboration diagram for rtr::PPM:



Public Member Functions

- std::string getPluginName () const override

  Gets the name of the plugin.
- void render (const std::vector< std::unique_ptr< AShape > > &shapes, const std::vector< std::unique_ptr< ALight > > &lights, const Camera &camera) override

  Renders the scene based on the shapes, lights and camera.
- void **writePixels** (const Color color, const std::size_t width, const std::size_t height)
- void **writeToFile** (const std::string &width, const std::string &height)
- bool **isShadowed** (const Vector &lightDir, const Vector &point, const std::vector< std::unique_ptr< AShape > > &shapes)

Public Member Functions inherited from rtr::ARenderer

- void setType (const RendererType &rendererType) override

  Sets the type of the renderer.
- void setName (const std::string &name) override

  Sets the name of the renderer.
- void setPixels (const std::vector< std::vector< rtr::Color > > &pixels) override

  Sets the pixels of the renderer.
- const RendererType & getType () const override

  Gets the type of the renderer based on the configuration file.
- Resolution & getResolution () override

  Gets the resolution of the renderer based on the configuration file.
- Color & getBackgroundColor () override

  Gets the background color of the renderer based on the configuration file.
- const std::string & getName () const override

  Gets the name of the renderer based on the configuration file.
- std::vector< std::vector< rtr::Color > > & getPixels () override

  Gets the pixels of the renderer based on the configuration file.

Static Public Member Functions

- static std::string **getHeader** (const std::string &width, const std::string &height)

4.24.1 Member Function Documentation

4.24.1.1 getPluginName()

```
std::string rtr::PPM::getPluginName ( ) const  [inline], [override], [virtual]
```

Gets the name of the plugin.

Returns

A string of the plugin's name, defined as const expressions.

Implements rtr::IPlugin.

4.24.1.2 render()

```
void rtr::PPM::render (
            const std::vector< std::unique_ptr< AShape > > & shapes,
            const std::vector< std::unique_ptr< ALight > > & lights,
            const Camera & camera )  [override], [virtual]
```

Renders the scene based on the shapes, lights and camera.

Parameters

shapes	The shapes of the scene.
lights	The lights of the scene.
camera	The camera of the scene.

Implements rtr::IRenderer.

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

- App/plugins/Renderer/PPM/include/RayTracer/PPM.hpp

4.25 ray_hit_s Struct Reference

A struct representing a ray hit in 3D space.

```
#include <RayHit.hpp>
```

4.25.1 Detailed Description

A struct representing a ray hit in 3D space.

Type alias for a ray hit.

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

- App/include/RayTracer/Utils/RayHit.hpp

4.26 rtr::RayHit Class Reference

A class representing a ray hit in 3D space.

`#include <RayHit.hpp>`

Public Member Functions

- const ray_hit_t & **getRayHit** () const noexcept
- void setRayHit (const ray_hit_t &ray_hit) noexcept

  Sets the ray hit data.
- void setRayHit (const Vector &point, const Vector &normal, const double &distance) noexcept

  Sets the ray hit data.
- void setPoint (const Vector &point) noexcept

  Sets the point of intersection.
- void setNormal (const Vector &normal) noexcept

  Sets the normal vector at the point of intersection.
- void setDistance (const double &distance) noexcept

  Sets the distance from the ray origin to the point of intersection.

4.26.1 Detailed Description

A class representing a ray hit in 3D space.

4.26.2 Member Function Documentation

4.26.2.1 setDistance()

```
void rtr::RayHit::setDistance (
            const double & distance )  [inline], [noexcept]
```

Sets the distance from the ray origin to the point of intersection.

Parameters

distance	The distance to set.

4.26.2.2 setNormal()

```
void rtr::RayHit::setNormal (
            const Vector & normal )  [inline], [noexcept]
```

Sets the normal vector at the point of intersection.

Parameters

normal	The normal vector to set.

4.26.2.3 setPoint()

```
void rtr::RayHit::setPoint (
            const Vector & point )  [inline], [noexcept]
```

Sets the point of intersection.

Parameters

point	The point of intersection to set.

4.26.2.4 setRayHit() [1/2]

```
void rtr::RayHit::setRayHit (
            const ray_hit_t & ray_hit )  [inline], [noexcept]
```

Sets the ray hit data.

Parameters

ray_hit	The ray hit data to set

4.26.2.5 setRayHit() [2/2]

```
void rtr::RayHit::setRayHit (
            const Vector & point,
            const Vector & normal,
            const double & distance )  [inline], [noexcept]
```

Sets the ray hit data.

Parameters

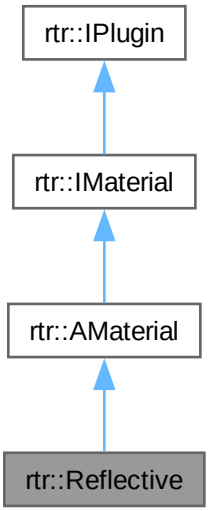
point	The point of intersection.
normal	The normal vector at the point of intersection.
distance	The distance from the ray origin to the point of intersection.

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

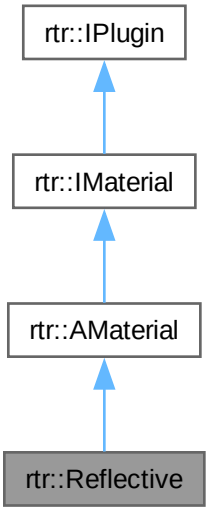
- App/include/RayTracer/Utils/RayHit.hpp

4.27 rtr::Reflective Class Reference

Inheritance diagram for rtr::Reflective:



Collaboration diagram for rtr::Reflective:



Public Member Functions

- void applyMaterial (Color *color) override

  Applies the material to the shape (transparency and reflectiveness).
- std::string getPluginName () const override

  Gets the name of the plugin.

Public Member Functions inherited from rtr::AMaterial

- void setType (const MaterialType &type) override

  Sets the type of the material.
- void setReflectivity (const float &reflectivity) override

  Sets the color of the material.
- void setTransparency (const float &transparency) override

  Sets the transparency of the material.
- const MaterialType & getType () const override

  Gets the type of the material based on the configuration file.
- Color & getColor () override

  Gets the color of the material based on the configuration file.
- const float & getReflectivity () const override

  Gets the reflectiveness of the material based on the configuration file.
- const float & getTransparency () const override

  Gets the transparency of the material based on the configuration file.

4.27.1 Member Function Documentation

4.27.1.1 applyMaterial()

```
void rtr::Reflective::applyMaterial (
            Color * color )  [inline], [override], [virtual]
```

Applies the material to the shape (transparency and reflectiveness).

Implements rtr::IMaterial.

4.27.1.2 getPluginName()

```
std::string rtr::Reflective::getPluginName ( ) const  [inline], [override], [virtual]
```

Gets the name of the plugin.

Returns

A string of the plugin's name, defined as const expressions.

Implements rtr::IPlugin.

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

- App/plugins/Material/Reflective/include/RayTracer/Reflective.hpp

4.28 rtr::RendererFactory Class Reference

A factory class to create the renderers.

`#include <Renderer.hpp>`

Static Public Member Functions

- static std::unique_ptr< ARenderer > createRenderer (const RendererType &type, const std::string &name, const Resolution &resolution, const Color &backgroundColor)

  Creates a renderer based on the type, name, resolution and background color.

4.28.1 Detailed Description

A factory class to create the renderers.

4.28.2 Member Function Documentation

4.28.2.1 createRenderer()

```
static std::unique_ptr< ARenderer > rtr::RendererFactory::createRenderer (
            const RendererType & type,
            const std::string & name,
            const Resolution & resolution,
            const Color & backgroundColor )  [static]
```

Creates a renderer based on the type, name, resolution and background color.

Parameters

type	The type of the renderer (defined in the RendererType enum class).
name	A string of the renderer's name.
resolution	The resolution of the renderer.
backgroundColor	The background color of the renderer.

Returns

A unique pointer to the renderer.

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

- App/include/RayTracer/Factory/Renderer.hpp

4.29 rtr::Resolution Class Reference

Class representing the resolution of an image.

`#include <Resolution.hpp>`

Public Member Functions

- **Resolution** (const uint16_t &width, const uint16_t &height)
- **Resolution** (const resolution_t &resolution)
- void setWidth (const uint16_t &width)

  Sets the width of the resolution.
- void setHeight (const uint16_t &height)

  Sets the height of the resolution.
- void setResolution (const uint16_t &width, const uint16_t &height)

  Sets the resolution to the given width and height.
- void setResolution (const resolution_t &resolution)

  Sets the resolution to the given resolution struct.
- uint16_t getWidth () const

  Gets the width of the resolution.
- uint16_t getHeight () const

  Gets the height of the resolution.
- resolution_t getValue () const

  Gets the resolution as a struct.

4.29.1 Detailed Description

Class representing the resolution of an image.

4.29.2 Member Function Documentation

4.29.2.1 getHeight()

```
uint16_t rtr::Resolution::getHeight ( ) const  [inline]
```

Gets the height of the resolution.

Returns

The height of the resolution.

4.29.2.2 getValue()

```
resolution_t rtr::Resolution::getValue ( ) const  [inline]
```

Gets the resolution as a struct.

Returns

The resolution struct.

4.29.2.3 getWidth()

```
uint16_t rtr::Resolution::getWidth ( ) const  [inline]
```

Gets the width of the resolution.

Returns

The width of the resolution.

4.29.2.4 setHeight()

```
void rtr::Resolution::setHeight (
            const uint16_t & height )  [inline]
```

Sets the height of the resolution.

Parameters

height	The height to set.

4.29.2.5 setResolution() [1/2]

```
void rtr::Resolution::setResolution (
```

```
            const resolution_t & resolution )  [inline]
```

Sets the resolution to the given resolution struct.

Parameters

resolution	The resolution struct to set.

4.29.2.6 setResolution() [2/2]

```
void rtr::Resolution::setResolution (
            const uint16_t & width,
            const uint16_t & height )  [inline]
```

Sets the resolution to the given width and height.

Parameters

width	The width of the resolution.
height	The height of the resolution.

4.29.2.7 setWidth()

```
void rtr::Resolution::setWidth (
            const uint16_t & width )  [inline]
```

Sets the width of the resolution.

Parameters

width	The width to set.

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

- App/include/RayTracer/Utils/Resolution.hpp

4.30 resolution_s Struct Reference

A struct representing the resolution of an image.

`#include <Resolution.hpp>`

4.30.1 Detailed Description

A struct representing the resolution of an image.

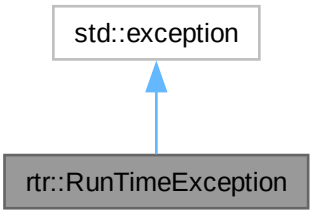
Type alias for the resolution of an image.

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

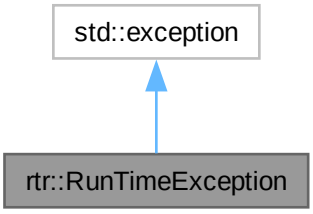
- App/include/RayTracer/Utils/Resolution.hpp

4.31 rtr::RunTimeException Class Reference

Inheritance diagram for rtr::RunTimeException:



Collaboration diagram for rtr::RunTimeException:



Public Member Functions

- **RunTimeException** (std::string msg)
- **RunTimeException** (const RunTimeException &)=delete
- RunTimeException & **operator=** (const RunTimeException &)=delete
- **RunTimeException** (const RunTimeException &&)=delete
- RunTimeException & **operator=** (const RunTimeException &&)=delete
- const char * **what** () const noexcept override

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

- App/include/RayTracer/Exception/RunTime.hpp

4.32 rtr::Scene Class Reference

A class to represent the scene.

`#include <Scene.hpp>`

Public Member Functions

- void setCamera (const Camera &camera)

  Sets the camera of the scene.
- void setRenderer (std::unique_ptr< ARenderer > renderer)

  Sets the renderer of the scene.
- void addShape (std::unique_ptr< AShape > shape)

  Adds a shape to the scene.
- void addLight (std::unique_ptr< ALight > light)

  Adds a light to the scene.
- Camera & **getCamera** ()

  Gets the camera of the scene.
- const std::unique_ptr< ARenderer > & getRenderer () const

  Gets the renderer of the scene.
- const std::vector< std::unique_ptr< AShape > > & getShapes () const

  Gets the shapes of the scene.
- const std::vector< std::unique_ptr< ALight > > & getLights () const

  Gets the lights of the scene.

4.32.1 Detailed Description

A class to represent the scene.

4.32.2 Member Function Documentation

4.32.2.1 addLight()

```
void rtr::Scene::addLight (
            std::unique_ptr< ALight > light )  [inline]
```

Adds a light to the scene.

Parameters

light	The light to add, which can be a point light, a directional light and an ambient light.

4.32.2.2 addShape()

```
void rtr::Scene::addShape (
            std::unique_ptr< AShape > shape )  [inline]
```

Adds a shape to the scene.

Parameters

shape	The shape to add, which can be a sphere, a plane and a cone.

4.32.2.3 getLights()

```
const std::vector< std::unique_ptr< ALight > > & rtr::Scene::getLights ( ) const  [inline]
```

Gets the lights of the scene.

Returns

A vector of unique pointers to the lights.

4.32.2.4 getRenderer()

```
const std::unique_ptr< ARenderer > & rtr::Scene::getRenderer ( ) const  [inline]
```

Gets the renderer of the scene.

Returns

A unique pointer to the renderer.

4.32.2.5 getShapes()

```
const std::vector< std::unique_ptr< AShape > > & rtr::Scene::getShapes ( ) const  [inline]
```

Gets the shapes of the scene.

Returns

A vector of unique pointers to the shapes.

4.32.2.6 setCamera()

```
void rtr::Scene::setCamera (
            const Camera & camera )  [inline]
```

Sets the camera of the scene.

Parameters

camera	The camera to set.

4.32.2.7 setRenderer()

```
void rtr::Scene::setRenderer (
            std::unique_ptr< ARenderer > renderer )  [inline]
```

Sets the renderer of the scene.

Parameters

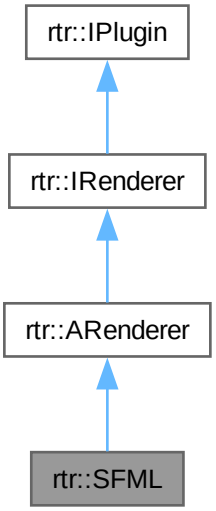
renderer	The renderer to set.

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

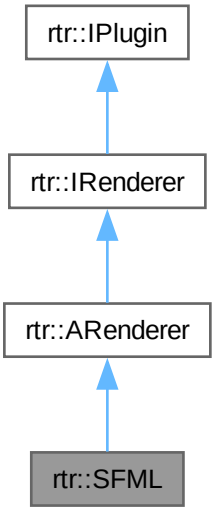
- App/include/RayTracer/Scene/Scene.hpp

4.33 rtr::SFML Class Reference

Inheritance diagram for rtr::SFML:



Collaboration diagram for rtr::SFML:



Public Member Functions

- std::string getPluginName () const override

  Gets the name of the plugin.
- void render (const std::vector< std::unique_ptr< AShape > > &shapes, const std::vector< std::unique_ptr< ALight > > &lights, const Camera &camera) override

  Renders the scene based on the shapes, lights and camera.

Public Member Functions inherited from rtr::ARenderer

- void setType (const RendererType &rendererType) override

  Sets the type of the renderer.
- void setName (const std::string &name) override

  Sets the name of the renderer.
- void setPixels (const std::vector< std::vector< rtr::Color > > &pixels) override

  Sets the pixels of the renderer.
- const RendererType & getType () const override

  Gets the type of the renderer based on the configuration file.
- Resolution & getResolution () override

  Gets the resolution of the renderer based on the configuration file.
- Color & getBackgroundColor () override

  Gets the background color of the renderer based on the configuration file.
- const std::string & getName () const override

  Gets the name of the renderer based on the configuration file.
- std::vector< std::vector< rtr::Color > > & getPixels () override

  Gets the pixels of the renderer based on the configuration file.

4.33.1 Member Function Documentation

4.33.1.1 getPluginName()

```
std::string rtr::SFML::getPluginName ( ) const  [inline], [override], [virtual]
```

Gets the name of the plugin.

Returns

A string of the plugin's name, defined as const expressions.

Implements rtr::IPlugin.

4.33.1.2 render()

```
void rtr::SFML::render (
            const std::vector< std::unique_ptr< AShape > > & shapes,
            const std::vector< std::unique_ptr< ALight > > & lights,
            const Camera & camera )  [override], [virtual]
```

Renders the scene based on the shapes, lights and camera.

Parameters

shapes	The shapes of the scene.
lights	The lights of the scene.
camera	The camera of the scene.

Implements rtr::IRenderer.

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

- App/plugins/Renderer/SFML/include/RayTracer/SFML.hpp

4.34 rtr::ShapeFactory Class Reference

A factory class for the shapes.

`#include <Shape.hpp>`

Static Public Member Functions

- static std::unique_ptr< AShape > createShape (const Vector &position, const Vector &normal)
  Creates a shape based on the position and normal (for the plane).
- static std::unique_ptr< AShape > createShape (const Vector &position, const double &radius)
  Creates a shape based on the position and radius (for the sphere).
- static std::unique_ptr< AShape > createShape (const ShapeType &type, const Vector &position, const Vector &rotation, const double &radius, const double &height)
  Creates a shape based on the type position, rotation, radius and height.

4.34.1 Detailed Description

A factory class for the shapes.

4.34.2 Member Function Documentation

4.34.2.1 createShape() [1/3]

```
static std::unique_ptr< AShape > rtr::ShapeFactory::createShape (
            const ShapeType & type,
            const Vector & position,
            const Vector & rotation,
            const double & radius,
            const double & height )  [static]
```

Creates a shape based on the type position, rotation, radius and height.

Parameters

type	The type of the shape (defined in the ShapeType enum class).
position	Used to get the position of the shape.
rotation	Used to get the rotation of the shape.
radius	Used to get the radius of the shape.
height	Used to get the height of the shape.

Returns

A unique pointer to the shape.

4.34.2.2 createShape() [2/3]

```
static std::unique_ptr< AShape > rtr::ShapeFactory::createShape (
            const Vector & position,
            const double & radius )  [static]
```

Creates a shape based on the position and radius (for the sphere).

Parameters

position	Used to get the position of the shape.
radius	Used to get the radius of the shape.

Returns

A unique pointer to the shape.

4.34.2.3 createShape() [3/3]

```
static std::unique_ptr< AShape > rtr::ShapeFactory::createShape (
            const Vector & position,
            const Vector & normal )  [static]
```

Creates a shape based on the position and normal (for the plane).

Parameters

position	Used to get the position of the shape.
normal	Used to get the normal of the shape.

Returns

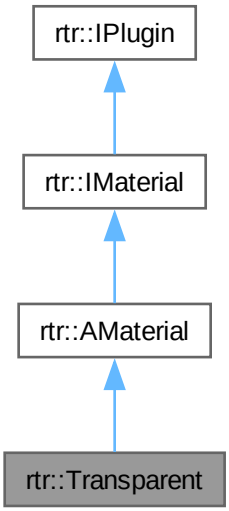
A unique pointer to the shape.

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

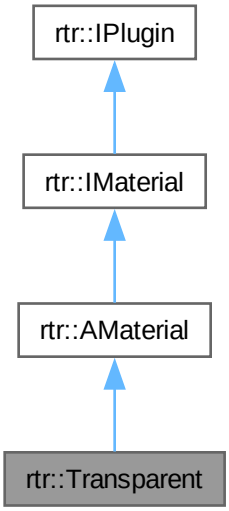
- App/include/RayTracer/Factory/Shape.hpp

4.35 rtr::Transparent Class Reference

Inheritance diagram for rtr::Transparent:



Collaboration diagram for rtr::Transparent:



Public Member Functions

- void applyMaterial (Color *color) override

  Applies the material to the shape (transparency and reflectiveness).
- std::string getPluginName () const override

  Gets the name of the plugin.

Public Member Functions inherited from rtr::AMaterial

- void setType (const MaterialType &type) override

  Sets the type of the material.
- void setReflectivity (const float &reflectivity) override

  Sets the color of the material.
- void setTransparency (const float &transparency) override

  Sets the transparency of the material.
- const MaterialType & getType () const override

  Gets the type of the material based on the configuration file.
- Color & getColor () override

  Gets the color of the material based on the configuration file.
- const float & getReflectivity () const override

  Gets the reflectiveness of the material based on the configuration file.
- const float & getTransparency () const override

  Gets the transparency of the material based on the configuration file.

4.35.1 Member Function Documentation

4.35.1.1 applyMaterial()

```
void rtr::Transparent::applyMaterial (
            Color * color )  [inline], [override], [virtual]
```

Applies the material to the shape (transparency and reflectiveness).

Implements rtr::IMaterial.

4.35.1.2 getPluginName()

```
std::string rtr::Transparent::getPluginName ( ) const  [inline], [override], [virtual]
```

Gets the name of the plugin.

Returns

A string of the plugin's name, defined as const expressions.

Implements rtr::IPlugin.

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

- App/plugins/Material/Transparent/include/RayTracer/Transparent.hpp

4.36 rtr::Vector Class Reference

Public Member Functions

- **Vector** (const double x, const double y, const double z)
- **Vector** (const vector_t position)
- void setX (const double x)

  Sets the x-component of the vector.
- void setY (const double y)

  Sets the y-component of the vector.
- void setZ (const double z)

  Sets the z-component of the vector.
- void setVector (const double x, const double y, const double z)

  Sets the vector to the given x, y, and z values.
- void setVector (const vector_t &position)

  Sets the vector to the given vector struct.
- double getX () const

  Gets the x-component of the vector.
- double getY () const

  Gets the y-component of the vector.
- double getZ () const

  Gets the z-component of the vector.

- vector_t getValue () const

  Gets the vector as a struct.
- Vector operator+ (const Vector &other) const

  Adds two vectors.
- Vector operator+ (const double scalar) const

  Adds a scalar to the vector.
- Vector operator- (const Vector &other) const

  Subtracts two vectors.
- Vector operator∗ (const Vector &other) const

  Multiplies two vectors.
- Vector operator∗ (const double scalar) const

  Multiplies the vector by a scalar.
- Vector operator/ (const double scalar) const

  Divides the vector by a scalar.
- double length () const

  Gets the length of the vector.
- double dot (const Vector &other) const

  Calculates the dot product of the vector and another vector.
- Vector cross (const Vector &other) const

  Calculates the cross product of the vector and another vector.
- Vector normalize () const

  Normalizes the vector.

4.36.1 Member Function Documentation

4.36.1.1 cross()

```
Vector rtr::Vector::cross (
            const Vector & other ) const  [inline]
```

Calculates the cross product of the vector and another vector.

Parameters

other	The other vector to calculate the cross product with.

Returns

The cross product of the two vectors.

4.36.1.2 dot()

```
double rtr::Vector::dot (
            const Vector & other ) const  [inline]
```

Calculates the dot product of the vector and another vector.

Parameters

other	The other vector to calculate the dot product with.

Returns

The dot product of the two vectors.

4.36.1.3 getValue()

```
vector_t rtr::Vector::getValue ( ) const  [inline]
```

Gets the vector as a struct.

Returns

The vector struct.

4.36.1.4 getX()

```
double rtr::Vector::getX ( ) const  [inline]
```

Gets the x-component of the vector.

Returns

The x-component of the vector.

4.36.1.5 getY()

```
double rtr::Vector::getY ( ) const  [inline]
```

Gets the y-component of the vector.

Returns

The y-component of the vector.

4.36.1.6 getZ()

```
double rtr::Vector::getZ ( ) const  [inline]
```

Gets the z-component of the vector.

Returns

The z-component of the vector.

4.36.1.7 length()

```
double rtr::Vector::length ( ) const  [inline]
```

Gets the length of the vector.

Returns

The length of the vector.

4.36.1.8 normalize()

```
Vector rtr::Vector::normalize ( ) const  [inline]
```

Normalizes the vector.

Returns

The normalized vector.

4.36.1.9 operator∗() [1/2]

```
Vector rtr::Vector::operator* (
            const double scalar ) const  [inline]
```

Multiplies the vector by a scalar.

Parameters

scalar	The scalar to multiply by.

Returns

The product of the vector and the scalar.

4.36.1.10 operator∗() [2/2]

```
Vector rtr::Vector::operator* (
            const Vector & other ) const  [inline]
```

Multiplies two vectors.

Parameters

other	The other vector to multiply.

Returns

The product of the two vectors.

4.36.1.11 operator+() [1/2]

```
Vector rtr::Vector::operator+ (
            const double scalar ) const  [inline]
```

Adds a scalar to the vector.

Parameters

scalar	The scalar to add.

Returns

The sum of the vector and the scalar.

4.36.1.12 operator+() [2/2]

```
Vector rtr::Vector::operator+ (
            const Vector & other ) const  [inline]
```

Adds two vectors.

Parameters

other	The other vector to add.

Returns

The sum of the two vectors.

4.36.1.13 operator-()

```
Vector rtr::Vector::operator- (
            const Vector & other ) const  [inline]
```

Subtracts two vectors.

Parameters

other	The other vector to subtract.

Returns

The difference of the two vectors.

4.36.1.14 operator/()

```
Vector rtr::Vector::operator/ (
            const double scalar ) const  [inline]
```

Divides the vector by a scalar.

Parameters

scalar	The scalar to divide by.

Returns

The quotient of the vector and the scalar.

4.36.1.15 setVector() [1/2]

```
void rtr::Vector::setVector (
            const double x,
            const double y,
            const double z )  [inline]
```

Sets the vector to the given x, y, and z values.

Parameters

x	The x-component of the vector.
y	The y-component of the vector.
z	The z-component of the vector.

4.36.1.16 setVector() [2/2]

```
void rtr::Vector::setVector (
            const vector_t & position )  [inline]
```

Sets the vector to the given vector struct.

Parameters

position	The vector struct to set.

4.36.1.17 setX()

```
void rtr::Vector::setX (
            const double x )  [inline]
```

Sets the x-component of the vector.

Parameters

x	The x-component to set.

4.36.1.18 setY()

```
void rtr::Vector::setY (
            const double y )  [inline]
```

Sets the y-component of the vector.

Parameters

y	The y-component to set.

4.36.1.19 setZ()

```
void rtr::Vector::setZ (
            const double z )  [inline]
```

Sets the z-component of the vector.

Parameters

z	The z-component to set.

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

- App/include/RayTracer/Utils/Vector.hpp

4.37 vector_s Struct Reference

A struct representing a 3D vector.

`#include <Vector.hpp>`

4.37.1 Detailed Description

A struct representing a 3D vector.

Type alias for a 3D vector.

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

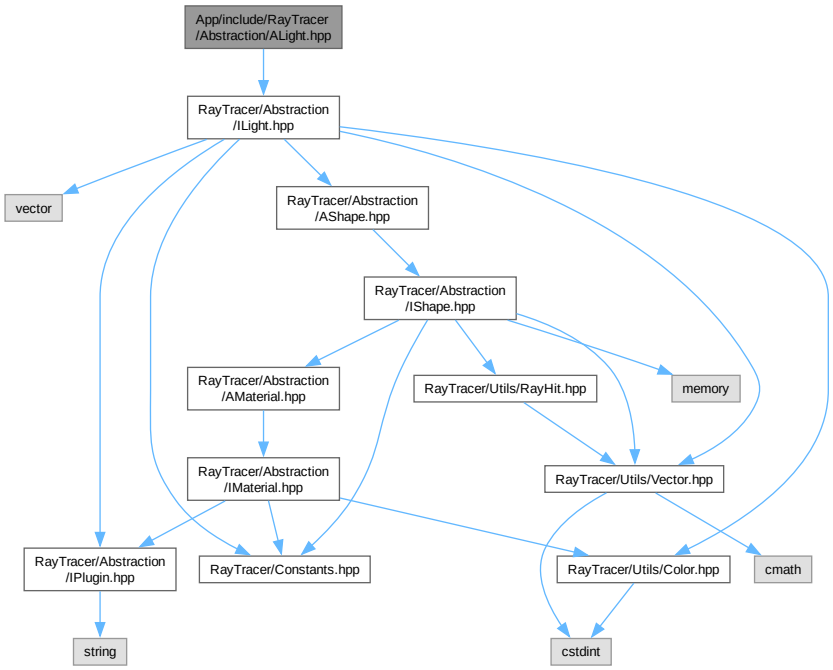
- App/include/RayTracer/Utils/Vector.hpp

Chapter 5

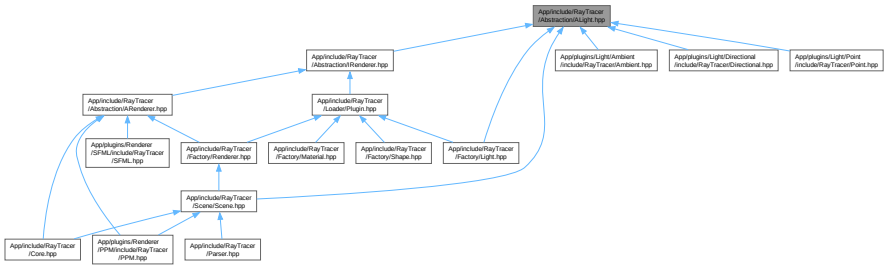
File Documentation

5.1 App/include/RayTracer/Abstraction/ALight.hpp File Reference

`#include "RayTracer/Abstraction/ILight.hpp"`
Include dependency graph for ALight.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class rtr::ALight

  An abstract class for lights.

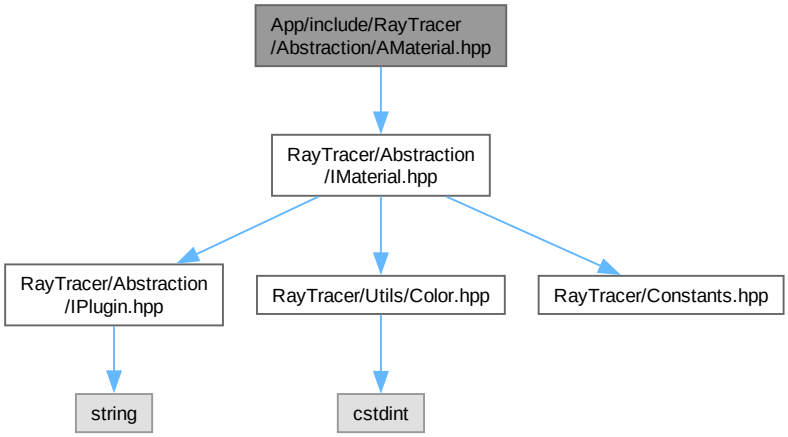
5.2 ALight.hpp

Go to the documentation of this file.

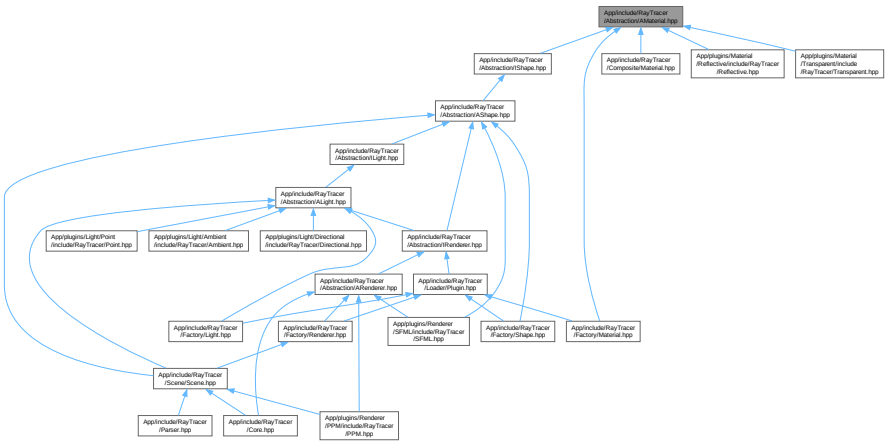
```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** Raytracer
00004 ** File description:
00005 ** ALight.cpp
00006 */
00007
00009 #ifndef RAYTRACER_ALIGHT_HPP
00010 #define RAYTRACER_ALIGHT_HPP
00011
00012 #include "RayTracer/Abstraction/ILight.hpp"
00013
00014 namespace rtr {
00015
00018     class ALight : public ILight {
00019
00020         public:
00021
00022             ~ALight() override = default;
00023
00024             void setType(const LightType &type) override { m_type = type; };
00025             void setIntensity(const float &intensity) override { m_intensity = intensity; };
00026
00027             [[nodiscard]] const LightType& getType() const override { return m_type; };
00028             [[nodiscard]] Vector& getPosition() override { return m_position; };
00029             [[nodiscard]] Vector& getDirection() override { return m_direction; };
00030             [[nodiscard]] Color& getColor() override { return m_color; };
00031             [[nodiscard]] float& getIntensity() override { return m_intensity; };
00032
00033         private:
00034
00036             LightType m_type{LightType::NONE};
00037
00039             Vector m_position{0, 0, 0};
00040
00042             Vector m_direction{0, 0, 0};
00043
00045             Color m_color{0, 0, 0};
00046
00048             float m_intensity{0};
00049
00050     }; // class ALight
00051
00052 } // namespace RayTracer
00053
00054 #endif //RAYTRACER_ALIGHT_HPP
```

5.3 App/include/RayTracer/Abstraction/AMaterial.hpp File Reference

`#include "RayTracer/Abstraction/IMaterial.hpp"`
Include dependency graph for AMaterial.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class rtr::AMaterial

  An abstract class for materials, based on the interface IMaterials.

5.4 AMaterial.hpp

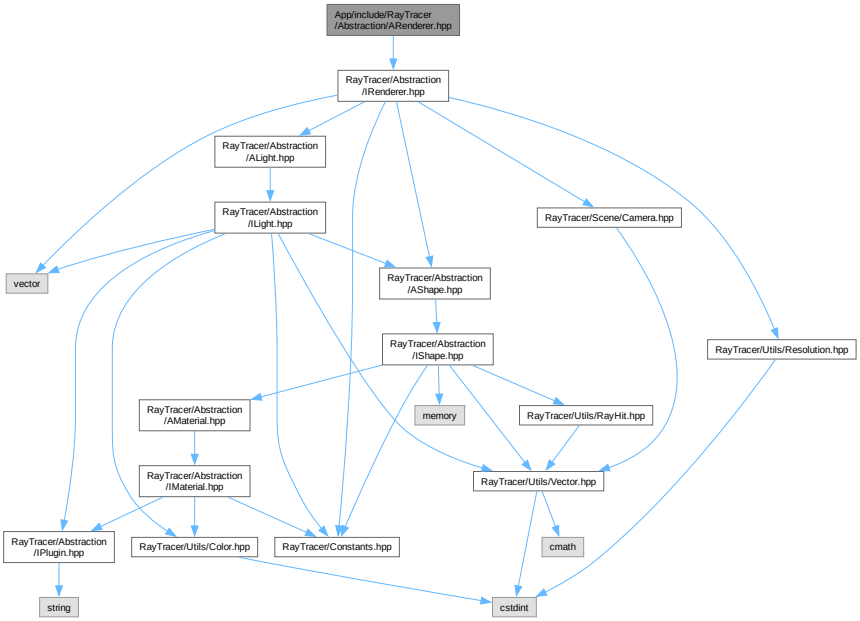
Go to the documentation of this file.

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** raytracer
00004 ** File description:
00005 ** AMaterial.hpp
00006 */
00007
00009 #ifndef RAYTRACER_AMATERIAL_HPP
00010 #define RAYTRACER_AMATERIAL_HPP
00011
00012 #include "RayTracer/Abstraction/IMaterial.hpp"
00013
00014 namespace rtr {
00015
00018     class AMaterial : public IMaterial {
00019
00020         public:
00021
00022             ~AMaterial() override = default;
00023
00024             void setType(const MaterialType &type) override { m_type = type; };
00025             void setReflectivity(const float &reflectivity) override { m_reflectivity = reflectivity;
      };
00026             void setTransparency(const float &transparency) override { m_transparency = transparency;
      };
00027
00028             [[nodiscard]] const MaterialType& getType() const override { return m_type; };
00029             [[nodiscard]] Color& getColor() override { return m_color; };
00030             [[nodiscard]] const float& getReflectivity() const override { return m_reflectivity; };
00031             [[nodiscard]] const float& getTransparency() const override { return m_transparency; };
00032
00033         private:
00034
00036             MaterialType m_type{MaterialType::NONE};
00037
00039             Color m_color{Color::getBlack()};
00040
00042             float m_reflectivity{0.0F};
00043
00045             float m_transparency{0.0F};
00046
00047     }; // class AMaterial
00048
00049 }; // namespace
00050
00051 #endif //RAYTRACER_AMATERIAL_HPP
```

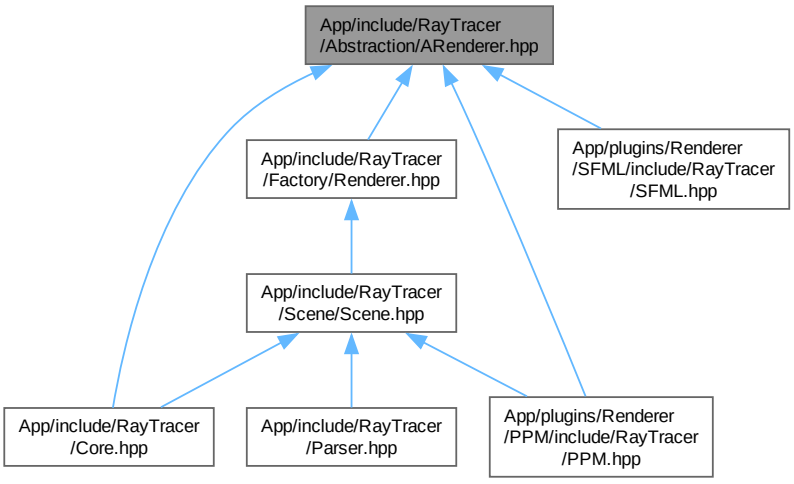
5.5 App/include/RayTracer/Abstraction/ARenderer.hpp File Reference

```
#include "RayTracer/Abstraction/IRenderer.hpp"
```

Include dependency graph for ARenderer.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class rtr::ARenderer

  An abstract class for renderers, based on the interface IRenderer.

5.6 ARenderer.hpp

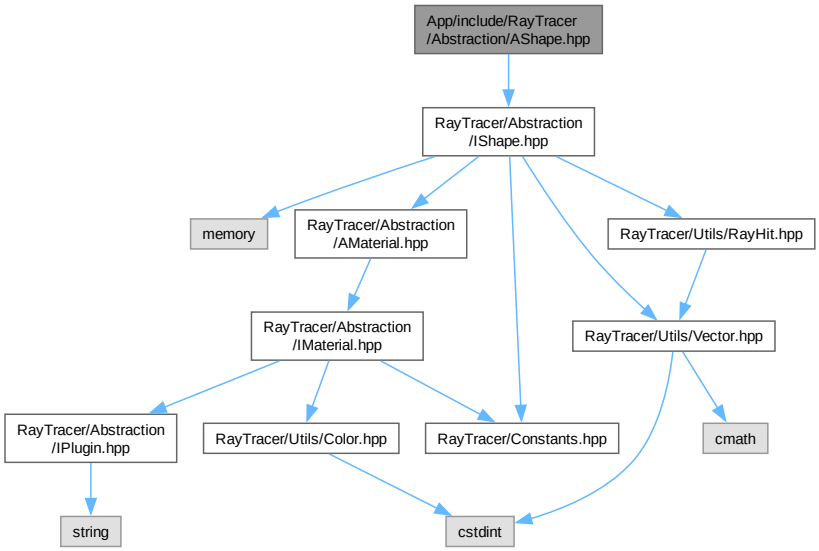
Go to the documentation of this file.

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** Raytracer
00004 ** File description:
00005 ** ARenderer.cpp
00006 */
00007
00009 #ifndef RAYTRACER_ARENDERER_HPP
00010 #define RAYTRACER_ARENDERER_HPP
00011
00012 #include "RayTracer/Abstraction/IRenderer.hpp"
00013
00014 namespace rtr {
00015
00018     class ARenderer : public IRenderer {
00019
00020         public:
00021
00022             ~ARenderer() override = default;
00023
00024             void setType(const RendererType &rendererType) override { m_type = rendererType; };
00025             void setName(const std::string &name) override { m_name = name; };
00026             void setPixels(const std::vector<std::vector<rtr::Color>& pixels) override { m_pixels =
      pixels; };
00027
00028             [[nodiscard]] const RendererType& getType() const override { return m_type; };
00029             [[nodiscard]] Resolution& getResolution() override { return m_resolution; };
00030             [[nodiscard]] Color& getBackgroundColor() override { return m_backgroundColor; };
00031             [[nodiscard]] const std::string& getName() const override { return m_name; };
00032             [[nodiscard]] std::vector<std::vector<rtr::Color>& getPixels() override { return m_pixels;
      };
00033
00034
00035         private:
00036
00038             RendererType m_type{RendererType::NONE};
00039
00041             Resolution m_resolution{1920, 1080};
00042
00044             std::string m_name{"Default Renderer Name"};
00045
00047             Color m_backgroundColor{Color::getBlack()};
00048
00050             std::vector<std::vector<rtr::Color> m_pixels;
00051
00052     }; // class ARenderer
00053
00054 } // namespace RayTracer
00055
00056 #endif //RAYTRACER_ARENDERER_HPP
```

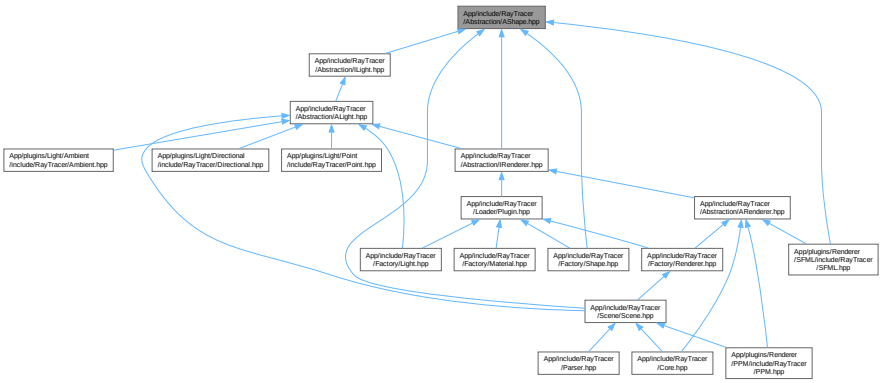
5.7 App/include/RayTracer/Abstraction/AShape.hpp File Reference

```
#include "RayTracer/Abstraction/IShape.hpp"
```

Include dependency graph for AShape.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class rtr::AShape

  An abstract class for shapes, based on the interface IShape.

5.8 AShape.hpp

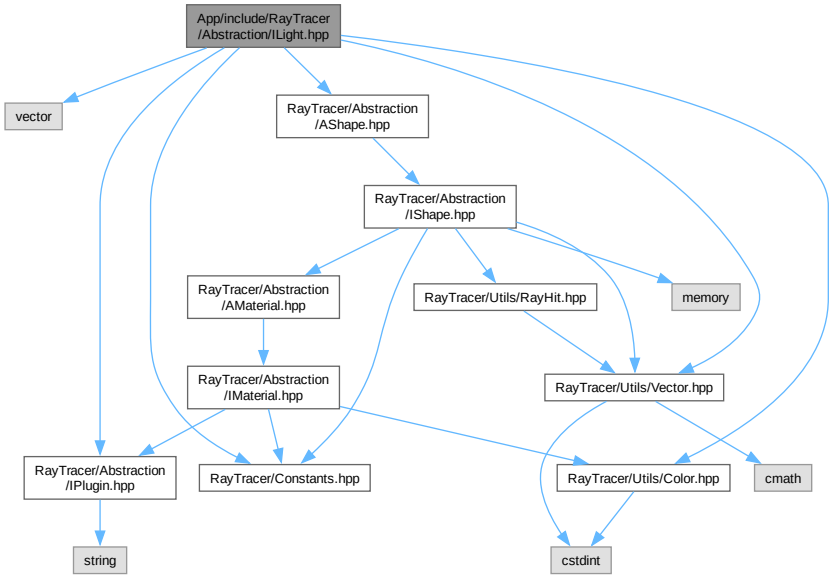
Go to the documentation of this file.

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** Raytracer
00004 ** File description:
00005 ** AShape.cpp
00006 */
00007
00009 #ifndef RAYTRACER_ASHAPE_HPP
00010 #define RAYTRACER_ASHAPE_HPP
00011
00012 #include "RayTracer/Abstraction/IShape.hpp"
00013
00014 namespace rtr {
00015
00018     class AShape : public IShape {
00019
00020         public:
00021
00022             ~AShape() override = default;
00023
00024             void setType(const ShapeType &type) override { m_type = type; };
00025             void setRadius(const double& radius) override { m_radius = radius; };
00026             void setHeight(const double& height) override { m_height = height; };
00027             void setMaterial(std::unique_ptr<AMaterial> material) override { m_material =
      std::move(material); };
00028
00029             [[nodiscard]] const ShapeType& getType() const override { return m_type; };
00030             [[nodiscard]] AMaterial& getMaterial() override { return *m_material; };
00031             [[nodiscard]] Vector& getPosition() override { return m_position; };
00032             [[nodiscard]] Vector& getNormal() override { return m_normal; };
00033             [[nodiscard]] Vector& getRotation() override { return m_rotation; };
00034             [[nodiscard]] const double &getRadius() const override { return m_radius; };
00035             [[nodiscard]] const double &getHeight() const override { return m_height; };
00036             [[nodiscard]] Vector getDistance(const Vector& point) override { return point -
      m_position; };
00037
00038         private:
00039
00041             ShapeType m_type{ShapeType::NONE};
00042
00044             std::unique_ptr<AMaterial> m_material;
00045
00047             Vector m_position{0, 0, 0};
00048
00050             Vector m_normal{0, 0, 0};
00051
00053             Vector m_rotation{0, 0, 0};
00054
00056             double m_radius{0};
00057
00059             double m_height{0};
00060
00061     }; // class AShape
00062
00063 } // namespace RayTracer
00064
00065 #endif //RAYTRACER_ASHAPE_HPP
```

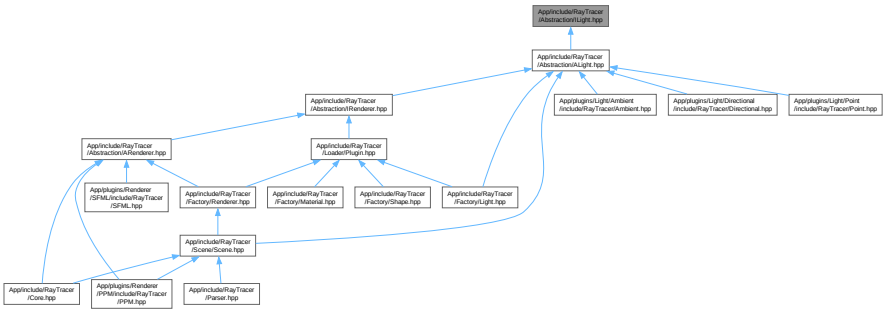
5.9 App/include/RayTracer/Abstraction/ILight.hpp File Reference

```
#include <vector>
#include "RayTracer/Abstraction/IPlugin.hpp"
#include "RayTracer/Abstraction/AShape.hpp"
#include "RayTracer/Constants.hpp"
#include "RayTracer/Utils/Vector.hpp"
#include "RayTracer/Utils/Color.hpp"
```

Include dependency graph for ILight.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class rtr::ILight

  An interface for lights.

5.10 ILight.hpp

Go to the documentation of this file.

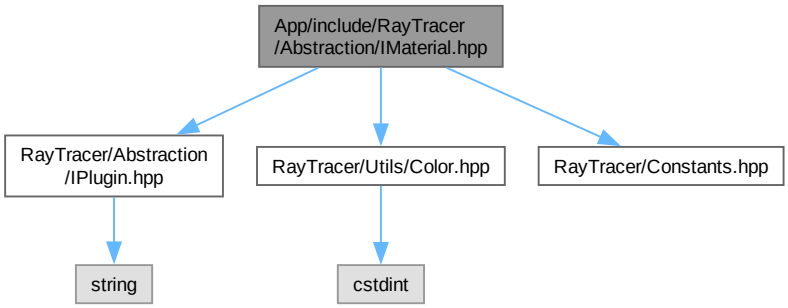
```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** Raytracer
00004 ** File description:
00005 ** ILight.cpp
00006 */
00007
00009 #ifndef RAYTRACER_ILIGHT_HPP
```

```
00010 #define RAYTRACER_ILIGHT_HPP
00011
00012 #include <vector>
00013
00014 #include "RayTracer/Abstraction/IPlugin.hpp"
00015 #include "RayTracer/Abstraction/AShape.hpp"
00016 #include "RayTracer/Constants.hpp"
00017 #include "RayTracer/Utils/Vector.hpp"
00018 #include "RayTracer/Utils/Color.hpp"
00019
00020 namespace rtr {
00021
00024     class ILight : public IPlugin {
00025
00026         public:
00027
00030             virtual void setType(const LightType &type) = 0;
00031
00033             virtual void setIntensity(const float &intensity) = 0;
00034
00039             virtual Color LightColor(const Vector &normal, const Color &col) = 0;
00040
00042             [[nodiscard]] virtual const LightType& getType() const = 0;
00043
00045             [[nodiscard]] virtual Vector& getPosition() = 0;
00046
00048             [[nodiscard]] virtual Vector& getDirection() = 0;
00049
00051             [[nodiscard]] virtual Color& getColor() = 0;
00052
00054             [[nodiscard]] virtual float& getIntensity() = 0;
00055
00056     }; // class ILight
00057
00058 } // namespace RayTracer
00059
00060 #endif //RAYTRACER_ILIGHT_HPP
```

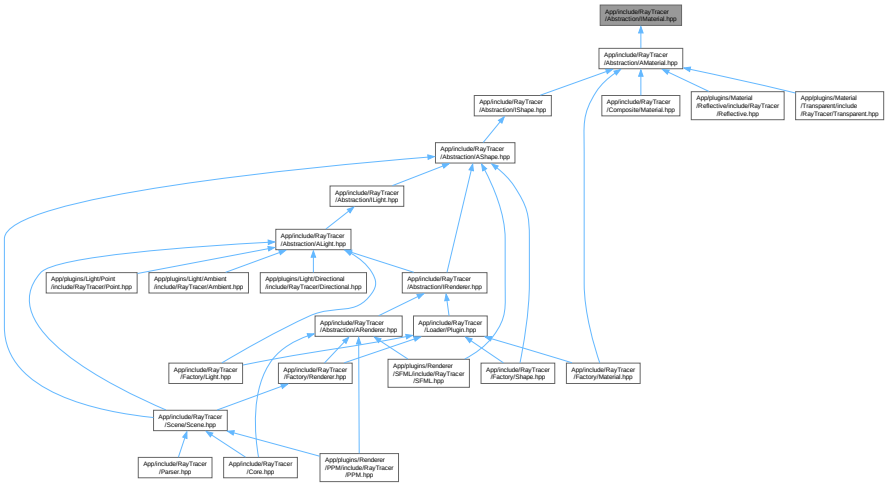
5.11 App/include/RayTracer/Abstraction/IMaterial.hpp File Reference

```
#include "RayTracer/Abstraction/IPlugin.hpp"
#include "RayTracer/Utils/Color.hpp"
#include "RayTracer/Constants.hpp"
```

Include dependency graph for IMaterial.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class rtr::IMaterial

  An interface for materials.

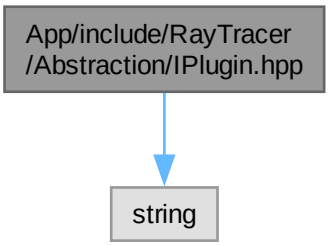
5.12 IMaterial.hpp

Go to the documentation of this file.

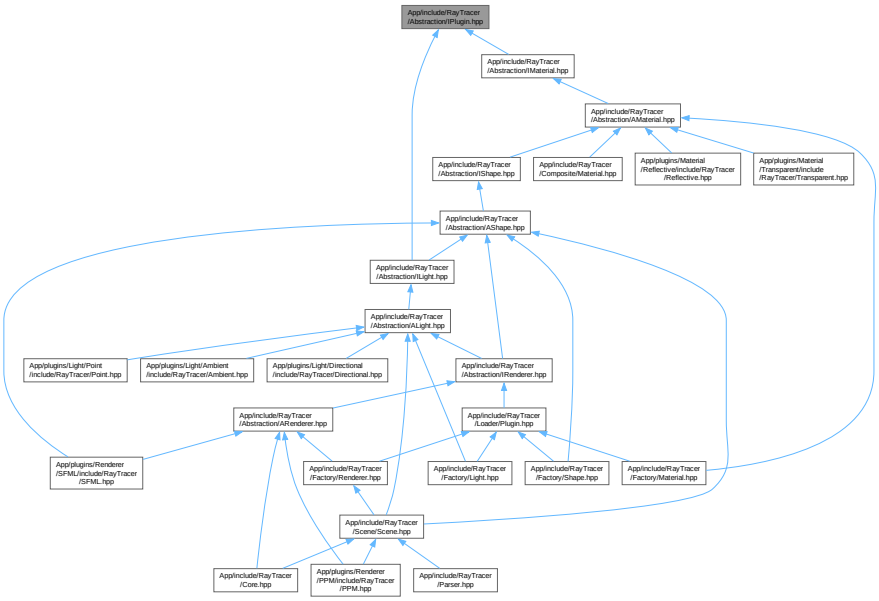
```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** raytracer
00004 ** File description:
00005 ** IMaterial
00006 */
00007
00009 #ifndef RAYTRACER_IMATERIAL_HPP
00010 #define RAYTRACER_IMATERIAL_HPP
00011
00012 #include "RayTracer/Abstraction/IPlugin.hpp"
00013 #include "RayTracer/Utils/Color.hpp"
00014 #include "RayTracer/Constants.hpp"
00015
00016 namespace rtr{
00017
00020     class IMaterial : public IPlugin {
00021
00022         public:
00023
00025             virtual void applyMaterial(Color* color) = 0;
00026
00028             virtual void setType(const MaterialType &type) = 0;
00029
00032             virtual void setReflectivity(const float &reflectivity) = 0;
00033
00036             virtual void setTransparency(const float &transparency) = 0;
00037
00039             [[nodiscard]] virtual const MaterialType& getType() const = 0;
00040
00042             [[nodiscard]] virtual Color& getColor() = 0;
00043
00045             [[nodiscard]] virtual const float& getReflectivity() const = 0;
00046
00048             [[nodiscard]] virtual const float& getTransparency() const = 0;
00049
00050     }; // class IMaterial
00051
00052 }; // namespace RayTracer
00053
00054 #endif //RAYTRACER_IMATERIAL_HPP
```

5.13 App/include/RayTracer/Abstraction/IPlugin.hpp File Reference

`#include <string>`
Include dependency graph for IPlugin.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class rtr::IPlugin

  An interface for plugins.

5.14 IPlugin.hpp

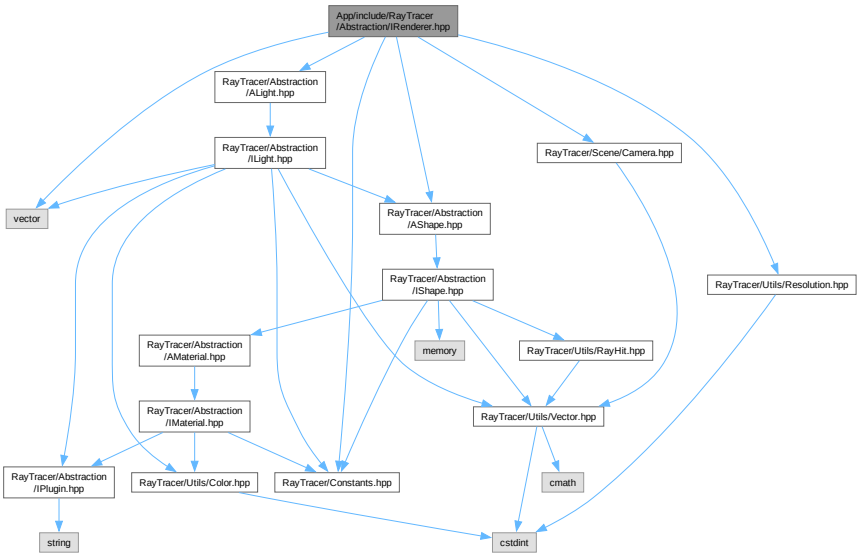
Go to the documentation of this file.

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** raytracer
00004 ** File description:
00005 ** IPlugin
00006 */
00007
00009 #ifndef RAYTRACER_IPLUGIN_HPP
00010 #define RAYTRACER_IPLUGIN_HPP
00011
00012 #include <string>
00013
00014 namespace rtr {
00015
00018     class IPlugin {
00019
00020         public:
00021
00022             virtual ~IPlugin() = default;
00023
00026             [[nodiscard]] virtual std::string getPluginName() const = 0;
00027
00028     }; // class IPlugin
00029
00030 } // namespace RayTracer
00031
00032 #endif //RAYTRACER_IPLUGIN_HPP
```

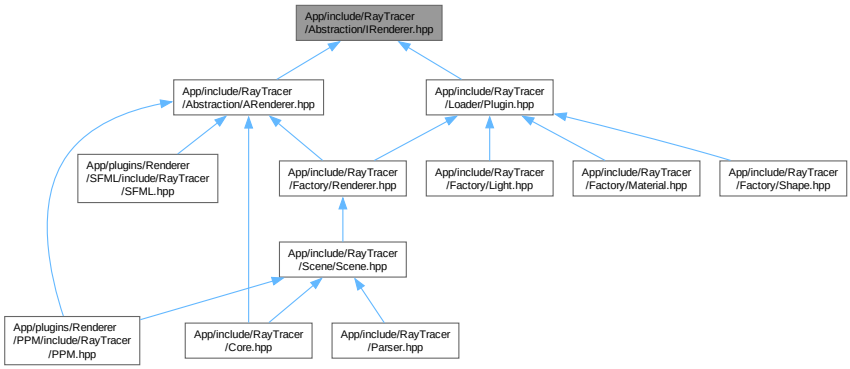
5.15 App/include/RayTracer/Abstraction/IRenderer.hpp File Reference

```
#include <vector>
#include "RayTracer/Abstraction/AShape.hpp"
#include "RayTracer/Abstraction/ALight.hpp"
#include "RayTracer/Scene/Camera.hpp"
#include "RayTracer/Utils/Resolution.hpp"
#include "RayTracer/Constants.hpp"
```

Include dependency graph for IRenderer.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class rtr::IRenderer

  An interface for renderers.

5.16 IRenderer.hpp

Go to the documentation of this file.

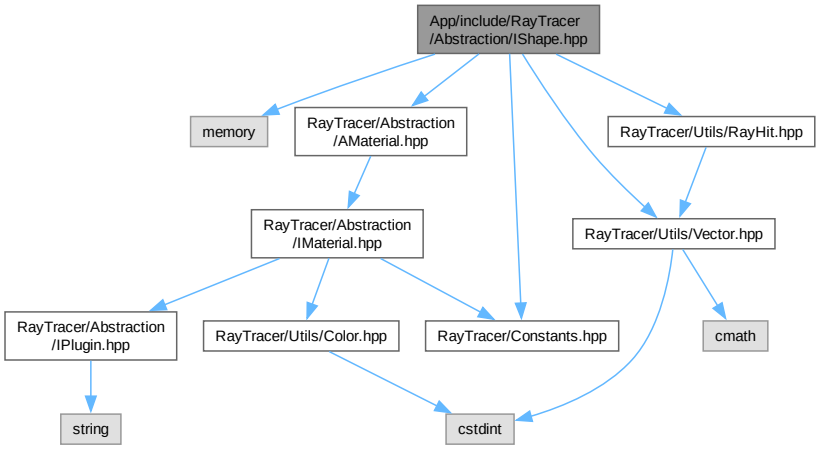
```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** Raytracer
00004 ** File description:
00005 ** IRenderer.cpp
00006 */
00007
00009 #ifndef RAYTRACER_IRENDERER_HPP
00010 #define RAYTRACER_IRENDERER_HPP
00011
00012 #include <vector>
00013
00014 #include "RayTracer/Abstraction/AShape.hpp"
00015 #include "RayTracer/Abstraction/ALight.hpp"
00016 #include "RayTracer/Scene/Camera.hpp"
00017 #include "RayTracer/Utils/Resolution.hpp"
00018 #include "RayTracer/Constants.hpp"
00019
00020 namespace rtr {
00021
00024     class IRenderer : public IPlugin {
00025
00026         public:
00027
00032             virtual void render(const std::vector<std::unique_ptr<AShape>> &shapes, const
      std::vector<std::unique_ptr<ALight>> &lights, const Camera &camera) = 0;
00033
00036             virtual void setType(const RendererType &rendererType) = 0;
00037
00040             virtual void setName(const std::string &name) = 0;
00041
00044             virtual void setPixels(const std::vector<std::vector<rtr::Color>& pixels) = 0;
00045
00048             [[nodiscard]] virtual const RendererType& getType() const = 0;
00049
00052             [[nodiscard]] virtual const std::string& getName() const = 0;
00053
00056             [[nodiscard]] virtual Resolution& getResolution() = 0;
00057
00060             [[nodiscard]] virtual Color& getBackgroundColor() = 0;
00061
00064             [[nodiscard]] virtual std::vector<std::vector<rtr::Color>& getPixels() = 0;
```

```
00065
00066     }; // IRenderer
00067
00068 } // namespace RayTracer
00069
00070 #endif //RAYTRACER_IRENDERER_HPP
```

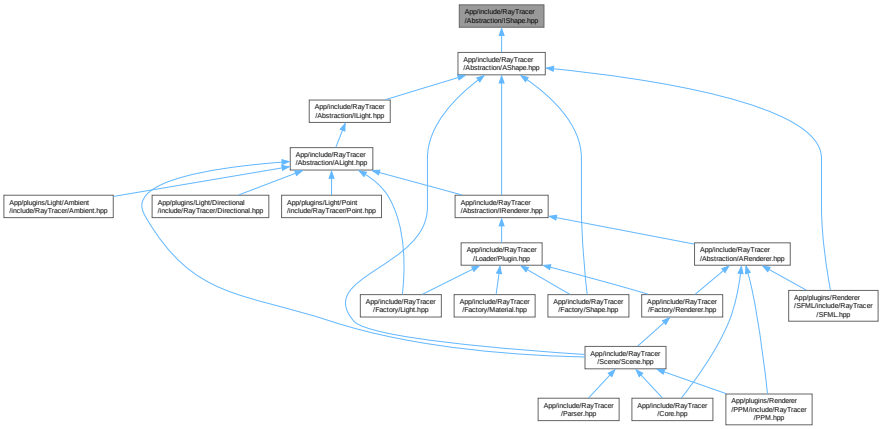
5.17 App/include/RayTracer/Abstraction/IShape.hpp File Reference

```
#include <memory>
#include "RayTracer/Abstraction/AMaterial.hpp"
#include "RayTracer/Constants.hpp"
#include "RayTracer/Utils/Vector.hpp"
#include "RayTracer/Utils/RayHit.hpp"
```

Include dependency graph for IShape.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class rtr::IShape

  An interface used to get the shape's parameters based on the configuration file.

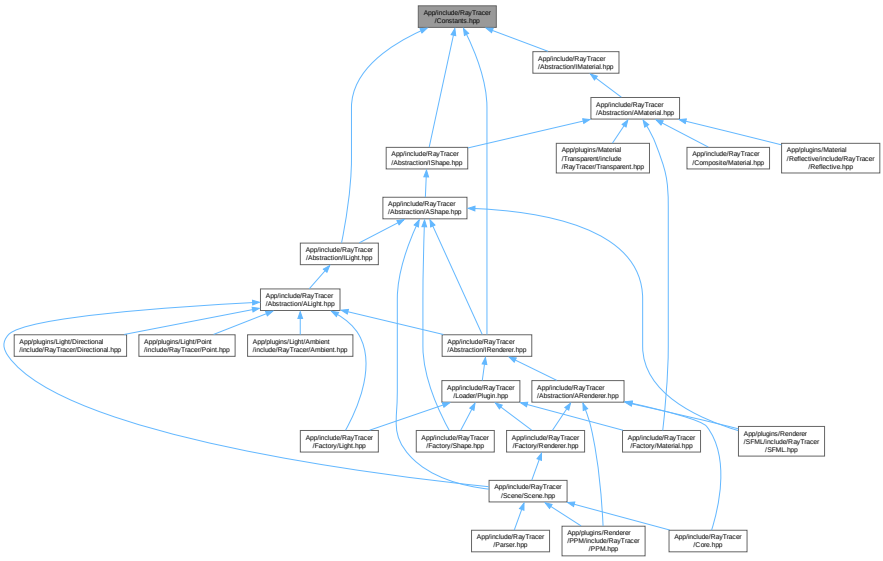
5.18 IShape.hpp

Go to the documentation of this file.

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** Raytracer
00004 ** File description:
00005 ** IShape.cpp
00006 */
00007
00009 #ifndef RAYTRACER_ISHAPE_HPP
00010 #define RAYTRACER_ISHAPE_HPP
00011
00012 #include <memory>
00013
00014 #include "RayTracer/Abstraction/AMaterial.hpp"
00015 #include "RayTracer/Constants.hpp"
00016 #include "RayTracer/Utils/Vector.hpp"
00017 #include "RayTracer/Utils/RayHit.hpp"
00018
00019 namespace rtr {
00020
00023     class IShape : public IPlugin {
00024
00025         public:
00026
00029             virtual void setType(const ShapeType& type) = 0;
00030
00033             virtual void setMaterial(std::unique_ptr<AMaterial> material) = 0;
00034
00037             virtual void setRadius(const double& radius) = 0;
00038
00041             virtual void setHeight(const double& height) = 0;
00042
00045             [[nodiscard]] virtual const ShapeType& getType() const = 0;
00046
00049             [[nodiscard]] virtual AMaterial& getMaterial() = 0;
00050
00053             [[nodiscard]] virtual Vector& getPosition() = 0;
00054
00057             [[nodiscard]] virtual Vector& getNormal() = 0;
00058
00061             [[nodiscard]] virtual Vector& getRotation() = 0;
00062
00065             [[nodiscard]] virtual const double &getRadius() const = 0;
00066
00069             [[nodiscard]] virtual const double &getHeight() const = 0;
00070
00075             [[nodiscard]] virtual bool hits(std::pair<Vector, Vector> ray, RayHit &hit) = 0;
00076
00080             [[nodiscard]] virtual Vector getDistance(const Vector& point) = 0;
00081
00082     }; // IShape
00083
00084 } // namespace RayTracer
00085
00086 #endif //RAYTRACER_ISHAPE_HPP
```

5.19 App/include/RayTracer/Constants.hpp File Reference

This graph shows which files directly or indirectly include this file:



Enumerations

- enum class **RendererType** { **PPM** , **SFML** , **NONE** }
- enum class **ShapeType** {
  SPHERE , **PLANE** , **CYLINDER** , **CONE** ,
  NONE }
- enum class **LightType** { **AMBIENT** , **DIRECTIONAL** , **POINT** , **NONE** }
- enum class **MaterialType** { **TRANSPARENT** , **REFLECTIVE** , **NONE** }

5.20 Constants.hpp

Go to the documentation of this file.

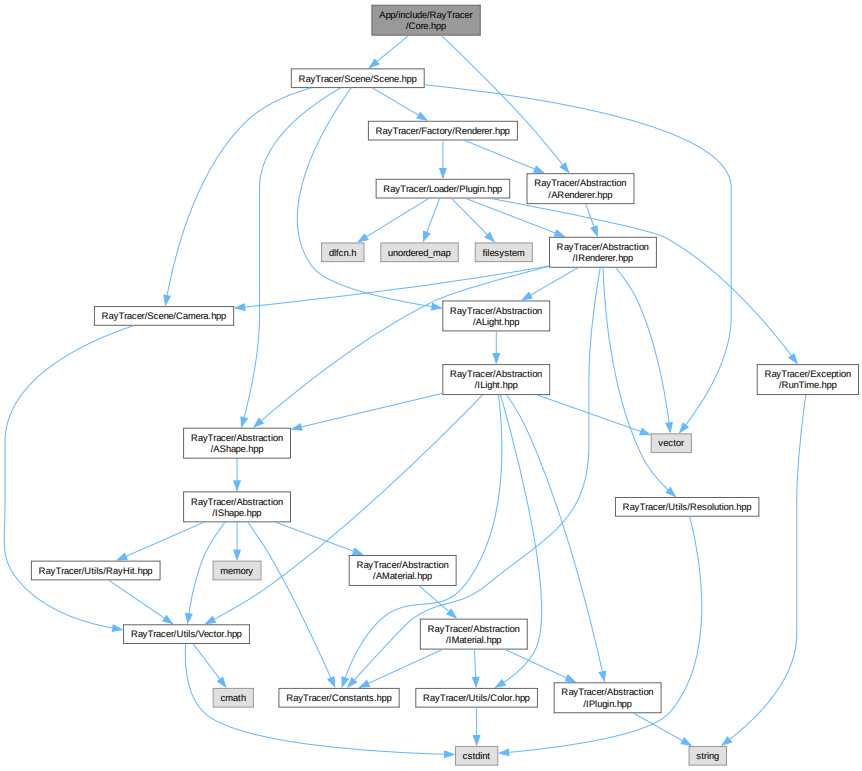
```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** Raytracer
00004 ** File description:
00005 ** Constants.hpp
00006 */
00007
00009 #ifndef RAYTRACER_CONSTANTS_HPP
00010 #define RAYTRACER_CONSTANTS_HPP
00011
00012 static constexpr int EPITECH_ERROR = 84;
00013 static constexpr int SUCCESS = 0;
00014 static constexpr int ERROR = 1;
00015
00016 static constexpr double EPSILON = 1e-6;
00017
00018 static constexpr const char * AMBIENT_LIGHT = "ambient";
00019 static constexpr const char * DIRECTIONAL_LIGHT = "directional";
00020 static constexpr const char * POINT_LIGHT = "point";
00021
00022 static constexpr const char * COMPOSITE_MATERIAL = "materialComposite";
00023 static constexpr const char * TRANSPARENT_MATERIAL = "transparent";
00024 static constexpr const char * REFLECTIVE_MATERIAL = "reflective";
00025
00026 static constexpr const char * PPM_RENDERER = "ppm";
```

```
00027 static constexpr const char * SFML_RENDERER = "sfml";
00028
00029 static constexpr const char * SPHERE_SHAPE = "sphere";
00030 static constexpr const char * PLANE_SHAPE = "plane";
00031 static constexpr const char * CYLINDER_SHAPE = "cylinder";
00032 static constexpr const char * CONE_SHAPE = "cone";
00033
00034 namespace rtr {
00035
00037     enum class RendererType {
00038         PPM,
00039         SFML,
00040         NONE
00041     };
00042
00044     enum class ShapeType {
00045         SPHERE,
00046         PLANE,
00047         CYLINDER,
00048         CONE,
00049         NONE
00050     };
00051
00053     enum class LightType {
00054         AMBIENT,
00055         DIRECTIONAL,
00056         POINT,
00057         NONE
00058     };
00059
00061     enum class MaterialType {
00062         TRANSPARENT,
00063         REFLECTIVE,
00064         NONE
00065     };
00066
00067 }
00068
00069 #endif //RAYTRACER_CONSTANTS_HPP
```

5.21 App/include/RayTracer/Core.hpp File Reference

```
#include "RayTracer/Abstraction/ARenderer.hpp"
#include "RayTracer/Scene/Scene.hpp"
```

Include dependency graph for Core.hpp:



Classes

- class rtr::Core

  A class representing the core functionality of the ray tracer.
- class rtr::Core::CoreException

  An exception class for core errors.

5.22 Core.hpp

Go to the documentation of this file.

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** Raytracer
00004 ** File description:
00005 ** Core.hpp
00006 */
00007
00009 #ifndef RAYTRACER_CORE_HPP
00010 #define RAYTRACER_CORE_HPP
00011
00012 #include "RayTracer/Abstraction/ARenderer.hpp"
00013 #include "RayTracer/Scene/Scene.hpp"
00014
00015 namespace rtr {
00016
00019     class Core {
00021         public:
00022             Core() = default;
00023
00024             ~Core() = default;
00025
```

```
00028             static void runRayTracer(Scene &scene);
00029
00032             class CoreException : public std::exception
00033             {
00034             public:
00035                 explicit CoreException(std::string msg) : m_msg{std::move(msg)} {}
00036
00037                 ~CoreException() override = default;
00038
00039                 CoreException(const CoreException &) = delete;
00040                 CoreException &operator=(const CoreException &) = delete;
00041
00042
00043                 CoreException(const CoreException &&) = delete;
00044                 CoreException &operator=(const CoreException &&) = delete;
00045
00047                 [[nodiscard]] const char *what() const noexcept override { return m_msg.c_str(); };
00048
00049             private:
00051                 std::string m_msg{0};
00052
00053             }; // CoreException
00054
00055     }; // Core
00056
00057 } // namespace RayTracer
00058
00059 #endif //RAYTRACER_CORE_HPP
```

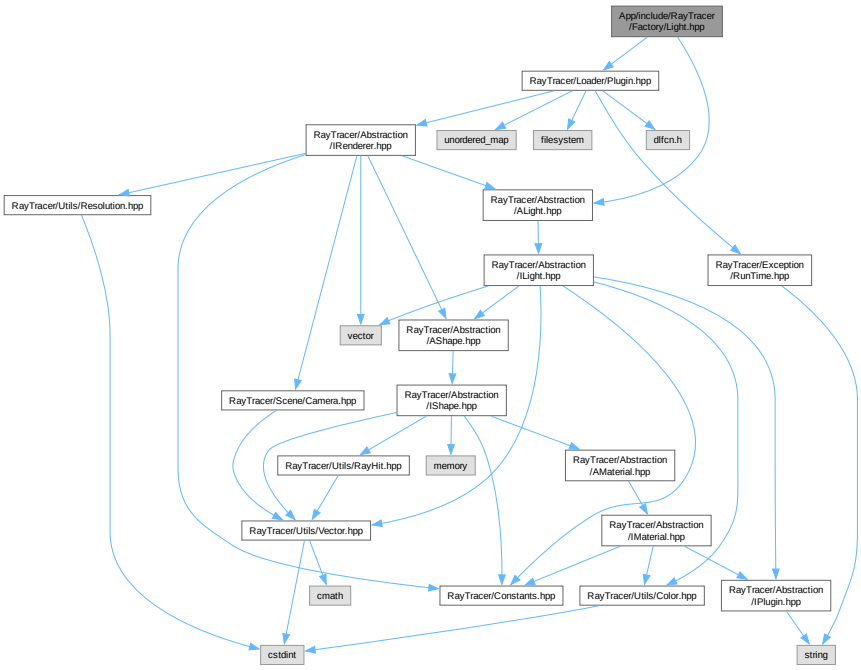
5.23 RunTime.hpp

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** Raytracer | Exceptions
00004 ** File description:
00005 ** RunTime.hpp
00006 */
00007
00008 #ifndef RAYTRACER_RUNTIME_EXCEPTION_HPP
00009 #define RAYTRACER_RUNTIME_EXCEPTION_HPP
00010
00011 #include <string>
00012
00013 namespace rtr {
00014
00015     class RunTimeException : public std::exception
00016     {
00017         public:
00018
00019             explicit RunTimeException(std::string msg) : m_msg{std::move(msg)} {};
00020             ~RunTimeException() override = default;
00021
00022             RunTimeException(const RunTimeException &) = delete;
00023             RunTimeException &operator=(const RunTimeException &) = delete;
00024             RunTimeException(const RunTimeException &&) = delete;
00025             RunTimeException &operator=(const RunTimeException &&) = delete;
00026
00027             [[nodiscard]] const char *what() const noexcept override { return m_msg.c_str(); };
00028
00029         private:
00030
00031             std::string m_msg{0};
00032
00033     }; // class RunTimeException
00034
00035 } // namespace RayTracer
00036
00037 #endif // RAYTRACER_RUNTIME_EXCEPTION_HPP
```

5.24 App/include/RayTracer/Factory/Light.hpp File Reference

```
#include "RayTracer/Abstraction/ALight.hpp"
#include "RayTracer/Loader/Plugin.hpp"
```

Include dependency graph for Light.hpp:



Classes

- class rtr::LightFactory

  A factory class for the lights.

5.25 Light.hpp

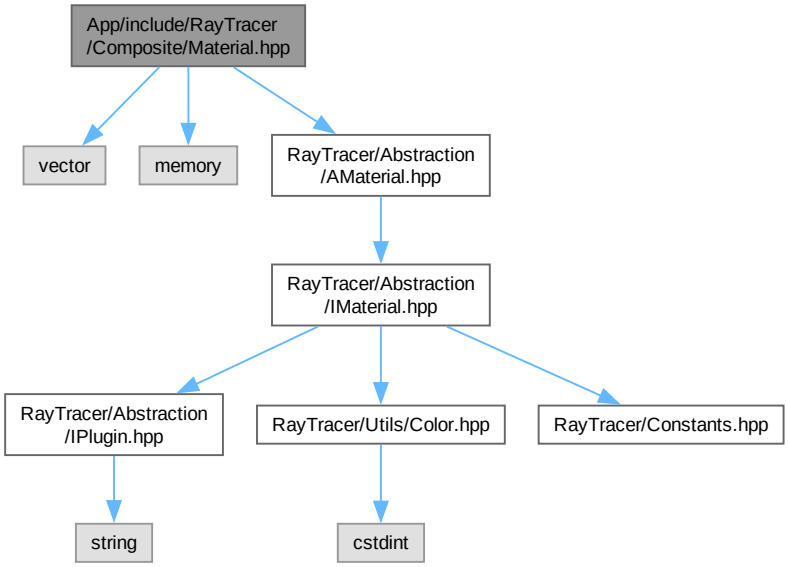
Go to the documentation of this file.

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** Raytracer | Factory
00004 ** File description:
00005 ** Light.hpp
00006 */
00007
00009 #ifndef RAYTRACER_LIGHT_FACTORY_HPP
00010 #define RAYTRACER_LIGHT_FACTORY_HPP
00011
00012 #include "RayTracer/Abstraction/ALight.hpp"
00013 #include "RayTracer/Loader/Plugin.hpp"
00014
00015 namespace rtr {
00016
00019     class LightFactory {
00020
00021         public:
00026             static std::unique_ptr<ALight> createLight(const Color &color,
00027                                                        const float &intensity);
00028
00035             static std::unique_ptr<ALight> createLight(const LightType &type,
00036                                                        const Color &color,
00037                                                        const float &intensity,
00038                                                        const Vector &vector);
00039
00040     }; // class LightFactory
00041
00042 } // namespace RayTracer
00043
00044 #endif //RAYTRACER_LIGHT_FACTORY_HPP
```

5.26 App/include/RayTracer/Composite/Material.hpp File Reference

```
#include <vector>
#include <memory>
#include "RayTracer/Abstraction/AMaterial.hpp"
```

Include dependency graph for Material.hpp:



Classes

- class rtr::CompositeMaterial

  A class to create a composite material.

5.27 Material.hpp

Go to the documentation of this file.

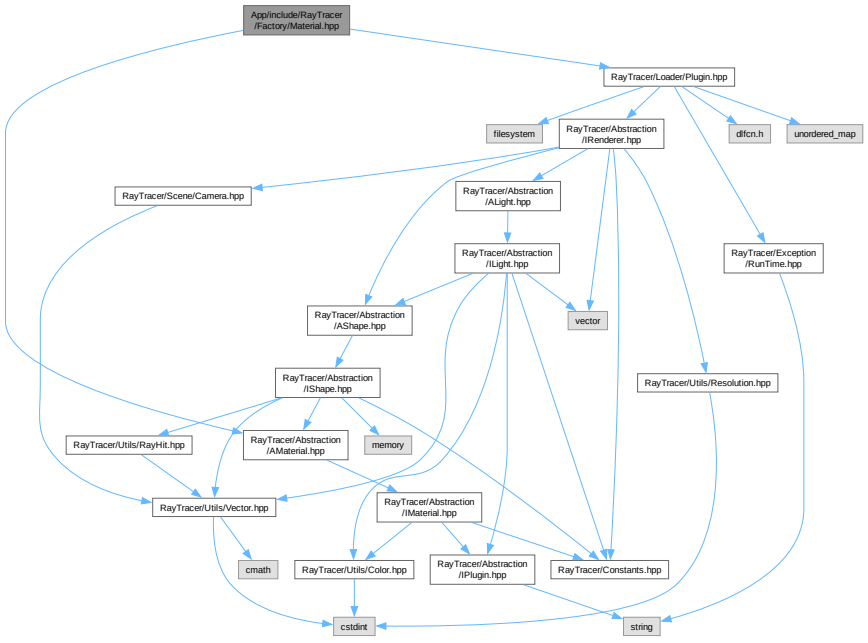
```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** raytracer | Composite
00004 ** File description:
00005 ** Material.hpp
00006 */
00007
00009 #ifndef RAYTRACER_COMPOSITE_MATERIAL_HPP
00010 #define RAYTRACER_COMPOSITE_MATERIAL_HPP
00011
00012 #include <vector>
00013 #include <memory>
00014
00015 #include "RayTracer/Abstraction/AMaterial.hpp"
00016
00017 namespace rtr {
00018
00021     class CompositeMaterial : public AMaterial {
00022
00023         public:
00024
```

```
00025             ~CompositeMaterial() override = default;
00026
00027             [[nodiscard]] std::string getPluginName() const override { return COMPOSITE_MATERIAL; };
00028
00031             void addMaterial(std::unique_ptr<AMaterial> material) {
      m_materials.emplace_back(std::move(material)); };
00032
00033             void applyMaterial(Color* color) override {
00034                 for (const auto& material : m_materials) {
00035                     material->applyMaterial(color);
00036                 }
00037                 this->getColor().setColor(color->getValue());
00038             }
00039
00040         private:
00041
00043             std::vector<std::unique_ptr<AMaterial» m_materials;
00044
00045     }; // class CompositeMaterial
00046
00047 }; // namespace RayTracer
00048
00049 #endif //RAYTRACER_COMPOSITE_MATERIAL_HPP
```

5.28 App/include/RayTracer/Factory/Material.hpp File Reference

```
#include "RayTracer/Abstraction/AMaterial.hpp"
#include "RayTracer/Loader/Plugin.hpp"
```

Include dependency graph for Material.hpp:



Classes

- class rtr::MaterialFactory

  A factory class for the materials of the shapes.

5.29 Material.hpp

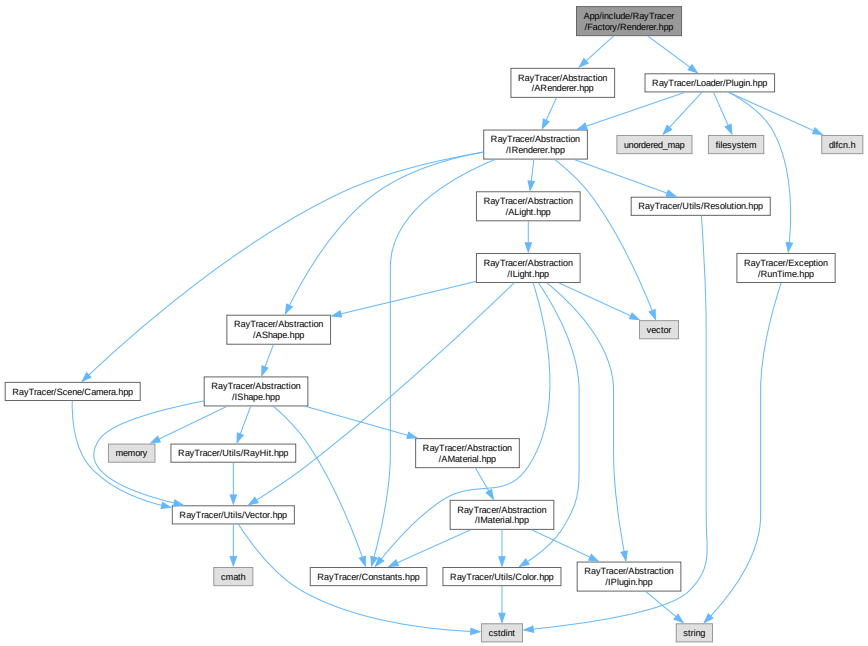
Go to the documentation of this file.

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** raytracer | Factory
00004 ** File description:
00005 ** Material.hpp
00006 */
00007
00009 #ifndef RAYTRACER_MATERIAL_FACTORY_HPP
00010 #define RAYTRACER_MATERIAL_FACTORY_HPP
00011
00012 #include "RayTracer/Abstraction/AMaterial.hpp"
00013 #include "RayTracer/Loader/Plugin.hpp"
00014
00015 namespace rtr {
00016
00019     class MaterialFactory {
00020
00021         public:
00022
00027             static std::unique_ptr<AMaterial> createMaterial(const MaterialType &type,
00028                                                              const float &floatValue);
00029
00030     }; // class MaterialFactory
00031
00032 }; // namespace RayTracer
00033
00034 #endif //RAYTRACER_MATERIAL_FACTORY_HPP
```

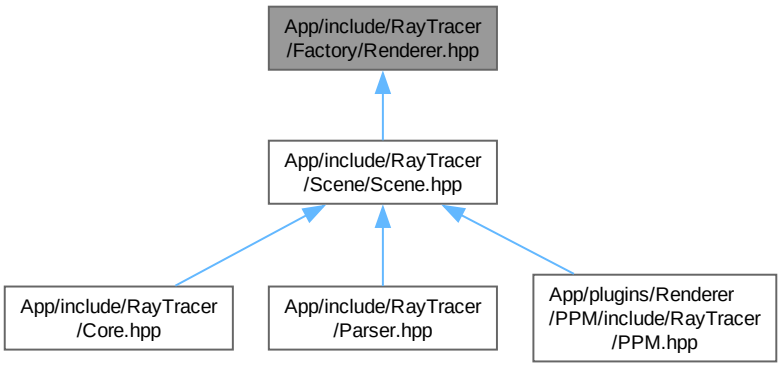
5.30 App/include/RayTracer/Factory/Renderer.hpp File Reference

```
#include "RayTracer/Abstraction/ARenderer.hpp"
#include "RayTracer/Loader/Plugin.hpp"
```

Include dependency graph for Renderer.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class rtr::RendererFactory

  A factory class to create the renderers.

5.31 Renderer.hpp

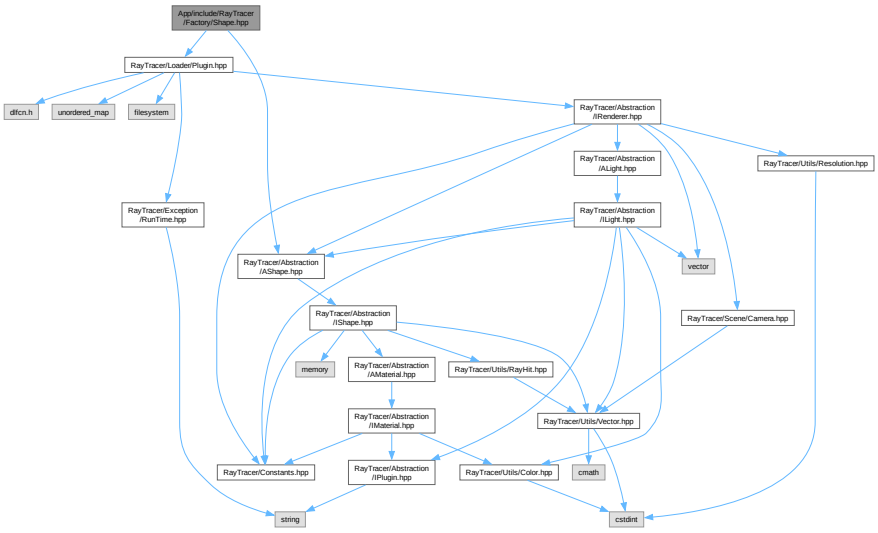
Go to the documentation of this file.

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** Raytracer | Factory
00004 ** File description:
00005 ** Renderer.hpp
00006 */
00007
00009 #ifndef RAYTRACER_RENDERER_FACTORY_HPP
00010 #define RAYTRACER_RENDERER_FACTORY_HPP
00011
00012 #include "RayTracer/Abstraction/ARenderer.hpp"
00013 #include "RayTracer/Loader/Plugin.hpp"
00014
00015 namespace rtr {
00016
00019     class RendererFactory {
00020
00021         public:
00028             static std::unique_ptr<ARenderer> createRenderer(const RendererType &type,
00029                                                              const std::string &name,
00030                                                              const Resolution &resolution,
00031                                                              const Color &backgroundColor);
00032
00033     }; // class RendererFactory
00034
00035 } // namespace RayTracer
00036
00037 #endif //RAYTRACER_RENDERER_FACTORY_HPP
```

5.32 App/include/RayTracer/Factory/Shape.hpp File Reference

```
#include "RayTracer/Abstraction/AShape.hpp"
#include "RayTracer/Loader/Plugin.hpp"
```

Include dependency graph for Shape.hpp:



Classes

- class rtr::ShapeFactory

  A factory class for the shapes.

5.33 Shape.hpp

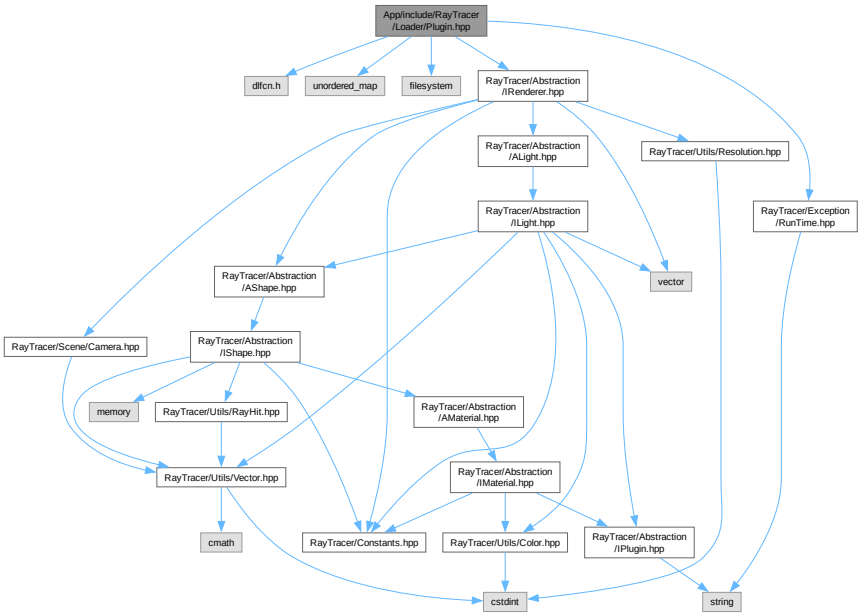
Go to the documentation of this file.

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** Raytracer | Factory
00004 ** File description:
00005 ** ShapeFactory.hpp
00006 */
00007
00009 #ifndef RAYTRACER_SHAPE_FACTORY_HPP
00010 #define RAYTRACER_SHAPE_FACTORY_HPP
00011
00012 #include "RayTracer/Abstraction/AShape.hpp"
00013 #include "RayTracer/Loader/Plugin.hpp"
00014
00015 namespace rtr {
00016
00019     class ShapeFactory {
00020
00021         public:
00022
00027             static std::unique_ptr<AShape> createShape(const Vector &position,
00028                                                        const Vector &normal);
00029
00034             static std::unique_ptr<AShape> createShape(const Vector &position,
00035                                                        const double &radius);
00036
00044             static std::unique_ptr<AShape> createShape(const ShapeType &type,
00045                                                        const Vector &position,
00046                                                        const Vector &rotation,
00047                                                        const double &radius,
00048                                                        const double &height);
00049     }; // class ShapeFactory
00050
00051 } // namespace RayTracer
00052
00053 #endif //RAYTRACER_SHAPE_FACTORY_HPP
```

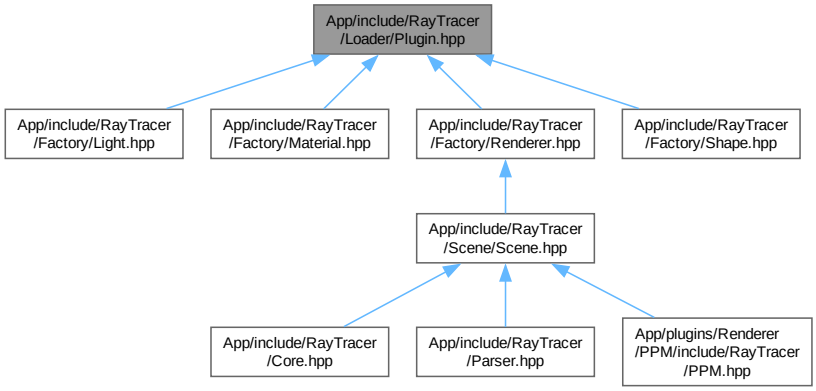
5.34 App/include/RayTracer/Loader/Plugin.hpp File Reference

```
#include <dlfcn.h>
#include <unordered_map>
#include <filesystem>
#include "RayTracer/Abstraction/IRenderer.hpp"
#include "RayTracer/Exception/RunTime.hpp"
```

Include dependency graph for Plugin.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class rtr::PluginLoader

  A class to load the plugins.

5.35 Plugin.hpp

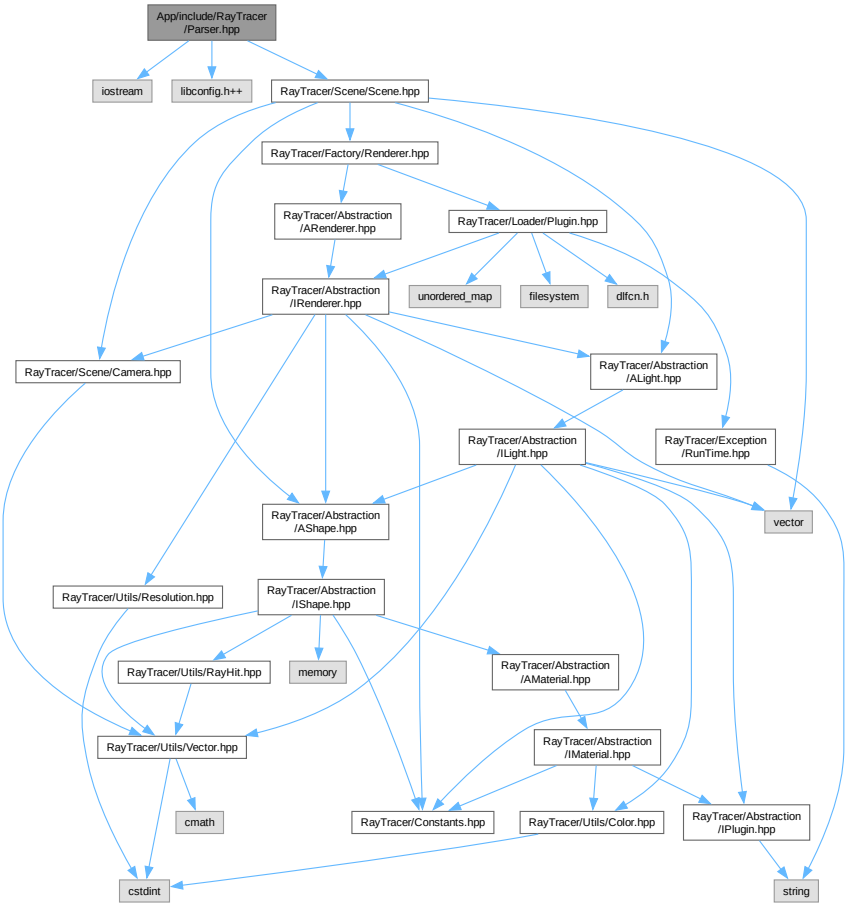
Go to the documentation of this file.

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** Raytracer | Loader
00004 ** File description:
00005 ** Plugin.hpp
00006 */
00007
00009 #ifndef RAYTRACER_PLUGIN_LOADER_HPP
00010 #define RAYTRACER_PLUGIN_LOADER_HPP
00011
00012 #include <dlfcn.h>
00013 #include <unordered_map>
00014 #include <filesystem>
00015
00016 #include "RayTracer/Abstraction/IRenderer.hpp"
00017 #include "RayTracer/Exception/RunTime.hpp"
00018
00019 namespace rtr {
00020
00023     class PluginLoader {
00024
00025         public:
00026
00027             using PluginCreator = std::unique_ptr<IPlugin> (*)();
00028
00029             ~PluginLoader() = default;
00030
00033             static PluginLoader &getInstance() {
00034                 static PluginLoader instance;
00035                 return instance;
00036             }
00037
00040             template <typename T>
00041             std::unique_ptr<T> getPlugin(const std::string &pluginName);
00042
00043         private:
00044
00045             PluginLoader() { loadPlugins(); };
00046
00048             void loadPlugins();
00049
00051             std::unordered_map<std::string, PluginCreator> m_plugins{};
00052
00053     }; // class PluginLoader
00054
00055 } // namespace RayTracer
00056
00057 #endif // RAYTRACER_PLUGIN_LOADER_HPP
```

5.36 App/include/RayTracer/Parser.hpp File Reference

```
#include <iostream>
#include <libconfig.h++>
#include "RayTracer/Scene/Scene.hpp"
```

Include dependency graph for Parser.hpp:



Classes

- class rtr::Parser

  Class dedicated to the parsing of configuration files and command-line arguments.
- class rtr::Parser::ParserException

  Exception class for errors in the parsers.

5.37 Parser.hpp

Go to the documentation of this file.

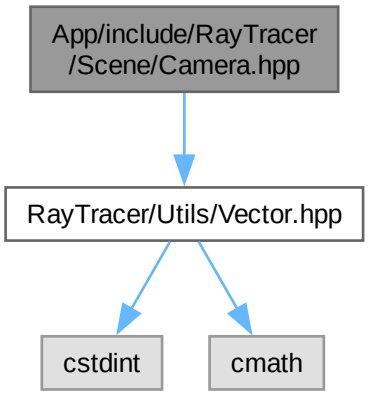
```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** Raytracer
00004 ** File description:
00005 ** Parser.cpp
00006 */
00007
00009 #ifndef RAYTRACER_PARSER_HPP
00010 #define RAYTRACER_PARSER_HPP
00011
00012 #include <iostream>
00013 #include <libconfig.h++>
00014
00015 #include "RayTracer/Scene/Scene.hpp"
```

```
00016
00017 namespace rtr {
00018
00021     class Parser {
00022
00023         public:
00024
00028             static int parseArgs(const std::string &filePath);
00029
00033             static std::unique_ptr<rtr::Scene> parseFile(const std::string &filePath);
00034
00038             static void parseRenderer(const libconfig::Setting &renderer, Scene &scene);
00039
00043             static void parseCamera(const libconfig::Setting &camera, Scene &scene);
00044
00048             static ShapeType parseShapeType(const std::string &type);
00049
00053             static void parseShapes(const libconfig::Setting &shapesSetting, Scene &scene);
00054
00058             static std::unique_ptr<AMaterial> parseMaterial(const libconfig::Setting
      &materialSetting);
00059
00063             static LightType parseLightType(const std::string &type);
00064
00068             static void parseLights(const libconfig::Setting &lightsSetting, Scene &scene);
00069
00076             template <typename T, typename ConversionFunc>
00077             static T getVector(const libconfig::Setting &setting, ConversionFunc convert);
00078
00083             template<typename T>
00084             static T convertInt(const libconfig::Setting &setting);
00085
00088             class ParserException : public std::exception
00089             {
00090                 public:
00091                     explicit ParserException(std::string msg) : m_msg{std::move(msg)} {}
00092
00093                     ~ParserException() override = default;
00094
00095                     ParserException(const ParserException &) = delete;
00096                     ParserException &operator=(const ParserException &) = delete;
00097
00098                     ParserException(const ParserException &&) = delete;
00099                     ParserException &operator=(const ParserException &&) = delete;
00100
00102                     [[nodiscard]] const char *what() const noexcept override { return m_msg.c_str();
      };
00103
00104                 private:
00106                     std::string m_msg{0};
00107
00108         }; // class ParserException
00109
00110     }; // class Parser
00111
00112 } // namespace RayTracer
00113
00114 #endif //RAYTRACER_PARSER_HPP
```

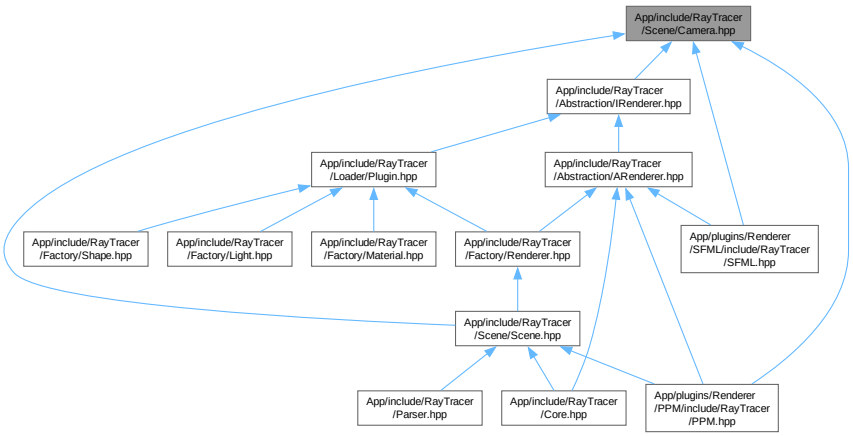
5.38 App/include/RayTracer/Scene/Camera.hpp File Reference

```
#include "RayTracer/Utils/Vector.hpp"
```

Include dependency graph for Camera.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class rtr::Camera

  A class to handle the camera.

5.39 Camera.hpp

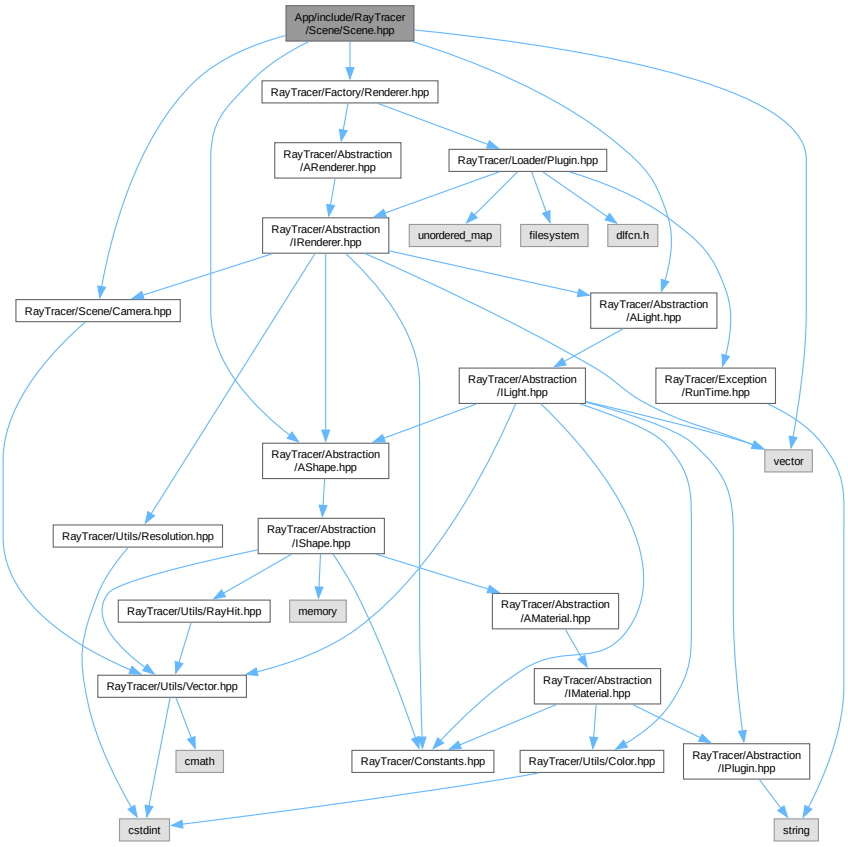
Go to the documentation of this file.

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** Raytracer | Camera
00004 ** File description:
00005 ** Camera.hpp
00006 */
00007
00009 #ifndef RAYTRACER_CAMERA_HPP
00010 #define RAYTRACER_CAMERA_HPP
00011
00012 #include "RayTracer/Utils/Vector.hpp"
00013
00014 namespace rtr {
00015
00018     class Camera {
00019         public:
00020             Camera() = default;
00021             ~Camera() = default;
00022             Camera(uint16_t fov, const Vector &origin, const Vector &direction);
00023
00026             void setFov(const uint16_t fov) { m_fov = fov; };
00027
00030             [[nodiscard]] uint16_t getFov() const { return m_fov; };
00031
00033             [[nodiscard]] const Vector &getOrigin() const { return m_origin; };
00034
00036             [[nodiscard]] const Vector &getDirection() const { return m_direction; };
00037
00039             [[nodiscard]] const Vector &getUp() const { return m_up; };
00040
00045             [[nodiscard]] std::pair<Vector, Vector> ray(const double u, const double v) const {
00046                 return {m_origin, (m_lowerLeftCorner + m_horizontal * u + m_vertical * v -
    m_origin).normalize()};
00047             }
00048
00049         private:
00051             uint16_t m_fov{0};
00052
00054             Vector m_origin{0, 0, 0};
00055
00057             Vector m_direction{0, 0, 0};
00058
00060             Vector m_up{0, 1, 0};
00061
00063             double m_aspectRatio{1.77777778};
00064
00066             Vector m_horizontal, m_vertical, m_lowerLeftCorner;
00067
00069             Vector m_u, m_v, m_w;
00070
00071     }; // class Camera
00072
00073 } // namespace RayTracer
00074
00075 #endif //RAYTRACER_CAMERA_HPP
```

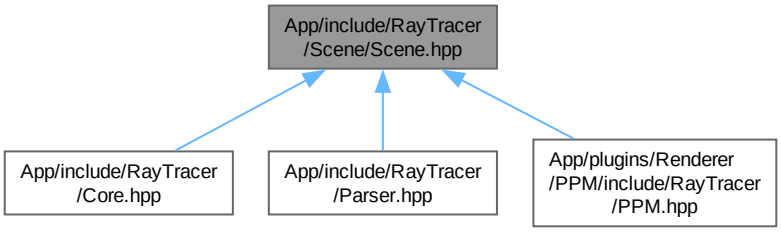
5.40 App/include/RayTracer/Scene/Scene.hpp File Reference

```
#include <vector>
#include "RayTracer/Scene/Camera.hpp"
#include "RayTracer/Abstraction/ALight.hpp"
#include "RayTracer/Abstraction/AShape.hpp"
#include "RayTracer/Factory/Renderer.hpp"
```

Include dependency graph for Scene.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class rtr::Scene

  A class to represent the scene.

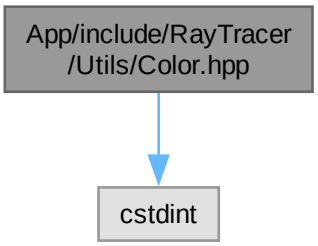
5.41 Scene.hpp

Go to the documentation of this file.

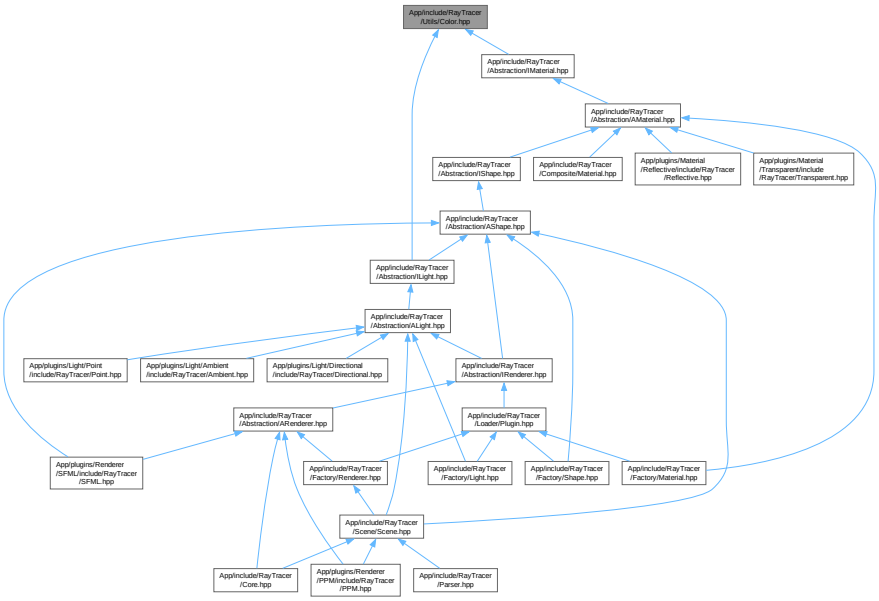
```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** Raytracer | Scene
00004 ** File description:
00005 ** Scene.hpp
00006 */
00007
00009 #ifndef RAYTRACER_SCENE_HPP
00010 #define RAYTRACER_SCENE_HPP
00011
00012 #include <vector>
00013
00014 #include "RayTracer/Scene/Camera.hpp"
00015 #include "RayTracer/Abstraction/ALight.hpp"
00016 #include "RayTracer/Abstraction/AShape.hpp"
00017 #include "RayTracer/Factory/Renderer.hpp"
00018
00019 namespace rtr {
00020
00023     class Scene {
00024         public:
00025             Scene() = default;
00026             ~Scene() = default;
00027
00030             void setCamera(const Camera &camera) { m_camera = camera; };
00031
00034             void setRenderer(std::unique_ptr<ARenderer> renderer) { m_renderer = std::move(renderer);
      };
00035
00038             void addShape(std::unique_ptr<AShape> shape) { m_shapes.emplace_back(std::move(shape)); };
00039
00042             void addLight(std::unique_ptr<ALight> light) { m_lights.emplace_back(std::move(light)); };
00043
00045             [[nodiscard]] Camera& getCamera() { return m_camera; };
00046
00049             [[nodiscard]] const std::unique_ptr<ARenderer>& getRenderer() const { return m_renderer;
      };
00050
00053             [[nodiscard]] const std::vector<std::unique_ptr<AShape>& getShapes() const { return
      m_shapes; };
00054
00057             [[nodiscard]] const std::vector<std::unique_ptr<ALight>& getLights() const { return
      m_lights; };
00058
00059         private:
00061             Camera m_camera;
00062
00064             std::unique_ptr<ARenderer> m_renderer;
00065
00067             std::vector<std::unique_ptr<AShape>> m_shapes;
00068
00070             std::vector<std::unique_ptr<ALight>> m_lights;
00071
00072     }; // class Scene
00073
00074 } // namespace RayTracer
00075
00076 #endif //RAYTRACER_SCENE_HPP
```

5.42 App/include/RayTracer/Utils/Color.hpp File Reference

`#include <cstdint>`
Include dependency graph for Color.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class rtr::Color

  Class representing RGB colors.

Typedefs

- using rtr::color_t

5.42.1 Typedef Documentation

5.42.1.1 color_t

```
using rtr::color_t
```

Initial value:

```
 struct color_s {

        uint8_t r{0};

        uint8_t g{0};

        uint8_t b{0};
    }
```

5.43 Color.hpp

Go to the documentation of this file.

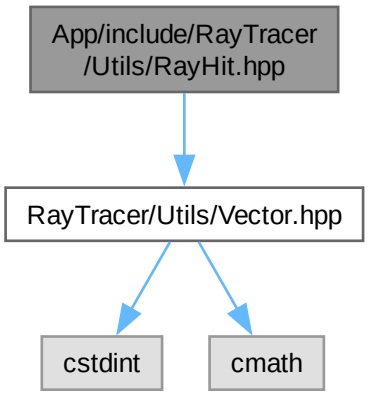
```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** RayTracer
00004 ** File description:
00005 ** Color.hpp
00006 */
00007
00009 #ifndef RAYTRACER_COLOR_HPP
00010 #define RAYTRACER_COLOR_HPP
00011
00012 #include <cstdint>
00013
00015 static constexpr int RGB_MAX = 255;
00016
00018 static constexpr int RGB_HALF = 128;
00019
00020 namespace rtr {
00021
00025     using color_t = struct color_s {
00027         uint8_t r{0};
00029         uint8_t g{0};
00031         uint8_t b{0};
00032     };
00033
00036     class Color {
00037
00038         public:
00039
00040             Color() : m_color{0, 0, 0} {};
00041             Color(const uint8_t r, const uint8_t g, const uint8_t b) : m_color{r, g, b} {};
00042             explicit Color(const color_t &color) : m_color{color} {};
00043             ~Color() = default;
00044
00049             void setColor(const uint8_t r, const uint8_t g, const uint8_t b) { m_color = {r, g, b}; };
00050
00053             void setColor(const color_t &color) { m_color = color; };
00054             void setR(const uint8_t r) { m_color.r = r; };
00055             void setG(const uint8_t g) { m_color.g = g; };
00056             void setB(const uint8_t b) { m_color.b = b; };
00057
00058             [[nodiscard]] color_t getValue() const { return m_color; };
00059             [[nodiscard]] uint8_t getR() const { return m_color.r; };
00060             [[nodiscard]] uint8_t getG() const { return m_color.g; };
00061             [[nodiscard]] uint8_t getB() const { return m_color.b; };
00062
00063             static constexpr color_t getRed() { return color_t{RGB_MAX, 0, 0}; };
00064             static constexpr color_t getGreen() { return color_t{0, RGB_MAX, 0}; };
00065             static constexpr color_t getBlue() { return color_t{0, 0, RGB_MAX}; };
00066             static constexpr color_t getWhite() { return color_t{RGB_MAX, RGB_MAX, RGB_MAX}; };
00067             static constexpr color_t getBlack() { return color_t{0, 0, 0}; };
00068             static constexpr color_t getYellow() { return color_t{RGB_MAX, RGB_MAX, 0}; };
00069             static constexpr color_t getMagenta() { return color_t{RGB_MAX, 0, RGB_MAX}; };
00070             static constexpr color_t getCyan() { return color_t{0, RGB_MAX, RGB_MAX}; };
00071             static constexpr color_t getGray() { return color_t{RGB_HALF, RGB_HALF, RGB_HALF}; };
00072             static constexpr color_t getOrange() { return color_t{RGB_MAX, 165, 0}; };
00073             static constexpr color_t getBrown() { return color_t{165, 42, 42}; };
00074
00075             static constexpr color_t getLightBlue() { return color_t{173, 216, 230}; };
```

```
00076             static constexpr color_t getLightGreen() { return color_t{144, 238, 144}; };
00077             static constexpr color_t getLightPink() { return color_t{RGB_MAX, 182, 193}; };
00078             static constexpr color_t getLightYellow() { return color_t{RGB_MAX, RGB_MAX, 224}; };
00079             static constexpr color_t getLightGray() { return color_t{211, 211, 211}; };
00080 
00081             static constexpr color_t getDarkGray() { return color_t{169, 169, 169}; };
00082             static constexpr color_t getDarkRed() { return color_t{139, 0, 0}; };
00083             static constexpr color_t getDarkGreen() { return color_t{0, 100, 0}; };
00084             static constexpr color_t getDarkBlue() { return color_t{0, 0, 139}; };
00085             static constexpr color_t getDarkYellow() { return color_t{139, 139, 0}; };
00086 
00090             Color operator+(const Color &other) const {
00091                 return {
00092                     static_cast<uint8_t>(m_color.r + other.getValue().r),
00093                     static_cast<uint8_t>(m_color.g + other.getValue().g),
00094                     static_cast<uint8_t>(m_color.b + other.getValue().b)
00095                 };
00096             };
00097 
00101             Color operator*(const double &scalar) const {
00102                 return {
00103                     static_cast<uint8_t>(m_color.r * scalar),
00104                     static_cast<uint8_t>(m_color.g * scalar),
00105                     static_cast<uint8_t>(m_color.b * scalar)
00106                 };
00107             };
00108 
00112             Color operator*(const Color &other) const {
00113                 return {
00114                     static_cast<uint8_t>(m_color.r * other.getValue().r),
00115                     static_cast<uint8_t>(m_color.g * other.getValue().g),
00116                     static_cast<uint8_t>(m_color.b * other.getValue().b)
00117                 };
00118             };
00119 
00123             Color operator+=(const Color &other) {
00124                 m_color = {
00125                     static_cast<uint8_t>(m_color.r + other.getValue().r),
00126                     static_cast<uint8_t>(m_color.g + other.getValue().g),
00127                     static_cast<uint8_t>(m_color.b + other.getValue().b)
00128                 };
00129                 return *this;
00130             };
00131 
00135             Color operator*=(const double &scalar) {
00136                 m_color = {
00137                     static_cast<uint8_t>(m_color.r * scalar),
00138                     static_cast<uint8_t>(m_color.g * scalar),
00139                     static_cast<uint8_t>(m_color.b * scalar)
00140                 };
00141                 return *this;
00142             };
00143 
00144         private:
00145 
00147             color_t m_color{0, 0, 0};
00148 
00149     }; // class Color
00150 
00151 } // namespace RayTracer
00152 
00153 #endif // RAYTRACER_COLOR_HPP
```

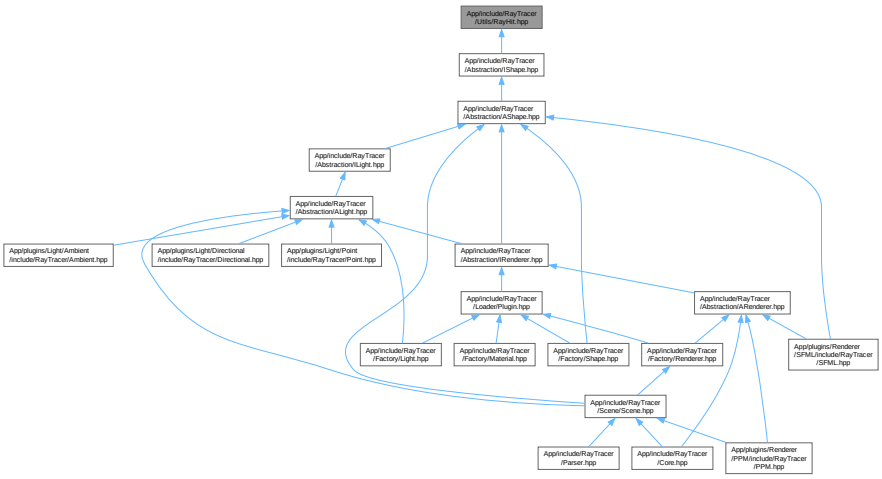
5.44 App/include/RayTracer/Utils/RayHit.hpp File Reference

```
#include "RayTracer/Utils/Vector.hpp"
```

Include dependency graph for RayHit.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class rtr::RayHit

  A class representing a ray hit in 3D space.

Typedefs

- using rtr::ray_hit_t

5.44.1 Typedef Documentation

5.44.1.1 ray_hit_t

```
using rtr::ray_hit_t
```

Initial value:

```
 struct ray_hit_s {

        Vector point;

        Vector normal;

        double distance;
    }
```

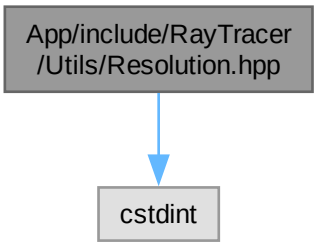
5.45 RayHit.hpp

Go to the documentation of this file.

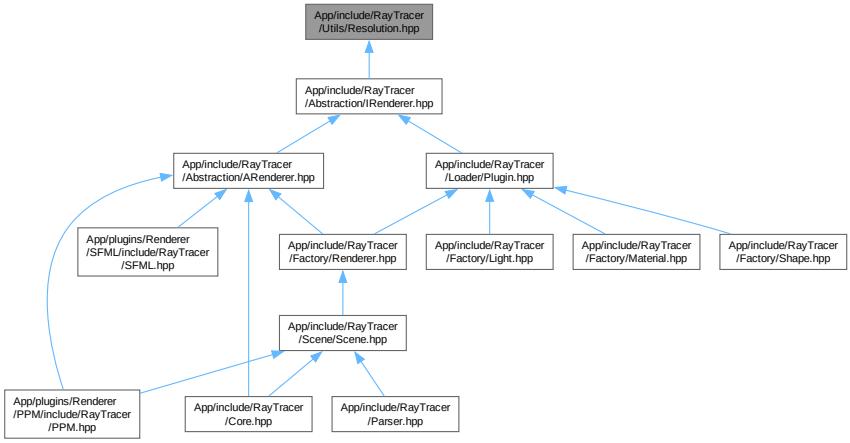
```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** raytracer
00004 ** File description:
00005 ** RayHit
00006 */
00007
00009 #ifndef RAYTRACER_RAY_HIT_HPP
00010 #define RAYTRACER_RAY_HIT_HPP
00011
00012 #include "RayTracer/Utils/Vector.hpp"
00013
00014 namespace rtr {
00015
00019     using ray_hit_t = struct ray_hit_s {
00021         Vector point;
00023         Vector normal;
00025         double distance;
00026     };
00027
00030     class RayHit {
00031
00032         public:
00033
00034             [[nodiscard]] const ray_hit_t &getRayHit() const noexcept { return m_rayHit; };
00035
00038             void setRayHit(const ray_hit_t &ray_hit) noexcept { m_rayHit = ray_hit; };
00039
00044             void setRayHit(const Vector &point, const Vector &normal, const double &distance) noexcept
      {
00045                 m_rayHit.point = point;
00046                 m_rayHit.normal = normal;
00047                 m_rayHit.distance = distance;
00048             };
00049
00052             void setPoint(const Vector &point) noexcept { m_rayHit.point = point; };
00053
00056             void setNormal(const Vector &normal) noexcept { m_rayHit.normal = normal; };
00057
00060             void setDistance(const double &distance) noexcept { m_rayHit.distance = distance; };
00061
00062         private:
00063
00065             ray_hit_t m_rayHit;
00066
00067     }; // class RayHit
00068
00069 } // namespace RayTracer
00070
00071 #endif // RAYTRACER_RAY_HIT_HPP
```

5.46 App/include/RayTracer/Utils/Resolution.hpp File Reference

`#include <cstdint>`
Include dependency graph for Resolution.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class rtr::Resolution

  Class representing the resolution of an image.

Typedefs

- using rtr::resolution_t

5.46.1 Typedef Documentation

5.46.1.1 resolution_t

```
using rtr::resolution_t
```

Initial value:

```
struct resolution_s {

        uint16_t width;

        uint16_t height;
    }
```

5.47 Resolution.hpp

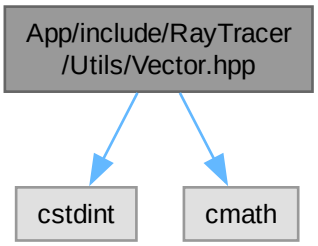
Go to the documentation of this file.

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** raytracer
00004 ** File description:
00005 ** Resolution
00006 */
00007
00009 #ifndef RAYTRACER_RESOLUTION_HPP
00010 #define RAYTRACER_RESOLUTION_HPP
00011
00012 #include <cstdint>
00013
00014 namespace rtr {
00015
00019     using resolution_t = struct resolution_s {
00021         uint16_t width;
00023         uint16_t height;
00024     };
00025
00028     class Resolution {
00029
00030         public:
00031             Resolution() : m_resolution{1920, 1080} {};
00032             Resolution(const uint16_t &width, const uint16_t &height) : m_resolution{width, height}
      {};
00033             explicit Resolution(const resolution_t &resolution) : m_resolution{resolution} {};
00034             ~Resolution() = default;
00035
00038             void setWidth(const uint16_t &width) { m_resolution.width = width; };
00039
00042             void setHeight(const uint16_t &height) { m_resolution.height = height; };
00043
00047             void setResolution(const uint16_t &width, const uint16_t &height) { m_resolution = {width,
      height}; };
00048
00051             void setResolution(const resolution_t &resolution) { m_resolution = resolution; };
00052
00055             [[nodiscard]] uint16_t getWidth() const { return m_resolution.width; }
00056
00059             [[nodiscard]] uint16_t getHeight() const { return m_resolution.height; }
00060
00063             [[nodiscard]] resolution_t getValue() const { return m_resolution; };
00064
00065         private:
00066
00068             resolution_t m_resolution{0, 0};
00069
00070     }; // class Resolution
00071
00072 }; // namespace RayTracer
00073
00074 #endif //RAYTRACER_RESOLUTION_HPP
```

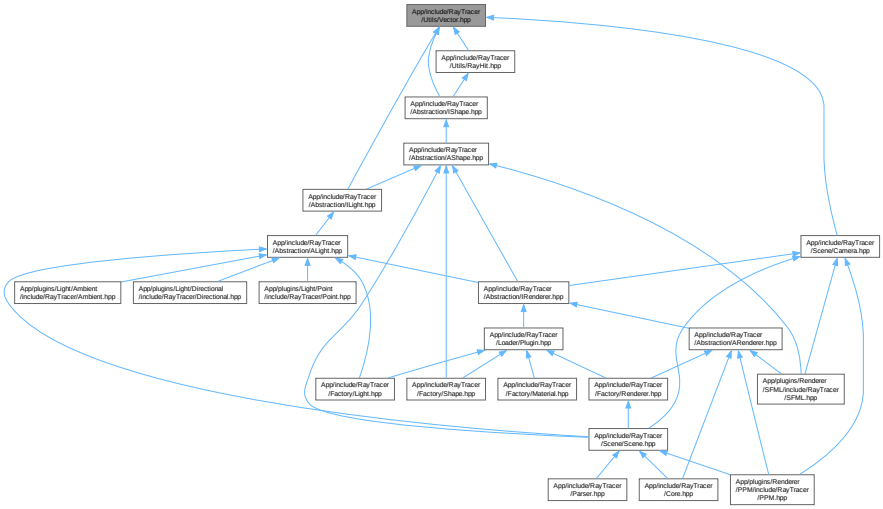
5.48 App/include/RayTracer/Utils/Vector.hpp File Reference

```
#include <cstdint>
#include <cmath>
```

Include dependency graph for Vector.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class rtr::Vector

Typedefs

- using rtr::vector_t

5.48.1 Typedef Documentation

5.48.1.1 vector_t

```
using rtr::vector_t
```

Initial value:

```
 struct vector_s {

        double x;

        double y;

        double z;
    }
```

5.49 Vector.hpp

Go to the documentation of this file.

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** raytracer
00004 ** File description:
00005 ** Vector.hpp
00006 */
00007
00009 #ifndef RAYTRACER_VECTOR_HPP
00010 #define RAYTRACER_VECTOR_HPP
00011
00012 #include <cstdint>
00013 #include <cmath>
00014
00015 namespace rtr {
00016
00020     using vector_t = struct vector_s {
00022         double x;
00024         double y;
00026         double z;
00027     };
00028
00029     class Vector {
00030
00031         public:
00032             Vector() : m_position{0, 0, 0} {};
00033             Vector(const double x, const double y, const double z) : m_position{x, y, z} {};
00034             explicit Vector(const vector_t position) : m_position{position} {};
00035             ~Vector() = default;
00036
00039             void setX(const double x) { m_position.x = x; };
00040
00043             void setY(const double y) { m_position.y = y; };
00044
00047             void setZ(const double z) { m_position.z = z; };
00048
00053             void setVector(const double x, const double y, const double z) { m_position = {x, y, z};
      };
00054
00057             void setVector(const vector_t &position) { m_position = position; };
00058
00061             [[nodiscard]] double getX() const { return m_position.x; };
00062
00065             [[nodiscard]] double getY() const { return m_position.y; };
00066
00069             [[nodiscard]] double getZ() const { return m_position.z; };
00070
00073             [[nodiscard]] vector_t getValue() const { return m_position; };
00074
00078             Vector operator+(const Vector &other) const {
00079                 return {
00080                     m_position.x + other.getX(),
00081                     m_position.y + other.getY(),
00082                     m_position.z + other.getZ()
00083                 };
00084             };
00085
00089             Vector operator+(const double scalar) const {
00090                 return {
```

```
00091                     m_position.x + scalar,
00092                     m_position.y + scalar,
00093                     m_position.z + scalar
00094                 };
00095             };
00096
00100             Vector operator-(const Vector &other) const {
00101                 return {
00102                     m_position.x - other.getX(),
00103                     m_position.y - other.getY(),
00104                     m_position.z - other.getZ()
00105                 };
00106             };
00107
00111             Vector operator*(const Vector &other) const {
00112                 return {
00113                         m_position.x * other.getX(),
00114                         m_position.y * other.getY(),
00115                         m_position.z * other.getZ()
00116                 };
00117             };
00118
00122             Vector operator*(const double scalar) const {
00123                 return {
00124                     m_position.x * scalar,
00125                     m_position.y * scalar,
00126                     m_position.z * scalar
00127                 };
00128             };
00129
00133             Vector operator/(const double scalar) const {
00134                 return {
00135                     m_position.x / scalar,
00136                     m_position.y / scalar,
00137                     m_position.z / scalar
00138                 };
00139             };
00140
00143             [[nodiscard]] double length() const { return sqrt(m_position.x * m_position.x +
      m_position.y * m_position.y + m_position.z * m_position.z); };
00144
00148             [[nodiscard]] double dot(const Vector &other) const { return m_position.x * other.getX() +
      m_position.y * other.getY() + m_position.z * other.getZ(); };
00149
00153             [[nodiscard]] Vector cross(const Vector &other) const { return {
00154                     m_position.y * other.getZ() - m_position.z * other.getY(),
00155                     m_position.z * other.getX() - m_position.x * other.getZ(),
00156                     m_position.x * other.getY() - m_position.y * other.getX()
00157                 };
00158             };
00159
00162             [[nodiscard]] Vector normalize() const {
00163                 const double len = length();
00164                 return {
00165                     m_position.x / len,
00166                     m_position.y / len,
00167                     m_position.z / len
00168                 };
00169             };
00170
00171         private:
00172
00174             vector_t m_position{0, 0, 0};
00175
00176     }; // class Vector
00177
00178 }; // namespace RayTracer
00179
00180 #endif //RAYTRACER_VECTOR_HPP
```

5.50 Ambient.hpp

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** Raytracer | Lights
00004 ** File description:
00005 ** Ambient.hpp
00006 */
00007
00008 #ifndef RAYTRACER_AMBIENT_LIGHT_HPP
00009 #define RAYTRACER_AMBIENT_LIGHT_HPP
00010
00011 #include "RayTracer/Abstraction/ALight.hpp"
```

```
00012 #include "RayTracer/Exception/RunTime.hpp"
00013
00014 namespace rtr {
00015
00016     class Ambient : public ALight {
00017
00018         public:
00019             ~Ambient() override = default;
00020
00021             Color LightColor(const Vector &normal, const Color &col) override;
00022
00023             [[nodiscard]] std::string getPluginName() const override { return AMBIENT_LIGHT; };
00024             [[nodiscard]] Vector& getDirection() override { throw RunTimeException("Ambient light has
      no direction");};
00025
00026     }; // class Ambient
00027
00028 } // namespace RayTracer
00029
00030 #endif // RAYTRACER_AMBIENT_LIGHT_HPP
```

5.51 Directional.hpp

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** Raytracer | Lights
00004 ** File description:
00005 ** Directional.hpp
00006 */
00007
00008 #ifndef RAYTRACER_DIRECTIONAL_LIGHT_HPP
00009 #define RAYTRACER_DIRECTIONAL_LIGHT_HPP
00010
00011 #include "RayTracer/Abstraction/ALight.hpp"
00012
00013 namespace rtr {
00014
00015     class Directional : public ALight {
00016
00017         public:
00018             Directional() = default;
00019             ~Directional() override = default;
00020
00021             Color LightColor(const Vector &normal, const Color &col) override;
00022
00023             [[nodiscard]] std::string getPluginName() const override { return DIRECTIONAL_LIGHT; };
00024
00025     }; // class Directional
00026
00027 } // namespace RayTracer
00028
00029 #endif // RAYTRACER_DIRECTIONAL_LIGHT_HPP
```

5.52 Point.hpp

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** Raytracer | Lights
00004 ** File description:
00005 ** Point.hpp
00006 */
00007
00008 #ifndef RAYTRACER_POINT_LIGHT_HPP
00009 #define RAYTRACER_POINT_LIGHT_HPP
00010
00011 #include "RayTracer/Abstraction/ALight.hpp"
00012 #include "RayTracer/Exception/RunTime.hpp"
00013
00014 namespace rtr {
00015
00016     class Point : public ALight {
00017
00018         public:
00019             ~Point() override = default;
00020
00021             [[nodiscard]] std::string getPluginName() const override { return POINT_LIGHT; };
00022
00023             Color LightColor(const Vector &normal, const Color &col) override;
00024
```

```
00025             [[nodiscard]] Vector& getDirection() override { throw RunTimeException("Point light has no
      direction");};
00026
00027     }; // class Point
00028
00029 } // namespace RayTracer
00030
00031 #endif // RAYTRACER_POINT_LIGHT_HPP
```

5.53 Reflective.hpp

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** Raytracer | Reflective
00004 ** File description:
00005 ** Reflective.hpp
00006 */
00007
00008 #ifndef RAYTRACER_REFLECTIVE_HPP
00009 #define RAYTRACER_REFLECTIVE_HPP
00010
00011 #include "RayTracer/Abstraction/AMaterial.hpp"
00012
00013 namespace rtr {
00014
00015     class Reflective : public AMaterial {
00016
00017         public:
00018
00019             ~Reflective() override = default;
00020
00021             void applyMaterial(Color* color) override { (void) color; };
00022
00023             [[nodiscard]] std::string getPluginName() const override { return REFLECTIVE_MATERIAL; };
00024
00025     }; // class Reflective
00026
00027 } // namespace RayTracer
00028
00029 #endif // RAYTRACER_REFLECTIVE_HPP
```

5.54 Transparent.hpp

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** Raytracer | Transparent
00004 ** File description:
00005 ** Transparent.hpp
00006 */
00007
00008 #ifndef RAYTRACER_TRANSPARENT_HPP
00009 #define RAYTRACER_TRANSPARENT_HPP
00010
00011 #include "RayTracer/Abstraction/AMaterial.hpp"
00012
00013 namespace rtr {
00014
00015     class Transparent : public AMaterial {
00016
00017         public:
00018             ~Transparent() override = default;
00019
00020             void applyMaterial(Color* color) override {
00021                 uint8_t r = static_cast<uint8_t>(color->getValue().r * (1 - this->getTransparency()));
00022                 uint8_t g = static_cast<uint8_t>(color->getValue().g * (1 - this->getTransparency()));
00023                 uint8_t b = static_cast<uint8_t>(color->getValue().b * (1 - this->getTransparency()));
00024
00025                 color->setColor(r, g, b);
00026
00027                 /*   ALPHA BLENDING
00028                  *
00029                  *  r = c1.r * c1.a + c2.r * c2.a * (1 - c1.a)
00030                  *  g = c1.g * c1.a + c2.g * c2.a * (1 - c1.a)
00031                  *  b = c1.b * c1.a + c2.b * c2.a * (1 - c1.a)
00032                  *  a = c1.a + c2.a * (1 - c1.a)
00033                  *
00034                  */
00035             };
00036
```

```
00037             [[nodiscard]] std::string getPluginName() const override { return TRANSPARENT_MATERIAL; };
00038
00039     }; // class Transparent
00040
00041 } // namespace RayTracer
00042
00043 #endif // RAYTRACER_TRANSPARENT_HPP
```

5.55 PPM.hpp

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** Raytracer | renderer
00004 ** File description:
00005 ** PPM.hpp
00006 */
00007
00008 #ifndef RAYTRACER_PPM_RENDERER_HPP
00009 #define RAYTRACER_PPM_RENDERER_HPP
00010
00011 #include "RayTracer/Abstraction/ARenderer.hpp"
00012 #include "RayTracer/Scene/Camera.hpp"
00013 #include "RayTracer/Scene/Scene.hpp"
00014
00015 namespace rtr {
00016
00017     class PPM : public ARenderer {
00018
00019         public:
00020             ~PPM() override = default;
00021
00022             [[nodiscard]] std::string getPluginName() const override { return PPM_RENDERER; };
00023
00024             [[nodiscard]] static std::string getHeader(const std::string &width, const std::string
      &height) { return "P6\n" + width + ' ' + height + "\n255\n"; };
00025
00026             void render(const std::vector<std::unique_ptr<AShape» &shapes, const
      std::vector<std::unique_ptr<ALight» &lights, const Camera &camera) override;
00027
00028             void writePixels(const Color color, const std::size_t width, const std::size_t height) {
      getPixels()[height][width].setColor(color.getValue()); };
00029             void writeToFile(const std::string &width, const std::string &height);
00030             bool isShadowed(const Vector &lightDir, const Vector &point, const
      std::vector<std::unique_ptr<AShape» &shapes);
00031
00032     }; // class PPM
00033
00034 } // namespace RayTracer
00035
00036 #endif // RAYTRACER_PPM_RENDERER_HPP
```

5.56 SFML.hpp

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** Raytracer | renderer
00004 ** File description:
00005 ** SFML.hpp
00006 */
00007
00008 #ifndef RAYTRACER_SFML_RENDERER_HPP
00009 #define RAYTRACER_SFML_RENDERER_HPP
00010
00011 #include "RayTracer/Abstraction/AShape.hpp"
00012 #include "RayTracer/Abstraction/ARenderer.hpp"
00013 #include "RayTracer/Scene/Camera.hpp"
00014
00015 namespace rtr {
00016
00017     class SFML : public ARenderer {
00018
00019         public:
00020             SFML() = default;
00021             ~SFML() override = default;
00022
00023             [[nodiscard]] std::string getPluginName() const override { return SFML_RENDERER; };
00024
00025             void render(const std::vector<std::unique_ptr<AShape» &shapes, const
      std::vector<std::unique_ptr<ALight» &lights, const Camera &camera) override;
```

```
00026
00027     }; // class SFML
00028
00029 } // namespace RayTracer
00030
00031 #endif // RAYTRACER_SFML_RENDERER_HPP
```

Index