My Project

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

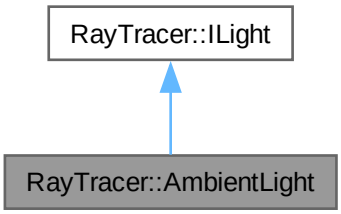
Class Documentation

4.1 RayTracer::AmbientLight Class Reference

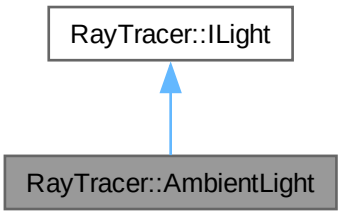
Represents an ambient light source.

`#include <AmbientLight.hpp>`

Inheritance diagram for RayTracer::AmbientLight:



Collaboration diagram for RayTracer::AmbientLight:



Public Member Functions

- AmbientLight (const Math::Vector3D &color, double intensity)

  Constructs an ambient light with a color and intensity.
- Math::Vector3D getDirectionTo (const Math::Point3D &point) const override

  Gets the direction to a point (not applicable for ambient light).
- double getDistanceTo (const Math::Point3D &point) const override

  Gets the distance to a point (not applicable for ambient light).
- ILight::Type getType () const override

  Gets the type of the light.
- const Math::Vector3D & getColor () const override

  Gets the color of the light.
- double getIntensity () const override

  Gets the intensity of the light.

Public Member Functions inherited from RayTracer::ILight

- virtual ~**ILight** ()=default

  Default virtual destructor.

Additional Inherited Members

Public Types inherited from RayTracer::ILight

- enum class Type { **POINT** , **DIRECTIONAL** , **AMBIENT** }

  Enum representing the type of light.

4.1.1 Detailed Description

Represents an ambient light source.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 AmbientLight()

```
RayTracer::AmbientLight::AmbientLight (
            const Math::Vector3D & color,
            double intensity )
```

Constructs an ambient light with a color and intensity.

Parameters

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

4.1.3 Member Function Documentation

4.1.3.1 getColor()

```
const Math::Vector3D & RayTracer::AmbientLight::getColor ( ) const  [override], [virtual]
```

Gets the color of the light.

Returns

The color vector.

Implements RayTracer::ILight.

4.1.3.2 getDirectionTo()

```
Math::Vector3D RayTracer::AmbientLight::getDirectionTo (
            const Math::Point3D & point ) const  [override], [virtual]
```

Gets the direction to a point (not applicable for ambient light).

Parameters

point	The point to calculate the direction to.

Returns

A zero vector as ambient light has no direction.

Implements RayTracer::ILight.

4.1.3.3 getDistanceTo()

```
double RayTracer::AmbientLight::getDistanceTo (
            const Math::Point3D & point ) const  [override], [virtual]
```

Gets the distance to a point (not applicable for ambient light).

Parameters

point	The point to calculate the distance to.

Returns

Zero as ambient light has no distance.

Implements RayTracer::ILight.

4.1.3.4 getIntensity()

```
double RayTracer::AmbientLight::getIntensity ( ) const  [override], [virtual]
```

Gets the intensity of the light.

Returns

The intensity value.

Implements RayTracer::ILight.

4.1.3.5 getType()

```
ILight::Type RayTracer::AmbientLight::getType ( ) const  [override], [virtual]
```

Gets the type of the light.

Returns

The type of the light (AMBIENT).

Implements RayTracer::ILight.

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

- src/lights/AmbientLight.hpp
- src/lights/AmbientLight.cpp

4.2 RayTracer::Camera Class Reference

Represents a camera in the ray tracer.

`#include <Camera.hpp>`

Public Member Functions

- Camera (const Math::Point3D &position, double fov, int image_width, int image_height)
  Constructs a camera with a position, field of view, and image dimensions.
- Ray ray (int i, int j) const
  Generates a ray for a specific pixel in the image.
- Math::Point3D getOrigin () const
  Gets the origin of the camera.

4.2.1 Detailed Description

Represents a camera in the ray tracer.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Camera()

```
RayTracer::Camera::Camera (
            const Math::Point3D & position,
            double fov,
            int image_width,
            int image_height )
```

Constructs a camera with a position, field of view, and image dimensions.

Parameters

position	The position of the camera in 3D space.
fov	The field of view of the camera in degrees.
image_width	The width of the image to render.
image_height	The height of the image to render.

4.2.3 Member Function Documentation

4.2.3.1 getOrigin()

```
Math::Point3D RayTracer::Camera::getOrigin ( ) const  [inline]
```

Gets the origin of the camera.

Returns

The origin point of the camera.

4.2.3.2 ray()

```
Ray RayTracer::Camera::ray (
            int i,
            int j ) const
```

Generates a ray for a specific pixel in the image.

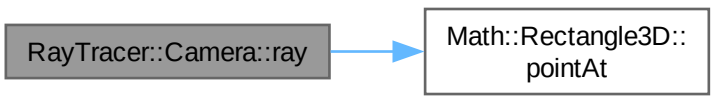
Parameters

i	The row index of the pixel.
j	The column index of the pixel.

Returns

The ray corresponding to the pixel.

Here is the call graph for this function:



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

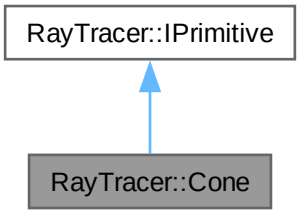
- src/core/Camera.hpp
- src/core/Camera.cpp

4.3 RayTracer::Cone Class Reference

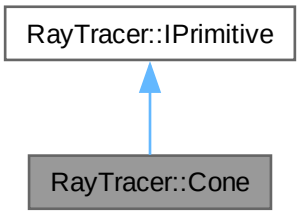
Represents a 3D cone primitive.

`#include <Cone.hpp>`

Inheritance diagram for RayTracer::Cone:



Collaboration diagram for RayTracer::Cone:



Public Member Functions

- Cone (const Math::Point3D &base, double radius, double height, const Math::Vector3D &color)
  Constructs a cone with a base, radius, height, and color.
- ~**Cone** ()=default
  Default destructor.
- bool hits (const Ray &ray, double &t) const override
  Checks if a ray intersects the cone.
- Math::Vector3D getNormal (const Math::Point3D &point) const override
  Gets the normal vector at a given point on the cone.
- Math::Vector3D getColor () const override
  Gets the color of the cone.

Public Member Functions inherited from RayTracer::IPrimitive

- virtual ~**IPrimitive** ()=default

  Default virtual destructor.

4.3.1 Detailed Description

Represents a 3D cone primitive.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Cone()

```
RayTracer::Cone::Cone (
            const Math::Point3D & base,
            double radius,
            double height,
            const Math::Vector3D & color )
```

Constructs a cone with a base, radius, height, and color.

Parameters

base	The base point of the cone.
radius	The radius of the cone's base.
height	The height of the cone.
color	The color of the cone.

4.3.3 Member Function Documentation

4.3.3.1 getColor()

```
Math::Vector3D RayTracer::Cone::getColor ( ) const  [override], [virtual]
```

Gets the color of the cone.

Returns

The color vector of the cone.

Implements RayTracer::IPrimitive.

4.3.3.2 getNormal()

```
Math::Vector3D RayTracer::Cone::getNormal (
            const Math::Point3D & point ) const  [override], [virtual]
```

Gets the normal vector at a given point on the cone.

Parameters

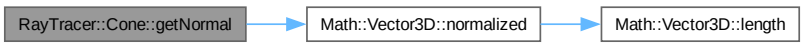
point	The point on the cone.

Returns

The normal vector at the given point.

Implements RayTracer::IPrimitive.

Here is the call graph for this function:



4.3.3.3 hits()

```
bool RayTracer::Cone::hits (
            const Ray & ray,
            double & t ) const  [override], [virtual]
```

Checks if a ray intersects the cone.

Parameters

ray	The ray to check.
t	The distance to the intersection point, if any.

Returns

True if the ray intersects the cone, false otherwise.

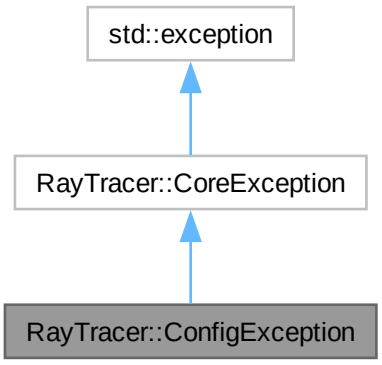
Implements RayTracer::IPrimitive.

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

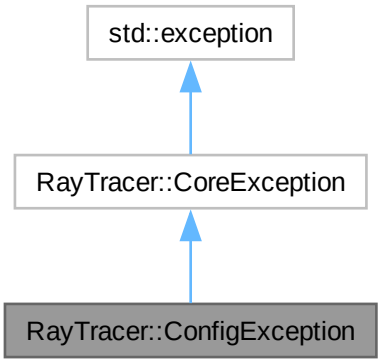
- src/primitives/Cone.hpp
- src/primitives/Cone.cpp

4.4 RayTracer::ConfigException Class Reference

Inheritance diagram for RayTracer::ConfigException:



Collaboration diagram for RayTracer::ConfigException:



Public Member Functions

- **ConfigException** (const std::string &message)

Public Member Functions inherited from RayTracer::CoreException

- **CoreException** (const std::string &message)
- virtual const char * **what** () const noexcept override

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

- src/core/Exception.hpp

4.5 RayTracer::Core Class Reference

Represents the core of the ray tracer, responsible for rendering scenes.

`#include <Core.hpp>`

Public Member Functions

- **Core** ()
  Default constructor.
- ~**Core** ()=default
  Default destructor.
- int render (std::string configfile)
  Renders a scene based on a configuration file.

4.5.1 Detailed Description

Represents the core of the ray tracer, responsible for rendering scenes.

4.5.2 Member Function Documentation

4.5.2.1 render()

```
int RayTracer::Core::render (
            std::string configfile )
```

Renders a scene based on a configuration file.

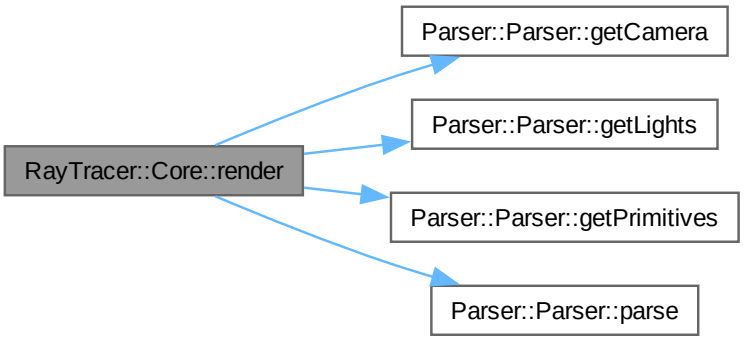
Parameters

configfile	The path to the configuration file.

Returns

An integer indicating the success or failure of the rendering process.

Here is the call graph for this function:

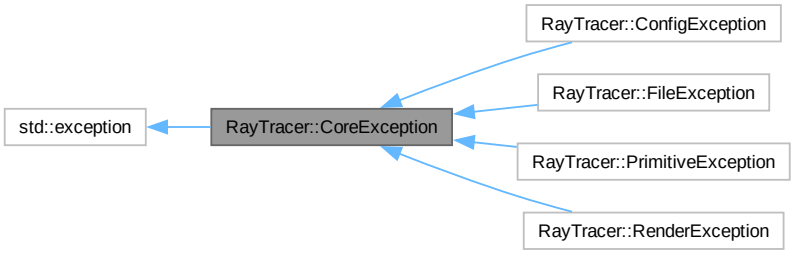


The documentation for this class was generated from the following files:

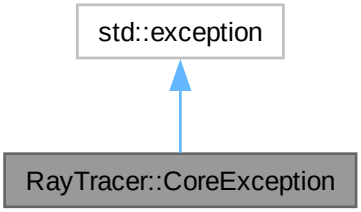
- src/core/Core.hpp
- src/core/Core.cpp

4.6 RayTracer::CoreException Class Reference

Inheritance diagram for RayTracer::CoreException:



Collaboration diagram for RayTracer::CoreException:



Public Member Functions

- **CoreException** (const std::string &message)
- virtual const char * **what** () const noexcept override

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

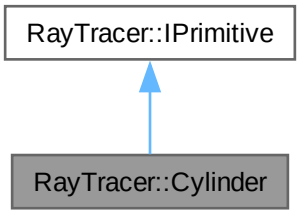
- src/core/Exception.hpp

4.7 RayTracer::Cylinder Class Reference

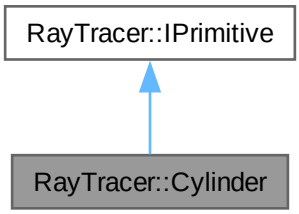
Represents a 3D cylinder primitive.

`#include <Cylinder.hpp>`

Inheritance diagram for RayTracer::Cylinder:



Collaboration diagram for RayTracer::Cylinder:



Public Member Functions

- Cylinder (const Math::Point3D &base, double radius, double height, const Math::Vector3D &color)

  Constructs a cylinder with a base, radius, height, and color.
- ~**Cylinder** ()=default

  Default destructor.
- bool hits (const Ray &ray, double &t) const override

  Checks if a ray intersects the cylinder.
- Math::Vector3D getNormal (const Math::Point3D &point) const override

  Gets the normal vector at a given point on the cylinder.
- Math::Vector3D getColor () const override

  Gets the color of the cylinder.

Public Member Functions inherited from RayTracer::IPrimitive

- virtual ~**IPrimitive** ()=default

  Default virtual destructor.

4.7.1 Detailed Description

Represents a 3D cylinder primitive.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 Cylinder()

```
RayTracer::Cylinder::Cylinder (
            const Math::Point3D & base,
            double radius,
            double height,
            const Math::Vector3D & color )
```

Constructs a cylinder with a base, radius, height, and color.

Parameters

base	The base point of the cylinder.
radius	The radius of the cylinder.
height	The height of the cylinder.
color	The color of the cylinder.

4.7.3 Member Function Documentation

4.7.3.1 getColor()

```
Math::Vector3D RayTracer::Cylinder::getColor ( ) const  [override], [virtual]
```

Gets the color of the cylinder.

Returns

The color vector of the cylinder.

Implements RayTracer::IPrimitive.

4.7.3.2 getNormal()

```
Math::Vector3D RayTracer::Cylinder::getNormal (
            const Math::Point3D & point ) const  [override], [virtual]
```

Gets the normal vector at a given point on the cylinder.

Parameters

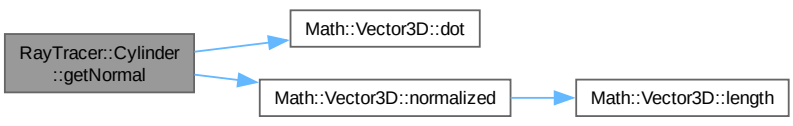
point	The point on the cylinder.

Returns

The normal vector at the given point.

Implements RayTracer::IPrimitive.

Here is the call graph for this function:



4.7.3.3 hits()

```
bool RayTracer::Cylinder::hits (
            const Ray & ray,
            double & t ) const  [override], [virtual]
```

Checks if a ray intersects the cylinder.

Parameters

ray	The ray to check.
t	The distance to the intersection point, if any.

Returns

True if the ray intersects the cylinder, false otherwise.

Implements RayTracer::IPrimitive.

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

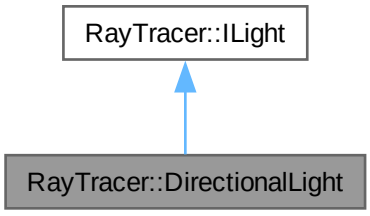
- src/primitives/Cylinder.hpp
- src/primitives/Cylinder.cpp

4.8 RayTracer::DirectionalLight Class Reference

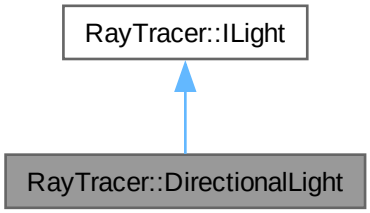
Represents a directional light source.

```
#include <DirectionalLight.hpp>
```

Inheritance diagram for RayTracer::DirectionalLight:



Collaboration diagram for RayTracer::DirectionalLight:



Public Member Functions

- DirectionalLight (const Math::Vector3D &direction, const Math::Vector3D &color, double intensity=1.0)

  Constructs a directional light.
- Math::Vector3D getDirectionTo (const Math::Point3D &point) const override

  Gets the direction to a point (always the light's direction).
- double getDistanceTo (const Math::Point3D &point) const override

  Gets the distance to a point (infinite for directional light).
- Type getType () const override

  Gets the type of the light.
- const Math::Vector3D & getColor () const override

  Gets the color of the light.
- double getIntensity () const override

  Gets the intensity of the light.

Public Member Functions inherited from RayTracer::ILight

- virtual ~**ILight** ()=default

  Default virtual destructor.

Additional Inherited Members

Public Types inherited from RayTracer::ILight

- enum class Type { **POINT** , **DIRECTIONAL** , **AMBIENT** }

  Enum representing the type of light.

4.8.1 Detailed Description

Represents a directional light source.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 DirectionalLight()

```
RayTracer::DirectionalLight::DirectionalLight (
            const Math::Vector3D & direction,
            const Math::Vector3D & color,
            double intensity = 1.0 )
```

Constructs a directional light.

Parameters

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

4.8.3 Member Function Documentation

4.8.3.1 getColor()

```
const Math::Vector3D & RayTracer::DirectionalLight::getColor ( ) const  [override], [virtual]
```

Gets the color of the light.

Returns

The color vector.

Implements RayTracer::ILight.

4.8.3.2 getDirectionTo()

```
Math::Vector3D RayTracer::DirectionalLight::getDirectionTo (
            const Math::Point3D & point ) const  [override], [virtual]
```

Gets the direction to a point (always the light's direction).

Parameters

point	The point to calculate the direction to.

Returns

The direction vector.

Implements RayTracer::ILight.

4.8.3.3 getDistanceTo()

```
double RayTracer::DirectionalLight::getDistanceTo (
            const Math::Point3D & point ) const  [override], [virtual]
```

Gets the distance to a point (infinite for directional light).

Parameters

point	The point to calculate the distance to.

Returns

The distance (always infinity).

Implements RayTracer::ILight.

4.8.3.4 getIntensity()

```
double RayTracer::DirectionalLight::getIntensity ( ) const  [override], [virtual]
```

Gets the intensity of the light.

Returns

The intensity value.

Implements RayTracer::ILight.

4.8.3.5 getType()

```
ILight::Type RayTracer::DirectionalLight::getType ( ) const  [override], [virtual]
```

Gets the type of the light.

Returns

The type of the light (DIRECTIONAL).

Implements RayTracer::ILight.

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

- src/lights/DirectionalLight.hpp
- src/lights/DirectionalLight.cpp

4.9 DLLoader< T > Class Template Reference

A generic dynamic library loader for plugins.

`#include <DLLoader.hpp>`

Public Member Functions

- DLLoader (const std::string &path)
  Constructs a DLLoader and attempts to load the specified library.
- ~**DLLoader** ()
  Destructor. Closes the dynamic library if it is open.
- bool isValid () const
  Checks if the library and its symbols were successfully loaded.
- std::shared_ptr< T > getInstance () const
  Creates an instance of the plugin.
- const std::string & getName () const
  Gets the name of the plugin.
- const std::string & getType () const
  Gets the type of the plugin.
- std::string getError () const
  Gets the error message if the library failed to load.

4.9.1 Detailed Description

template<typename T>
class DLLoader< T >

A generic dynamic library loader for plugins.

Template Parameters

T	The type of the plugin interface.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 DLLoader()

```
template<typename T >
DLLoader< T >::DLLoader (
            const std::string & path )  [inline]
```

Constructs a DLLoader and attempts to load the specified library.

Parameters

path	The path to the dynamic library.

4.9.3 Member Function Documentation

4.9.3.1 getError()

```
template<typename T >
std::string DLLoader< T >::getError ( ) const  [inline]
```

Gets the error message if the library failed to load.

Returns

The error message.

4.9.3.2 getInstance()

```
template<typename T >
std::shared_ptr< T > DLLoader< T >::getInstance ( ) const  [inline]
```

Creates an instance of the plugin.

Returns

A shared pointer to the created plugin instance.

Exceptions

std::runtime_error	If the library is not valid.

Here is the call graph for this function:

4.9.3.3 getName()

```
template<typename T >
const std::string & DLLoader< T >::getName ( ) const  [inline]
```

Gets the name of the plugin.

Returns

The name of the plugin.

Exceptions

std::runtime_error	If the library is not valid.

Here is the call graph for this function:

4.9.3.4 getType()

```
template<typename T >
const std::string & DLLoader< T >::getType ( ) const  [inline]
```

Gets the type of the plugin.

Returns

The type of the plugin.

Exceptions

std::runtime_error	If the library is not valid.

Here is the call graph for this function:

4.9.3.5 isValid()

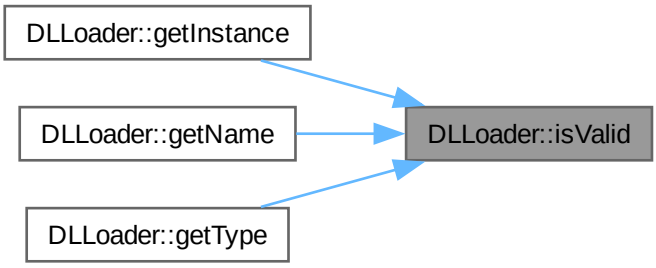
```
template<typename T >
bool DLLoader< T >::isValid ( ) const  [inline]
```

Checks if the library and its symbols were successfully loaded.

Returns

True if valid, false otherwise.

Here is the caller graph for this function:

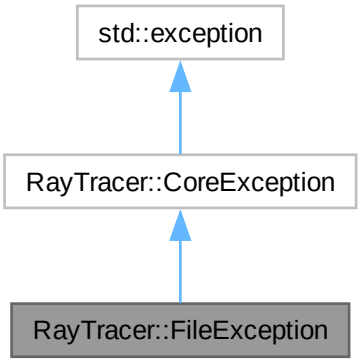


The documentation for this class was generated from the following file:

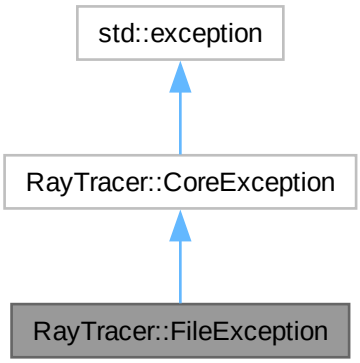
- src/parser/DLLoader.hpp

4.10 RayTracer::FileException Class Reference

Inheritance diagram for RayTracer::FileException:



Collaboration diagram for RayTracer::FileException:



Public Member Functions

- **FileException** (const std::string &message)

Public Member Functions inherited from RayTracer::CoreException

- **CoreException** (const std::string &message)
- virtual const char * **what** () const noexcept override

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

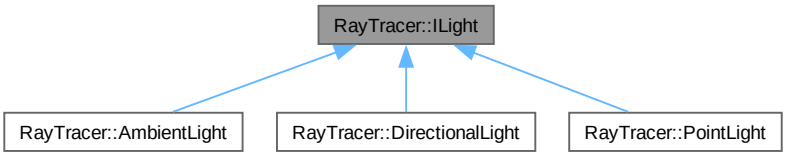
- src/core/Exception.hpp

4.11 RayTracer::ILight Class Reference

Interface for light sources in the ray tracer.

`#include <ILight.hpp>`

Inheritance diagram for RayTracer::ILight:



Public Types

- enum class Type { **POINT** , **DIRECTIONAL** , **AMBIENT** }

  Enum representing the type of light.

Public Member Functions

- virtual ~**ILight** ()=default

  Default virtual destructor.
- virtual Math::Vector3D getDirectionTo (const Math::Point3D &point) const =0

  Gets the direction to a point from the light.
- virtual double getDistanceTo (const Math::Point3D &point) const =0

  Gets the distance to a point from the light.
- virtual Type getType () const =0

  Gets the type of the light.
- virtual const Math::Vector3D & getColor () const =0

  Gets the color of the light.
- virtual double getIntensity () const =0

  Gets the intensity of the light.

4.11.1 Detailed Description

Interface for light sources in the ray tracer.

4.11.2 Member Function Documentation

4.11.2.1 getColor()

```
virtual const Math::Vector3D & RayTracer::ILight::getColor ( ) const  [pure virtual]
```

Gets the color of the light.

Returns

The color vector.

Implemented in RayTracer::AmbientLight, RayTracer::DirectionalLight, and RayTracer::PointLight.

4.11.2.2 getDirectionTo()

```
virtual Math::Vector3D RayTracer::ILight::getDirectionTo (
            const Math::Point3D & point ) const  [pure virtual]
```

Gets the direction to a point from the light.

Parameters

point	The point to calculate the direction to.

Returns

The direction vector.

Implemented in RayTracer::AmbientLight, RayTracer::DirectionalLight, and RayTracer::PointLight.

4.11.2.3 getDistanceTo()

```
virtual double RayTracer::ILight::getDistanceTo (
            const Math::Point3D & point ) const  [pure virtual]
```

Gets the distance to a point from the light.

Parameters

point	The point to calculate the distance to.

Returns

The distance value.

Implemented in RayTracer::AmbientLight, RayTracer::DirectionalLight, and RayTracer::PointLight.

4.11.2.4 getIntensity()

```
virtual double RayTracer::ILight::getIntensity ( ) const  [pure virtual]
```

Gets the intensity of the light.

Returns

The intensity value.

Implemented in RayTracer::AmbientLight, RayTracer::DirectionalLight, and RayTracer::PointLight.

4.11.2.5 getType()

```
virtual Type RayTracer::ILight::getType ( ) const  [pure virtual]
```

Gets the type of the light.

Returns

The type of the light.

Implemented in RayTracer::AmbientLight, RayTracer::DirectionalLight, and RayTracer::PointLight.

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

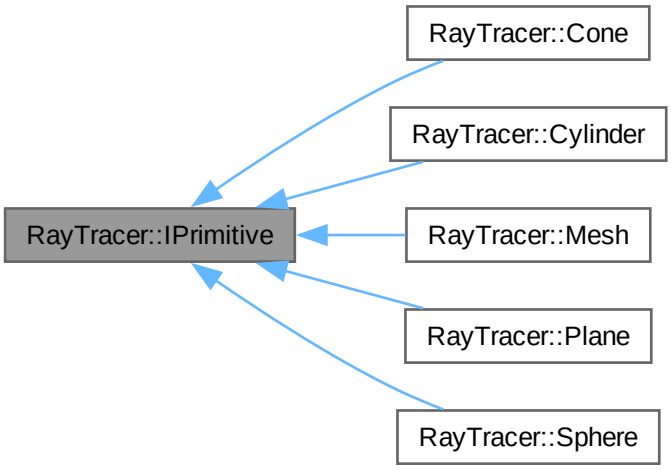
- src/lights/ILight.hpp

4.12 RayTracer::IPrimitive Class Reference

Interface for 3D primitives in the ray tracer.

`#include <IPrimitive.hpp>`

Inheritance diagram for RayTracer::IPrimitive:



Public Member Functions

- virtual ~**IPrimitive** ()=default
  Default virtual destructor.
- virtual bool hits (const Ray &ray, double &t) const =0
  Checks if a ray intersects the primitive.
- virtual Math::Vector3D getNormal (const Math::Point3D &point) const =0
  Gets the normal vector at a given point on the primitive.
- virtual Math::Vector3D getColor () const =0
  Gets the color of the primitive.

4.12.1 Detailed Description

Interface for 3D primitives in the ray tracer.

4.12.2 Member Function Documentation

4.12.2.1 getColor()

```
virtual Math::Vector3D RayTracer::IPrimitive::getColor ( ) const  [pure virtual]
```

Gets the color of the primitive.

Returns

The color vector of the primitive.

Implemented in RayTracer::Cone, RayTracer::Cylinder, RayTracer::Mesh, RayTracer::Plane, and RayTracer::Sphere.

4.12.2.2 getNormal()

```
virtual Math::Vector3D RayTracer::IPrimitive::getNormal (
            const Math::Point3D & point ) const  [pure virtual]
```

Gets the normal vector at a given point on the primitive.

Parameters

point	The point on the primitive.

Returns

The normal vector at the given point.

Implemented in RayTracer::Cone, RayTracer::Cylinder, RayTracer::Mesh, RayTracer::Plane, and RayTracer::Sphere.

4.12.2.3 hits()

```
virtual bool RayTracer::IPrimitive::hits (
            const Ray & ray,
            double & t ) const  [pure virtual]
```

Checks if a ray intersects the primitive.

Parameters

ray	The ray to check.
t	The distance to the intersection point, if any.

Returns

True if the ray intersects the primitive, false otherwise.

Implemented in RayTracer::Cone, RayTracer::Cylinder, RayTracer::Mesh, RayTracer::Plane, and RayTracer::Sphere.

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

- src/primitives/IPrimitive.hpp

4.13 Math::Matrix Class Reference

Represents a 4x4 transformation matrix for 3D operations.

```
#include <Matrix.hpp>
```

Public Member Functions

- **Matrix** ()

  Default constructor. Initializes the matrix to zero.
- Matrix rotationX (double angle)

  Creates a rotation matrix around the X-axis.
- Matrix rotationY (double angle)

  Creates a rotation matrix around the Y-axis.
- Point3D transform (const Point3D &p) const

  Transforms a 3D point using the matrix.
- Vector3D transform (const Vector3D &v) const

  Transforms a 3D vector using the matrix.
- Matrix operator∗ (const Matrix &other) const

  Multiplies this matrix with another matrix.
- double & elementAt (int row, int col)

  Accesses an element of the matrix for modification.
- double elementAt (int row, int col) const

  Accesses an element of the matrix for reading.
- void **print** () const

  Prints the matrix to the standard output.

Static Public Member Functions

- static Matrix identity ()

  Creates an identity matrix.
- static Matrix translation (double tx, double ty, double tz)

  Creates a translation matrix.
- static Matrix scale (double sx, double sy, double sz)

  Creates a scaling matrix.
- static Matrix rotationZ (double angle)

  Creates a rotation matrix around the Z-axis.

4.13.1 Detailed Description

Represents a 4x4 transformation matrix for 3D operations.

4.13.2 Member Function Documentation

4.13.2.1 elementAt() [1/2]

```
double & Math::Matrix::elementAt (
            int row,
            int col )
```

Accesses an element of the matrix for modification.

Parameters

row	The row index (0-based).
col	The column index (0-based).

Returns

A reference to the element.

Exceptions

std::out_of_range	If the indices are out of bounds.

4.13.2.2 elementAt() [2/2]

```
double Math::Matrix::elementAt (
            int row,
            int col ) const
```

Accesses an element of the matrix for reading.

Parameters

row	The row index (0-based).
col	The column index (0-based).

Returns

The value of the element.

Exceptions

std::out_of_range	If the indices are out of bounds.

4.13.2.3 identity()

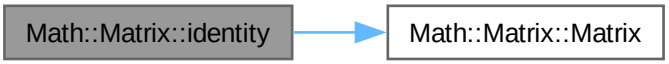
```
Matrix Math::Matrix::identity ( )  [static]
```

Creates an identity matrix.

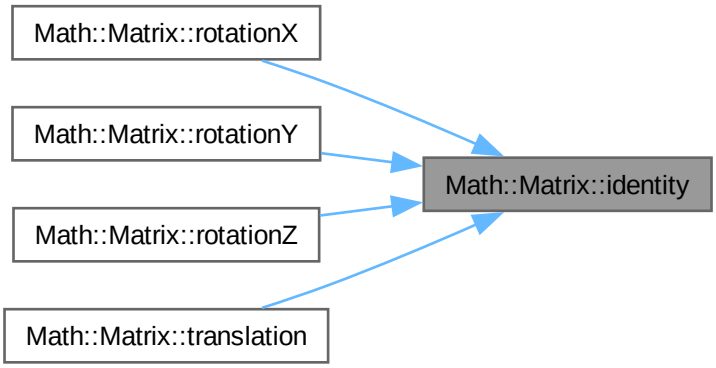
Returns

An identity matrix.

Here is the call graph for this function:



Here is the caller graph for this function:



4.13.2.4 operator∗()

```
Matrix Math::Matrix::operator* (
            const Matrix & other ) const
```

Multiplies this matrix with another matrix.

Parameters

other	The matrix to multiply with.

Returns

The resulting matrix.

4.13.2.5 rotationX()

```
Matrix Math::Matrix::rotationX (
            double angle )
```

Creates a rotation matrix around the X-axis.

Parameters

angle	The rotation angle in radians.

Returns

A rotation matrix for the X-axis.

Here is the call graph for this function:

4.13.2.6 rotationY()

```
Matrix Math::Matrix::rotationY (
            double angle )
```

Creates a rotation matrix around the Y-axis.

Parameters

angle	The rotation angle in radians.

Returns

A rotation matrix for the Y-axis.

Here is the call graph for this function:

4.13.2.7 rotationZ()

```
Matrix Math::Matrix::rotationZ (
            double angle )  [static]
```

Creates a rotation matrix around the Z-axis.

Parameters

angle	The rotation angle in radians.

Returns

A rotation matrix for the Z-axis.

Here is the call graph for this function:

4.13.2.8 scale()

```
Matrix Math::Matrix::scale (
            double sx,
            double sy,
            double sz )  [static]
```

Creates a scaling matrix.

Parameters

sx	Scaling factor along the x-axis.
sy	Scaling factor along the y-axis.
sz	Scaling factor along the z-axis.

Returns

A scaling matrix.

4.13.2.9 transform() [1/2]

```
Point3D Math::Matrix::transform (
            const Point3D & p ) const
```

Transforms a 3D point using the matrix.

Parameters

p	The point to transform.

Returns

The transformed point.

4.13.2.10 transform() [2/2]

```
Vector3D Math::Matrix::transform (
            const Vector3D & v ) const
```

Transforms a 3D vector using the matrix.

Parameters

v	The vector to transform.

Returns

The transformed vector.

4.13.2.11 translation()

```
Matrix Math::Matrix::translation (
            double tx,
            double ty,
            double tz )  [static]
```

Creates a translation matrix.

Parameters

tx	Translation along the x-axis.
ty	Translation along the y-axis.
tz	Translation along the z-axis.

Returns

A translation matrix.

Here is the call graph for this function:

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

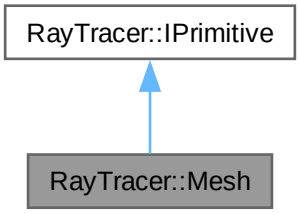
- src/utils/Matrix.hpp
- src/utils/Matrix.cpp

4.14 RayTracer::Mesh Class Reference

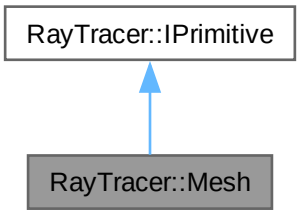
Class representing a 3D mesh loaded from an .obj file.

`#include <Mesh.hpp>`

Inheritance diagram for RayTracer::Mesh:



Collaboration diagram for RayTracer::Mesh:



Public Member Functions

- Mesh (const std::string &filename, const Math::Vector3D &color, const Math::Point3D &position=Math::Point3D(0, 0, 0), double scale=1.0)

  Constructs a new Mesh from an OBJ file.
- ~**Mesh** ()=default

  Default destructor.
- bool hits (const Ray &ray, double &t) const override

  Checks if a ray hits the mesh.
- Math::Vector3D getNormal (const Math::Point3D &point) const override

  Gets the normal vector at a given point on the mesh.
- Math::Vector3D getColor () const override

  Gets the color of the mesh.
- void setPosition (const Math::Point3D &position)

  Sets the position of the mesh.
- void setScale (double scale)

  Sets the scale of the mesh.

Public Member Functions inherited from RayTracer::IPrimitive

- virtual ~**IPrimitive** ()=default

  Default virtual destructor.

4.14.1 Detailed Description

Class representing a 3D mesh loaded from an .obj file.

Supports loading vertices and faces from an OBJ file.

4.14.2 Constructor & Destructor Documentation

4.14.2.1 Mesh()

```
RayTracer::Mesh::Mesh (
            const std::string & filename,
            const Math::Vector3D & color,
            const Math::Point3D & position = Math::Point3D(0, 0, 0),
            double scale = 1.0 )
```

Constructs a new Mesh from an OBJ file.

Parameters

filename	Path to the .obj file to load.
color	Color of the mesh.
position	Position of the mesh in the scene.
scale	Scale factor of the mesh.

4.14.3 Member Function Documentation

4.14.3.1 getColor()

```
Math::Vector3D RayTracer::Mesh::getColor ( ) const  [override], [virtual]
```

Gets the color of the mesh.

Returns

The color vector.

Implements RayTracer::IPrimitive.

4.14.3.2 getNormal()

```
Math::Vector3D RayTracer::Mesh::getNormal (
            const Math::Point3D & point ) const  [override], [virtual]
```

Gets the normal vector at a given point on the mesh.

Parameters

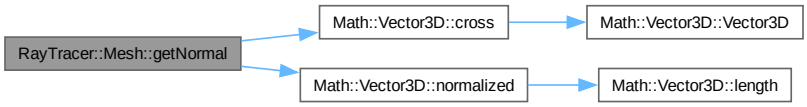
point	The point to get the normal at.

Returns

The normal vector.

Implements RayTracer::IPrimitive.

Here is the call graph for this function:



4.14.3.3 hits()

```
bool RayTracer::Mesh::hits (
            const Ray & ray,
            double & t ) const  [override], [virtual]
```

Checks if a ray hits the mesh.

Parameters

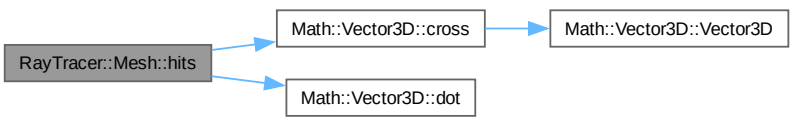
ray	The ray to check against.
t	The distance to the hit point if a hit occurred.

Returns

True if hit, false otherwise.

Implements RayTracer::IPrimitive.

Here is the call graph for this function:



4.14.3.4 setPosition()

```
void RayTracer::Mesh::setPosition (
            const Math::Point3D & position )  [inline]
```

Sets the position of the mesh.

Parameters

position	The new position.

4.14.3.5 setScale()

```
void RayTracer::Mesh::setScale (
            double scale )  [inline]
```

Sets the scale of the mesh.

Parameters

scale	The new scale factor.

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

- src/primitives/Mesh.hpp
- src/primitives/Mesh.cpp

4.15 Parser::Parser Class Reference

Parses configuration files for the ray tracer.

`#include <Parser.hpp>`

Public Member Functions

- **Parser** ()
  Default constructor.
- ~**Parser** ()=default
  Default destructor.
- int parse (const std::string &filePath)
  Parses a configuration file.
- void parseCamera (const libconfig::Setting &root)
  Parses the camera settings from the configuration.
- void parseLight (const libconfig::Setting &root)
  Parses the lights from the configuration.
- void parseDirectionalLight (const libconfig::Setting &lights)
  Parses directional lights from the configuration.

- void parsePrimitives (const libconfig::Setting &root)

  Parses primitives from the configuration.
- void parseObjects (const libconfig::Setting &primitives, std::string type)

  Parses objects of a specific type from the configuration.
- void parseMesh (const libconfig::Setting &root)

  Parses mesh objects from the configuration.
- RayTracer::Camera getCamera () const

  Gets the parsed camera.
- std::vector< std::shared_ptr< RayTracer::IPrimitive > > getPrimitives () const

  Gets the parsed primitives.
- std::vector< std::shared_ptr< RayTracer::ILight > > getLights () const

  Gets the parsed lights.

4.15.1 Detailed Description

Parses configuration files for the ray tracer.

4.15.2 Member Function Documentation

4.15.2.1 getCamera()

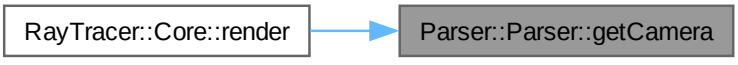
```
RayTracer::Camera Parser::Parser::getCamera ( ) const  [inline]
```

Gets the parsed camera.

Returns

The camera object.

Here is the caller graph for this function:



4.15.2.2 getLights()

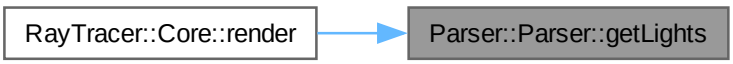
```
std::vector< std::shared_ptr< RayTracer::ILight > > Parser::Parser::getLights ( ) const  [inline]
```

Gets the parsed lights.

Returns

A vector of shared pointers to the lights.

Here is the caller graph for this function:



4.15.2.3 getPrimitives()

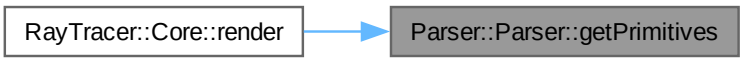
```
std::vector< std::shared_ptr< RayTracer::IPrimitive > > Parser::Parser::getPrimitives ( )
const  [inline]
```

Gets the parsed primitives.

Returns

A vector of shared pointers to the primitives.

Here is the caller graph for this function:



4.15.2.4 parse()

```
int Parser::Parser::parse (
            const std::string & filePath )
```

Parses a configuration file.

Parameters

filePath	The path to the configuration file.

Returns

An integer indicating success or failure.

Here is the caller graph for this function:

4.15.2.5 parseCamera()

```
void Parser::Parser::parseCamera (
            const libconfig::Setting & root )
```

Parses the camera settings from the configuration.

Parameters

root	The root setting of the configuration.

4.15.2.6 parseDirectionalLight()

```
void Parser::Parser::parseDirectionalLight (
            const libconfig::Setting & lights )
```

Parses directional lights from the configuration.

Parameters

lights	The setting containing directional lights.

4.15.2.7 parseLight()

```
void Parser::Parser::parseLight (
            const libconfig::Setting & root )
```

Parses the lights from the configuration.

Parameters

root	The root setting of the configuration.

4.15.2.8 parseMesh()

```
void Parser::Parser::parseMesh (
            const libconfig::Setting & root )
```

Parses mesh objects from the configuration.

Parameters

root	The root setting of the configuration.

4.15.2.9 parseObjects()

```
void Parser::Parser::parseObjects (
            const libconfig::Setting & primitives,
            std::string type )
```

Parses objects of a specific type from the configuration.

Parameters

primitives	The setting containing the primitives.
type	The type of objects to parse.

4.15.2.10 parsePrimitives()

```
void Parser::Parser::parsePrimitives (
            const libconfig::Setting & root )
```

Parses primitives from the configuration.

Parameters

root	The root setting of the configuration.

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

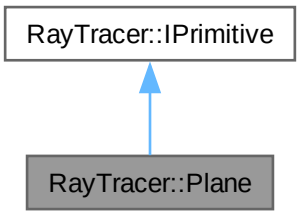
- src/parser/Parser.hpp
- src/parser/Parser.cpp

4.16 RayTracer::Plane Class Reference

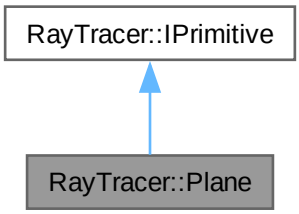
Represents a 3D plane primitive.

`#include <Plane.hpp>`

Inheritance diagram for RayTracer::Plane:



Collaboration diagram for RayTracer::Plane:



Public Member Functions

- **Plane** ()=default

  Default constructor.
- Plane (const std::string &axis, const Math::Point3D &position, const Math::Vector3D &color)

  Constructs a plane with an axis, position, and color.
- ~**Plane** ()=default

  Default destructor.
- bool hits (const Ray &ray, double &t) const override

  Checks if a ray intersects the plane.
- Math::Vector3D getNormal (const Math::Point3D &point) const override

  Gets the normal vector at a given point on the plane.
- Math::Vector3D getColor () const override

  Gets the color of the plane.
- Color getColorAt (const Math::Point3D &point) const

  Gets the color at a specific point on the plane.

Public Member Functions inherited from RayTracer::IPrimitive

- virtual ~**IPrimitive** ()=default

  Default virtual destructor.

4.16.1 Detailed Description

Represents a 3D plane primitive.

4.16.2 Constructor & Destructor Documentation

4.16.2.1 Plane()

```
RayTracer::Plane::Plane (
            const std::string & axis,
            const Math::Point3D & position,
            const Math::Vector3D & color )
```

Constructs a plane with an axis, position, and color.

Parameters

axis	The axis of the plane (e.g., "x", "y", or "z").
position	The position of the plane in 3D space.
color	The color of the plane.

4.16.3 Member Function Documentation

4.16.3.1 getColor()

```
Math::Vector3D RayTracer::Plane::getColor ( ) const  [override], [virtual]
```

Gets the color of the plane.

Returns

The color vector of the plane.

Implements RayTracer::IPrimitive.

4.16.3.2 getColorAt()

```
Color RayTracer::Plane::getColorAt (
            const Math::Point3D & point ) const
```

Gets the color at a specific point on the plane.

Parameters

point	The point on the plane.

Returns

The color at the given point.

4.16.3.3 getNormal()

```
Math::Vector3D RayTracer::Plane::getNormal (
            const Math::Point3D & point ) const  [override], [virtual]
```

Gets the normal vector at a given point on the plane.

Parameters

point	The point on the plane.

Returns

The normal vector at the given point.

Implements RayTracer::IPrimitive.

4.16.3.4 hits()

```
bool RayTracer::Plane::hits (
            const Ray & ray,
            double & t ) const  [override], [virtual]
```

Checks if a ray intersects the plane.

Parameters

ray	The ray to check.
t	The distance to the intersection point, if any.

Returns

True if the ray intersects the plane, false otherwise.

Implements RayTracer::IPrimitive.

Here is the call graph for this function:

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

- src/primitives/Plane.hpp
- src/primitives/Plane.cpp

4.17 Math::Point3D Class Reference

Represents a point in 3D space.

`#include <Point3D.hpp>`

Public Member Functions

- **Point3D** ()
  Default constructor.
- Point3D (double x, double y, double z)
  Constructs a point with given coordinates.
- ~**Point3D** ()=default
  Default destructor.
- Point3D operator+ (const Vector3D &vec) const
  Adds a vector to the point.
- Point3D & operator+= (const Vector3D &vec)
  Adds a vector to the point in place.
- Vector3D operator- (const Point3D &point) const
  Subtracts another point from this point.
- Point3D operator- (const Vector3D &vec) const
  Subtracts a vector from the point.

Public Attributes

- double **x**
  The x-coordinate of the point.
- double **y**
  The y-coordinate of the point.
- double **z**
  The z-coordinate of the point.

4.17.1 Detailed Description

Represents a point in 3D space.

4.17.2 Constructor & Destructor Documentation

4.17.2.1 Point3D()

```
Math::Point3D::Point3D (
            double x,
            double y,
            double z )
```

Constructs a point with given coordinates.

Parameters

x	The x-coordinate.
y	The y-coordinate.
z	The z-coordinate.

4.17.3 Member Function Documentation

4.17.3.1 operator+()

```
Point3D Math::Point3D::operator+ (
            const Vector3D & vec ) const
```

Adds a vector to the point.

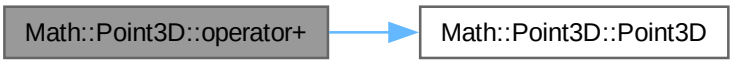
Parameters

vec	The vector to add.

Returns

A new point resulting from the addition.

Here is the call graph for this function:



4.17.3.2 operator+=()

```
Point3D & Math::Point3D::operator+= (
            const Vector3D & vec )
```

Adds a vector to the point in place.

Parameters

vec	The vector to add.

Returns

A reference to the modified point.

4.17.3.3 operator-() [1/2]

```
Vector3D Math::Point3D::operator- (
            const Point3D & point ) const
```

Subtracts another point from this point.

Parameters

point	The point to subtract.

Returns

A vector representing the difference.

4.17.3.4 operator-() [2/2]

```
Point3D Math::Point3D::operator- (
            const Vector3D & vec ) const
```

Subtracts a vector from the point.

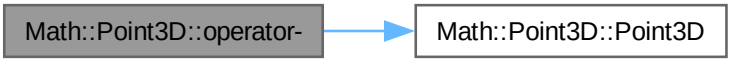
Parameters

vec	The vector to subtract.

Returns

A new point resulting from the subtraction.

Here is the call graph for this function:



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

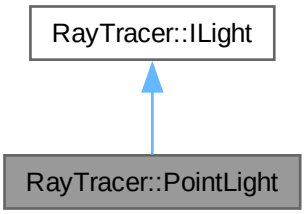
- src/utils/Point3D.hpp
- src/utils/Point3D.cpp

4.18 RayTracer::PointLight Class Reference

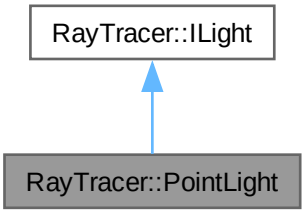
Represents a point light source.

`#include <PointLight.hpp>`

Inheritance diagram for RayTracer::PointLight:



Collaboration diagram for RayTracer::PointLight:



Public Member Functions

- PointLight (const Math::Point3D &position, const Math::Vector3D &color, double intensity=1.0)

  Constructs a point light.
- Math::Vector3D getDirectionTo (const Math::Point3D &point) const override

  Gets the direction to a point from the light.
- double getDistanceTo (const Math::Point3D &point) const override

  Gets the distance to a point from the light.
- Type getType () const override

  Gets the type of the light.
- const Math::Vector3D & getColor () const override

  Gets the color of the light.
- double getIntensity () const override

  Gets the intensity of the light.

Public Member Functions inherited from RayTracer::ILight

- virtual ~**ILight** ()=default

  Default virtual destructor.

Additional Inherited Members

Public Types inherited from RayTracer::ILight

- enum class Type { **POINT** , **DIRECTIONAL** , **AMBIENT** }

  Enum representing the type of light.

4.18.1 Detailed Description

Represents a point light source.

4.18.2 Constructor & Destructor Documentation

4.18.2.1 PointLight()

```
RayTracer::PointLight::PointLight (
            const Math::Point3D & position,
            const Math::Vector3D & color,
            double intensity = 1.0 )
```

Constructs a point light.

Parameters

position	The position of the light in 3D space.
color	The color of the light.
intensity	The intensity of the light.

4.18.3 Member Function Documentation

4.18.3.1 getColor()

```
const Math::Vector3D & RayTracer::PointLight::getColor ( ) const  [override], [virtual]
```

Gets the color of the light.

Returns

The color vector.

Implements RayTracer::ILight.

4.18.3.2 getDirectionTo()

```
Math::Vector3D RayTracer::PointLight::getDirectionTo (
            const Math::Point3D & point ) const  [override], [virtual]
```

Gets the direction to a point from the light.

Parameters

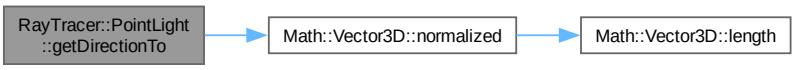
point	The point to calculate the direction to.

Returns

The direction vector.

Implements RayTracer::ILight.

Here is the call graph for this function:



4.18.3.3 getDistanceTo()

```
double RayTracer::PointLight::getDistanceTo (
            const Math::Point3D & point ) const  [override], [virtual]
```

Gets the distance to a point from the light.

Parameters

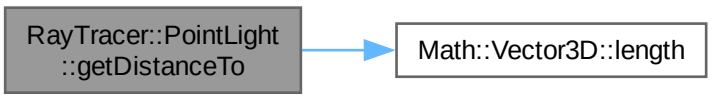
point	The point to calculate the distance to.

Returns

The distance value.

Implements RayTracer::ILight.

Here is the call graph for this function:



4.18.3.4 getIntensity()

```
double RayTracer::PointLight::getIntensity ( ) const  [override], [virtual]
```

Gets the intensity of the light.

Returns

The intensity value.

Implements RayTracer::ILight.

4.18.3.5 getType()

```
ILight::Type RayTracer::PointLight::getType ( ) const  [override], [virtual]
```

Gets the type of the light.

Returns

The type of the light (POINT).

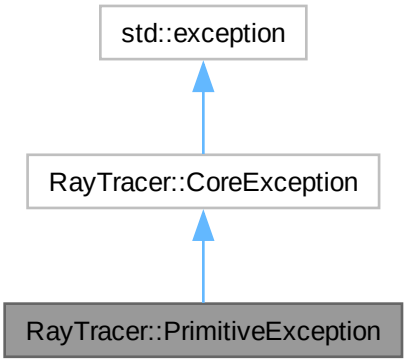
Implements RayTracer::ILight.

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

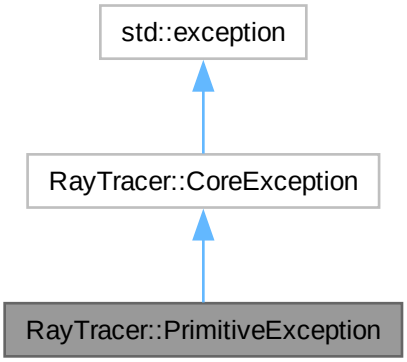
- src/lights/PointLight.hpp
- src/lights/PointLight.cpp

4.19 RayTracer::PrimitiveException Class Reference

Inheritance diagram for RayTracer::PrimitiveException:



Collaboration diagram for RayTracer::PrimitiveException:



Public Member Functions

- **PrimitiveException** (const std::string &message)

Public Member Functions inherited from RayTracer::CoreException

- **CoreException** (const std::string &message)
- virtual const char * **what** () const noexcept override

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

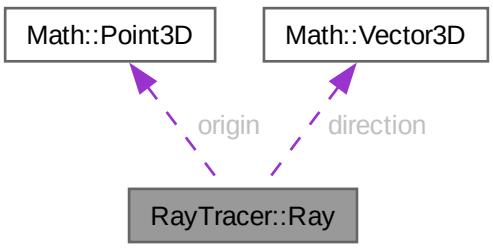
- src/core/Exception.hpp

4.20 RayTracer::Ray Class Reference

Represents a ray in 3D space.

`#include <Ray.hpp>`

Collaboration diagram for RayTracer::Ray:



Public Member Functions

- **Ray** ()=default

  Default constructor.
- Ray (const Math::Point3D &origin, const Math::Vector3D &direction)

  Constructs a ray with an origin and direction.
- ~**Ray** ()=default

  Default destructor.
- Math::Point3D getOrigin () const

  Gets the origin of the ray.
- Math::Vector3D getDirection () const

  Gets the direction of the ray.

Public Attributes

- Math::Point3D **origin**

  The origin point of the ray.
- Math::Vector3D **direction**

  The direction vector of the ray.

4.20.1 Detailed Description

Represents a ray in 3D space.

4.20.2 Constructor & Destructor Documentation

4.20.2.1 Ray()

```
RayTracer::Ray::Ray (
            const Math::Point3D & origin,
            const Math::Vector3D & direction )
```

Constructs a ray with an origin and direction.

Parameters

origin	The starting point of the ray.
direction	The direction vector of the ray.

4.20.3 Member Function Documentation

4.20.3.1 getDirection()

```
Math::Vector3D RayTracer::Ray::getDirection ( ) const
```

Gets the direction of the ray.

Returns

The direction vector of the ray.

4.20.3.2 getOrigin()

```
Math::Point3D RayTracer::Ray::getOrigin ( ) const
```

Gets the origin of the ray.

Returns

The origin point of the ray.

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

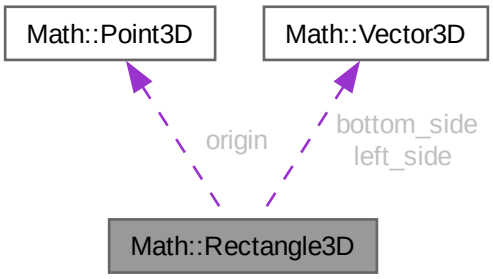
- src/utils/Ray.hpp
- src/utils/Ray.cpp

4.21 Math::Rectangle3D Class Reference

Represents a 3D rectangle defined by an origin and two side vectors.

`#include <Rectangle3D.hpp>`

Collaboration diagram for Math::Rectangle3D:



Public Member Functions

- **Rectangle3D** ()=default

  Default constructor.
- Rectangle3D (const Point3D &origin, const Vector3D &bottom_side, const Vector3D &left_side)

  Constructs a rectangle with an origin and two side vectors.
- ~**Rectangle3D** ()=default

  Default destructor.
- Point3D pointAt (double u, double v) const

  Computes a point on the rectangle given parameters u and v.

Public Attributes

- Point3D **origin**

  The origin point of the rectangle.
- Vector3D **bottom_side**

  The vector representing the bottom side of the rectangle.
- Vector3D **left_side**

  The vector representing the left side of the rectangle.

4.21.1 Detailed Description

Represents a 3D rectangle defined by an origin and two side vectors.

4.21.2 Constructor & Destructor Documentation

4.21.2.1 Rectangle3D()

```
Math::Rectangle3D::Rectangle3D (
            const Point3D & origin,
            const Vector3D & bottom_side,
            const Vector3D & left_side )
```

Constructs a rectangle with an origin and two side vectors.

Parameters

origin	The origin point of the rectangle.
bottom_side	The vector representing the bottom side of the rectangle.
left_side	The vector representing the left side of the rectangle.

4.21.3 Member Function Documentation

4.21.3.1 pointAt()

```
Point3D Math::Rectangle3D::pointAt (
            double u,
            double v ) const
```

Computes a point on the rectangle given parameters u and v.

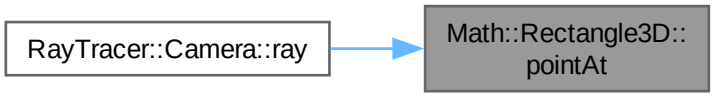
Parameters

u	The horizontal parameter (0 to 1).
v	The vertical parameter (0 to 1).

Returns

The computed point on the rectangle.

Here is the caller graph for this function:

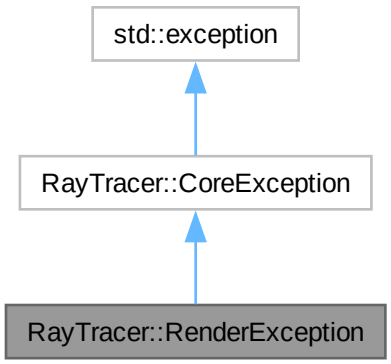


The documentation for this class was generated from the following files:

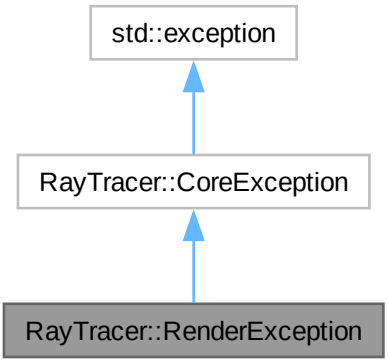
- src/utils/Rectangle3D.hpp
- src/utils/Rectangle3D.cpp

4.22 RayTracer::RenderException Class Reference

Inheritance diagram for RayTracer::RenderException:



Collaboration diagram for RayTracer::RenderException:



Public Member Functions

- **RenderException** (const std::string &message)

Public Member Functions inherited from RayTracer::CoreException

- **CoreException** (const std::string &message)
- virtual const char * **what** () const noexcept override

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

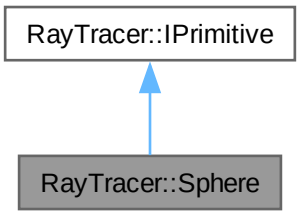
- src/core/Exception.hpp

4.23 RayTracer::Sphere Class Reference

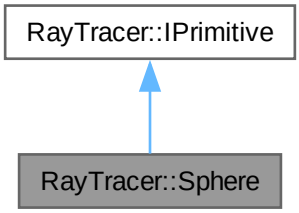
Represents a 3D sphere primitive.

`#include <Sphere.hpp>`

Inheritance diagram for RayTracer::Sphere:



Collaboration diagram for RayTracer::Sphere:



Public Member Functions

- **Sphere** ()=default

  Default constructor.
- Sphere (const Math::Point3D &center, double radius, const Math::Vector3D &color)

  Constructs a sphere with a center, radius, and color.
- ~**Sphere** ()=default

  Default destructor.
- bool hits (const Ray &ray, double &t) const override

  Checks if a ray intersects the sphere.
- Math::Vector3D getNormal (const Math::Point3D &point) const override

  Gets the normal vector at a given point on the sphere.
- Math::Vector3D getColor () const override

  Gets the color of the sphere.

Public Member Functions inherited from RayTracer::IPrimitive

- virtual ~**IPrimitive** ()=default

  Default virtual destructor.

4.23.1 Detailed Description

Represents a 3D sphere primitive.

4.23.2 Constructor & Destructor Documentation

4.23.2.1 Sphere()

```
RayTracer::Sphere::Sphere (
            const Math::Point3D & center,
            double radius,
            const Math::Vector3D & color )
```

Constructs a sphere with a center, radius, and color.

Parameters

center	The center point of the sphere.
radius	The radius of the sphere.
color	The color of the sphere.

4.23.3 Member Function Documentation

4.23.3.1 getColor()

```
Math::Vector3D RayTracer::Sphere::getColor ( ) const  [override], [virtual]
```

Gets the color of the sphere.

Returns

The color vector of the sphere.

Implements RayTracer::IPrimitive.

4.23.3.2 getNormal()

```
Math::Vector3D RayTracer::Sphere::getNormal (
            const Math::Point3D & point ) const  [override], [virtual]
```

Gets the normal vector at a given point on the sphere.

Parameters

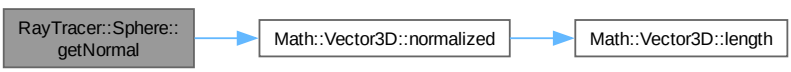
point	The point on the sphere.

Returns

The normal vector at the given point.

Implements RayTracer::IPrimitive.

Here is the call graph for this function:



4.23.3.3 hits()

```
bool RayTracer::Sphere::hits (
            const Ray & ray,
            double & t ) const  [override], [virtual]
```

Checks if a ray intersects the sphere.

Parameters

ray	The ray to check.
t	The distance to the intersection point, if any.

Returns

True if the ray intersects the sphere, false otherwise.

Implements RayTracer::IPrimitive.

Here is the call graph for this function:

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

- src/primitives/Sphere.hpp
- src/primitives/Sphere.cpp

4.24 Math::Vector< N > Class Template Reference

Public Member Functions

- **Vector** (const std::array< double, N > &components)

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

- src/utils/Vector3D.hpp

4.25 Math::Vector3D Class Reference

Represents a 3D vector with various mathematical operations.

`#include <Vector3D.hpp>`

Public Member Functions

- **Vector3D** ()

  Default constructor. Initializes the vector to (0, 0, 0).
- Vector3D (double x, double y, double z)

  Constructs a vector with given coordinates.
- ∼**Vector3D** ()=default

  Default destructor.
- double length () const

  Computes the length (magnitude) of the vector.
- Vector3D **operator+** (const Vector3D &other) const
- Vector3D **operator-** (const Vector3D &other) const
- Vector3D **operator**∗ (const Vector3D &other) const
- Vector3D **operator**/ (const Vector3D &other) const
- Vector3D **operator+=** (const Vector3D &other)
- Vector3D **operator-=** (const Vector3D &other)
- Vector3D **operator**∗**=** (const Vector3D &other)
- Vector3D **operator/=** (const Vector3D &other)
- Vector3D **operator**∗ (double scalar) const
- Vector3D **operator**/ (double scalar) const
- Vector3D **operator**∗**=** (double scalar)
- Vector3D **operator/=** (double scalar)
- Vector3D **operator-** () const
- double dot (const Vector3D &other) const

  Computes the dot product of this vector with another.
- Vector3D cross (const Vector3D &other) const

  Computes the cross product of this vector with another.
- Vector3D normalized () const

  Normalizes the vector to have a length of 1.

Public Attributes

- double **x**

  The x-coordinate of the vector.
- double **y**

  The y-coordinate of the vector.
- double **z**

  The z-coordinate of the vector.

4.25.1 Detailed Description

Represents a 3D vector with various mathematical operations.

4.25.2 Constructor & Destructor Documentation

4.25.2.1 Vector3D()

```
Math::Vector3D::Vector3D (
            double x,
            double y,
            double z )
```

Constructs a vector with given coordinates.

Parameters

x	The x-coordinate.
y	The y-coordinate.
z	The z-coordinate.

4.25.3 Member Function Documentation

4.25.3.1 cross()

```
Vector3D Math::Vector3D::cross (
            const Vector3D & other ) const
```

Computes the cross product of this vector with another.

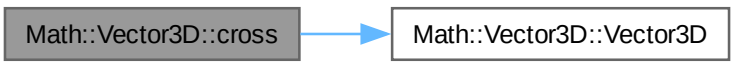
Parameters

other	The other vector.

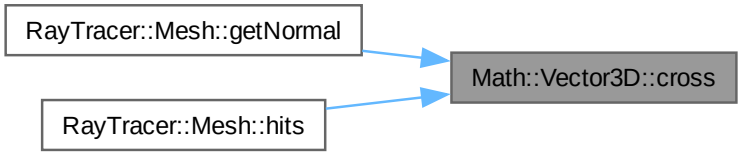
Returns

The resulting vector from the cross product.

Here is the call graph for this function:



Here is the caller graph for this function:



4.25.3.2 dot()

```
double Math::Vector3D::dot (
            const Vector3D & other ) const
```

Computes the dot product of this vector with another.

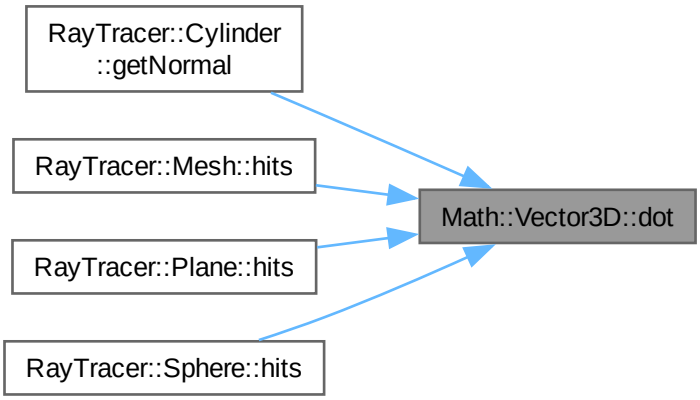
Parameters

other	The other vector.

Returns

The dot product.

Here is the caller graph for this function:



4.25.3.3 length()

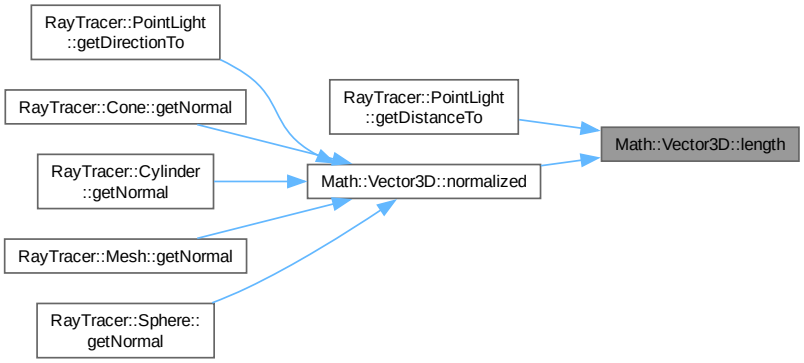
```
double Math::Vector3D::length ( ) const
```

Computes the length (magnitude) of the vector.

Returns

The length of the vector.

Here is the caller graph for this function:



4.25.3.4 normalized()

```
Vector3D Math::Vector3D::normalized ( ) const
```

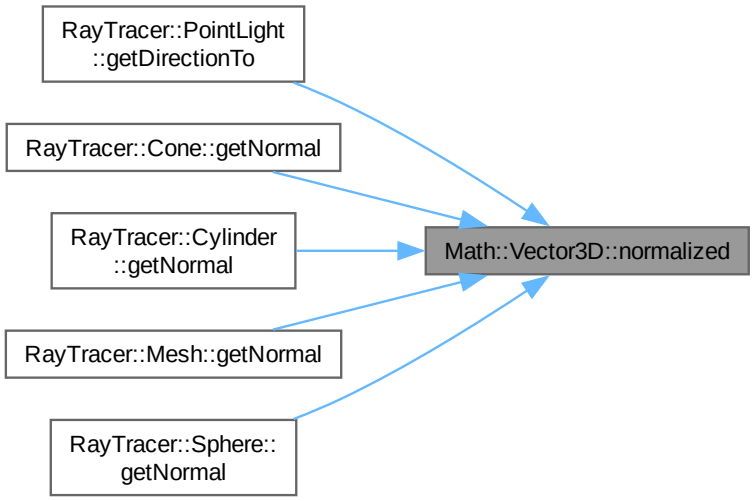
Normalizes the vector to have a length of 1.

Returns

The normalized vector.

Here is the call graph for this function:

Here is the caller graph for this function:



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

- src/utils/Vector3D.hpp
- src/utils/Vector3D.cpp

Chapter 5

File Documentation

5.1 Camera.hpp

```
100001 /*
100002 ** EPITECH PROJECT, 2025
100003 ** project
100004 ** File description:
100005 ** Camera
100006 */
100007
100008 #pragma once
100009
100010 #include <cmath>
100011 #include "Point3D.hpp"
100012 #include "Ray.hpp"
100013 #include "Rectangle3D.hpp"
100014 #include "Vector3D.hpp"
100015
100016 namespace RayTracer {
100020   class Camera {
100021   public:
100029     Camera(const Math::Point3D& position, double fov, int image_width,
100030            int image_height);
100031
100038     Ray ray(int i, int j) const;
100039
100044     Math::Point3D getOrigin() const { return origin_; }
100045
100046   private:
100047     Math::Point3D origin_;
100048     Math::Rectangle3D screen_;
100049     int image_width_;
100050     int image_height_;
100051   };
100052 }  // namespace RayTracer
```

5.2 Core.hpp

```
100001 /*
100002 ** EPITECH PROJECT, 2025
100003 ** B-OOP-400-NAN-4-1-bsraytracer-robin.schuffenecker
100004 ** File description:
100005 ** Core
100006 */
100007
100008 #pragma once
100009
100010 #include <algorithm>
100011 #include <fstream>
100012 #include <memory>
100013 #include <vector>
100014
100015 #include "Camera.hpp"
100016 #include "DirectionalLight.hpp"
100017 #include "IPrimitive.hpp"
100018 #include "PointLight.hpp"
100019
```

```
100020 namespace RayTracer {
100024   class Core {
100025   public:
100029     Core();
100030
100034     ~Core() = default;
100035
100041     int render(std::string configfile);
100042
100043   private:
100047     void render_scene();
100048
100054     Color cast_ray(const Ray& ray);
100055
100063     Color calculate_lighting(const Math::Point3D& hit_point,
100064                              const Math::Vector3D& normal,
100065                              const std::shared_ptr<IPrimitive>& hit_obj);
100066
100073     bool calculate_shadow(const Math::Point3D& hit_point,
100074                           const std::shared_ptr<ILight>& light);
100075
100083     Math::Vector3D calculate_phong(const Math::Point3D& hit_point,
100084                                    const Math::Vector3D& normal,
100085                                    const std::shared_ptr<ILight>& light);
100086
100091     void save_framebuffer(const std::string& filename);
100092
100100     void save_as_ppm(const std::vector<Color>& framebuffer, int width,
100101                      int height, const std::string& filename);
100102
100103     int image_width_;
100104     int image_height_;
100105     std::vector<Color> framebuffer;
100106     Camera cam;
100107     std::vector<std::shared_ptr<IPrimitive» primitives_;
100108     std::vector<std::shared_ptr<ILight» lights_;
100109   };
100110 }  // namespace RayTracer
```

5.3 Exception.hpp

```
100001 /*
100002 ** EPITECH PROJECT, 2025
100003 ** B-OOP-400-NAN-4-1-raytracer-evann.bloutin
100004 ** File description:
100005 ** Exception
100006 */
100007
100008 #pragma once
100009 #include <exception>
100010 #include <string>
100011
100012 namespace RayTracer {
100013
100014   class CoreException : public std::exception {
100015   public:
100016     CoreException(const std::string& message) : message_(message) {}
100017     virtual const char* what() const noexcept override {
100018       return message_.c_str();
100019     }
100020
100021   private:
100022     std::string message_;
100023   };
100024
100025   class FileException : public CoreException {
100026   public:
100027     FileException(const std::string& message)
100028         : CoreException("File Error: " + message) {}
100029   };
100030
100031   class ConfigException : public CoreException {
100032   public:
100033     ConfigException(const std::string& message)
100034         : CoreException("Config Error: " + message) {}
100035   };
100036
100037   class RenderException : public CoreException {
100038   public:
100039     RenderException(const std::string& message)
100040         : CoreException("Render Error: " + message) {}
100041   };
100042
```

```
100043   class PrimitiveException : public CoreException {
100044   public:
100045     PrimitiveException(const std::string& message)
100046         : CoreException("Primitive Error: " + message) {}
100047   };
100048
100049 }  // namespace RayTracer
```

5.4 AmbientLight.hpp

```
100001 /*
100002 ** EPITECH PROJECT, 2025
100003 ** B-OOP-400-NAN-4-1-raytracer-evann.bloutin
100004 ** File description:
100005 ** AmbientLight
100006 */
100007
100008 #pragma once
100009
100010 #include "ILight.hpp"
100011
100012 namespace RayTracer {
100016   class AmbientLight : public ILight {
100017   public:
100023     AmbientLight(const Math::Vector3D& color, double intensity);
100024
100030     Math::Vector3D getDirectionTo(const Math::Point3D& point) const override;
100031
100037     double getDistanceTo(const Math::Point3D& point) const override;
100038
100043     ILight::Type getType() const override;
100044
100049     const Math::Vector3D& getColor() const override;
100050
100055     double getIntensity() const override;
100056
100057   private:
100058     Math::Vector3D color_;
100059     double intensity_;
100060   };
100061
100062 }  // namespace RayTracer
100063
100064 extern "C" {
100069   std::shared_ptr<RayTracer::ILight> entryPoint();
100070
100075   const std::string getName();
100076
100081   const std::string getType();
100082 }
```

5.5 DirectionalLight.hpp

```
100001 /*
100002 ** EPITECH PROJECT, 2025
100003 ** B-OOP-400-NAN-4-1-raytracer-evann.bloutin
100004 ** File description:
100005 ** DirectionalLight
100006 */
100007
100008 #pragma once
100009
100010 #include "ILight.hpp"
100011 #include "Point3D.hpp"
100012 #include "Vector3D.hpp"
100013
100014 namespace RayTracer {
100018   class DirectionalLight : public ILight {
100019   public:
100026     DirectionalLight(const Math::Vector3D& direction,
100027                      const Math::Vector3D& color, double intensity = 1.0);
100028
100034     Math::Vector3D getDirectionTo(const Math::Point3D& point) const override;
100035
100041     double getDistanceTo(const Math::Point3D& point) const override;
100042
100047     Type getType() const override;
100048
100053     const Math::Vector3D& getColor() const override;
```

```
100054
100059     double getIntensity() const override;
100060
100061   private:
100062     Math::Vector3D direction_;
100063     Math::Vector3D color_;
100064     double intensity_;
100065   };
100066
100067 }  // namespace RayTracer
100068
100069 extern "C" {
100074   std::shared_ptr<RayTracer::ILight> entryPoint();
100075
100080   const std::string getName();
100081
100086   const std::string getType();
100087 }
```

5.6 ILight.hpp

```
100001 /*
100002 ** EPITECH PROJECT, 2025
100003 ** B-OOP-400-NAN-4-1-raytracer-evann.bloutin
100004 ** File description:
100005 ** Light
100006 */
100007
100008 #pragma once
100009
100010 #include <memory>
100011 #include <string>
100012 #include "Point3D.hpp"
100013 #include "Vector3D.hpp"
100014
100015 namespace RayTracer {
100019   class ILight {
100020   public:
100024     enum class Type { POINT, DIRECTIONAL, AMBIENT };
100025
100029     virtual ~ILight() = default;
100030
100036     virtual Math::Vector3D getDirectionTo(const Math::Point3D& point) const = 0;
100037
100043     virtual double getDistanceTo(const Math::Point3D& point) const = 0;
100044
100049     virtual Type getType() const = 0;
100050
100055     virtual const Math::Vector3D& getColor() const = 0;
100056
100061     virtual double getIntensity() const = 0;
100062   };
100063
100064 }  // namespace RayTracer
```

5.7 PointLight.hpp

```
100001 /*
100002 ** EPITECH PROJECT, 2025
100003 ** B-OOP-400-NAN-4-1-raytracer-evann.bloutin
100004 ** File description:
100005 ** PointLight
100006 */
100007
100008 #pragma once
100009
100010 #include "ILight.hpp"
100011
100012 namespace RayTracer {
100016   class PointLight : public ILight {
100017   public:
100024     PointLight(const Math::Point3D& position, const Math::Vector3D& color,
100025                double intensity = 1.0);
100026
100032     Math::Vector3D getDirectionTo(const Math::Point3D& point) const override;
100033
100039     double getDistanceTo(const Math::Point3D& point) const override;
100040
100045     Type getType() const override;
```

```
100046
100051     const Math::Vector3D& getColor() const override;
100052
100057     double getIntensity() const override;
100058
100059   private:
100060     Math::Point3D position_;
100061     Math::Vector3D color_;
100062     double intensity_;
100063   };
100064
100065 }  // namespace RayTracer
100066
100067 extern "C" {
100072   std::shared_ptr<RayTracer::ILight> entryPoint();
100073
100078   const std::string getName();
100079
100084   const std::string getType();
100085 }
```

5.8 DLLoader.hpp

```
100001 /*
100002 ** EPITECH PROJECT, 2025
100003 ** B-OOP-400-NAN-4-1-raytracer-evann.bloutin
100004 ** File description:
100005 ** DLLoader
100006 */
100007
100008 #pragma once
100009 #include <dlfcn.h>
100010 #include <memory>
100011 #include <stdexcept>
100012 #include <string>
100013
100018 template <typename T>
100019 class DLLoader {
100020 public:
100025   DLLoader(const std::string& path)
100026       : _handle(nullptr),
100027         _entry(nullptr),
100028         _getName(nullptr),
100029         _getType(nullptr),
100030         _error("") {
100031     _handle = dlopen(path.c_str(), RTLD_LAZY);
100032     if (!_handle) {
100033       _error = std::string("dlopen error: ") + dlerror();
100034       return;
100035     }
100036
100037     _entry = reinterpret_cast<std::shared_ptr<T> (*)()>(
100038         dlsym(_handle, "entryPoint"));
100039     if (!_entry) {
100040       _error = std::string("dlsym entryPoint error: ") + dlerror();
100041       dlclose(_handle);
100042       _handle = nullptr;
100043       return;
100044     }
100045
100046     _getName =
100047         reinterpret_cast<const std::string& (*) ()>(dlsym(_handle, "getName"));
100048     if (!_getName) {
100049       _error = std::string("dlsym getName error: ") + dlerror();
100050       dlclose(_handle);
100051       _handle = nullptr;
100052       _entry = nullptr;
100053       return;
100054     }
100055
100056     _getType =
100057         reinterpret_cast<const std::string& (*) ()>(dlsym(_handle, "getType"));
100058     if (!_getType) {
100059       _error = std::string("dlsym getType error: ") + dlerror();
100060       dlclose(_handle);
100061       _handle = nullptr;
100062       _entry = nullptr;
100063     }
100064   }
100065
100069   ~DLLoader() {
100070     if (_handle)
100071       dlclose(_handle);
```

```
100072   }
100073
100078   bool isValid() const { return _handle && _entry && _getName && _getType; }
100079
100085   std::shared_ptr<T> getInstance() const {
100086     if (!isValid())
100087       throw std::runtime_error("DLLoader: Plugin is not valid.");
100088     return _entry();
100089   }
100090
100096   const std::string& getName() const {
100097     if (!isValid())
100098       throw std::runtime_error("DLLoader: Plugin is not valid.");
100099     return _getName();
100100   }
100101
100107   const std::string& getType() const {
100108     if (!isValid())
100109       throw std::runtime_error("DLLoader: Plugin is not valid.");
100110     return _getType();
100111   }
100112
100117   std::string getError() const { return _error; }
100118
100119 private:
100120   void* _handle;
100121   std::shared_ptr<T> (*_entry)();
100122   const std::string& (*_getName)();
100123   const std::string& (*_getType)();
100124   std::string _error;
100125 };
```

5.9 Parser.hpp

```
100001 /*
100002 ** EPITECH PROJECT, 2024
100003 ** raytracer
100004 ** File description:
100005 ** Parser.hpp
100006 */
100007
100008 #pragma once
100009
100010 #include <algorithm>
100011 #include <fstream>
100012 #include <iostream>
100013 #include <libconfig.h++>
100014 #include <sstream>
100015 #include <string>
100016 #include <vector>
100017 #include "AmbientLight.hpp"
100018 #include "Camera.hpp"
100019 #include "Cone.hpp"
100020 #include "Cylinder.hpp"
100021 #include "DirectionalLight.hpp"
100022 #include "Matrix.hpp"
100023 #include "Mesh.hpp"
100024 #include "Plane.hpp"
100025 #include "Rectangle3D.hpp"
100026 #include "Sphere.hpp"
100027
100028 namespace Parser {
100032   class Parser {
100033   public:
100037     Parser() : cam_(Math::Point3D(0, 9, -55), 72.0, 1920, 1080) {}
100038
100042     ~Parser() = default;
100043
100049     int parse(const std::string& filePath);
100050
100055     void parseCamera(const libconfig::Setting& root);
100056
100061     void parseLight(const libconfig::Setting& root);
100062
100067     void parseDirectionalLight(const libconfig::Setting& lights);
100068
100073     void parsePrimitives(const libconfig::Setting& root);
100074
100080     void parseObjects(const libconfig::Setting& primitives, std::string type);
100081
100086     void parseMesh(const libconfig::Setting& root);
100087
100092     RayTracer::Camera getCamera() const { return cam_; }
```

```
100093
100098     std::vector<std::shared_ptr<RayTracer::IPrimitive» getPrimitives() const {
100099       return primitives_;
100100     }
100101
100106     std::vector<std::shared_ptr<RayTracer::ILight» getLights() const {
100107       return lights_;
100108     }
100109
100110   private:
100111     std::string _filePath;
100112     RayTracer::Camera cam_;
100113     std::vector<std::shared_ptr<RayTracer::IPrimitive» primitives_;
100114     std::vector<std::shared_ptr<RayTracer::ILight» lights_;
100115   };
100116 }  // namespace Parser
```

5.10 Cone.hpp

```
100001 /*
100002 ** EPITECH PROJECT, 2025
100003 ** project
100004 ** File description:
100005 ** Cone
100006 */
100007
100008 #pragma once
100009
100010 #include "IPrimitive.hpp"
100011 #include "Point3D.hpp"
100012 #include "Vector3D.hpp"
100013
100014 #include <memory>
100015
100016 namespace RayTracer {
100020   class Cone : public IPrimitive {
100021   public:
100029     Cone(const Math::Point3D& base, double radius, double height,
100030          const Math::Vector3D& color);
100031
100035     ~Cone() = default;
100036
100043     bool hits(const Ray& ray, double& t) const override;
100044
100050     Math::Vector3D getNormal(const Math::Point3D& point) const override;
100051
100056     Math::Vector3D getColor() const override;
100057
100058   private:
100059     Math::Point3D base_;
100060     Math::Vector3D normal_;
100061     double radius_;
100062     double height_;
100063     Math::Vector3D color_;
100064   };
100065 }  // namespace RayTracer
```

5.11 Cylinder.hpp

```
100001 /*
100002 ** EPITECH PROJECT, 2025
100003 ** project
100004 ** File description:
100005 ** Cylinder
100006 */
100007
100008 #pragma once
100009
100010 #include "IPrimitive.hpp"
100011 #include "Point3D.hpp"
100012 #include "Vector3D.hpp"
100013
100014 #include <memory>
100015
100016 namespace RayTracer {
100020   class Cylinder : public IPrimitive {
100021   public:
100029     Cylinder(const Math::Point3D& base, double radius, double height,
100030              const Math::Vector3D& color);
```

```
100031
100035     ~Cylinder() = default;
100036
100043     bool hits(const Ray& ray, double& t) const override;
100044
100050     Math::Vector3D getNormal(const Math::Point3D& point) const override;
100051
100056     Math::Vector3D getColor() const override;
100057
100058   private:
100059     Math::Point3D base_;
100060     Math::Vector3D normal_;
100061     double radius_;
100062     double height_;
100063     Math::Vector3D color_;
100064   };
100065 }  // namespace RayTracer
100066
100067 extern "C" {
100072   std::shared_ptr<RayTracer::IPrimitive> entryPoint();
100073
100078   const std::string getName();
100079
100084   const std::string getType();
100085 }
```

5.12 IPrimitive.hpp

```
100001 /*
100002 ** EPITECH PROJECT, 2025
100003 ** B-OOP-400-NAN-4-1-raytracer-evann.bloutin
100004 ** File description:
100005 ** IPrimitive
100006 */
100007
100008 #pragma once
100009
100010 #include "Ray.hpp"
100011
100012 namespace RayTracer {
100016   class IPrimitive {
100017   public:
100021     virtual ~IPrimitive() = default;
100022
100029     virtual bool hits(const Ray& ray, double& t) const = 0;
100030
100036     virtual Math::Vector3D getNormal(const Math::Point3D& point) const = 0;
100037
100042     virtual Math::Vector3D getColor() const = 0;
100043   };
100044 }  // namespace RayTracer
```

5.13 Mesh.hpp

```
100001 /*
100002 ** EPITECH PROJECT, 2025
100003 ** project
100004 ** File description:
100005 ** Mesh - OBJ File loader
100006 */
100007
100008 #pragma once
100009
100010 #include "IPrimitive.hpp"
100011 #include "Matrix.hpp"
100012 #include "Point3D.hpp"
100013 #include "Vector3D.hpp"
100014
100015 #include <array>
100016 #include <memory>
100017 #include <string>
100018 #include <vector>
100019
100020 namespace RayTracer {
100026   class Mesh : public IPrimitive {
100027   public:
100035     Mesh(const std::string& filename, const Math::Vector3D& color,
100036          const Math::Point3D& position = Math::Point3D(0, 0, 0),
100037          double scale = 1.0);
```

```
100038
100042     ~Mesh() = default;
100043
100050     bool hits(const Ray& ray, double& t) const override;
100051
100057     Math::Vector3D getNormal(const Math::Point3D& point) const override;
100058
100063     Math::Vector3D getColor() const override;
100064
100069     void setPosition(const Math::Point3D& position) { position_ = position; }
100070
100075     void setScale(double scale) { scale_ = scale; }
100076
100077   private:
100083     bool loadFromFile(const std::string& filename);
100084
100090     Math::Point3D transformPoint(const Math::Vector3D& point) const;
100091
100095     struct Face {
100096       size_t v1, v2, v3;
100097     };
100098
100099     std::vector<Math::Vector3D> vertices_;
100100     std::vector<Face> faces_;
100101     Math::Vector3D color_;
100102     Math::Point3D position_;
100103     double scale_;
100104     mutable size_t lastHitFace_;
100105   };
100106 }  // namespace RayTracer
100107
100108 extern "C" {
100113   std::shared_ptr<RayTracer::IPrimitive> entryPoint();
100114
100119   const std::string getName();
100120
100125   const std::string getType();
100126 }
```

5.14 Plane.hpp

```
100001 /*
100002 ** EPITECH PROJECT, 2025
100003 ** B-OOP-400-NAN-4-1-raytracer-evann.bloutin
100004 ** File description:
100005 ** Plane
100006 */
100007
100008 #pragma once
100009
100010 #include "Core.hpp"
100011 #include "IPrimitive.hpp"
100012 #include "Point3D.hpp"
100013 #include "Ray.hpp"
100014 #include "Vector3D.hpp"
100015
100016 #include <memory>
100017 #include <string>
100018
100019 namespace RayTracer {
100023   class Plane : public IPrimitive {
100024   public:
100028     Plane() = default;
100029
100036     Plane(const std::string& axis, const Math::Point3D& position,
100037           const Math::Vector3D& color);
100038
100042     ~Plane() = default;
100043
100050     bool hits(const Ray& ray, double& t) const override;
100051
100057     Math::Vector3D getNormal(const Math::Point3D& point) const override;
100058
100063     Math::Vector3D getColor() const override;
100064
100070     Color getColorAt(const Math::Point3D& point) const;
100071
100072   private:
100073     std::string axis_;
100074     Math::Point3D position_;
100075     Math::Vector3D normal_;
100076     Math::Vector3D color_;
100077   };
100078 }  // namespace RayTracer
```

5.15 Sphere.hpp

```
100001 /*
100002 ** EPITECH PROJECT, 2025
100003 ** project
100004 ** File description:
100005 ** Sphere
100006 */
100007
100008 #pragma once
100009
100010 #include "IPrimitive.hpp"
100011 #include "Point3D.hpp"
100012
100013 #include <memory>
100014 #include <string>
100015
100016 namespace RayTracer {
100020   class Sphere : public IPrimitive {
100021   public:
100025     Sphere() = default;
100026
100033     Sphere(const Math::Point3D& center, double radius,
100034            const Math::Vector3D& color);
100035
100039     ~Sphere() = default;
100040
100047     bool hits(const Ray& ray, double& t) const override;
100048
100054     Math::Vector3D getNormal(const Math::Point3D& point) const override;
100055
100060     Math::Vector3D getColor() const override;
100061
100062   private:
100063     Math::Point3D center_;
100064     double radius_;
100065     Math::Vector3D color_;
100066   };
100067 }  // namespace RayTracer
```

5.16 Matrix.hpp

```
100001 /*
100002 ** EPITECH PROJECT, 2025
100003 ** B-OOP-400-NAN-4-1-raytracer-evann.bloutin
100004 ** File description:
100005 ** Matrix
100006 */
100007
100008 #pragma once
100009
100010 #include "Point3D.hpp"
100011 #include "Vector3D.hpp"
100012
100013 #include <array>
100014 #include <cmath>
100015 #include <iostream>
100016 #include <stdexcept>
100017
100018 namespace Math {
100022   class Matrix {
100023   public:
100027     Matrix();
100028
100033     static Matrix identity();
100034
100042     static Matrix translation(double tx, double ty, double tz);
100043
100051     static Matrix scale(double sx, double sy, double sz);
100052
100058     static Matrix rotationZ(double angle);
100059
100065     Matrix rotationX(double angle);
100066
100072     Matrix rotationY(double angle);
100073
100079     Point3D transform(const Point3D& p) const;
100080
100086     Vector3D transform(const Vector3D& v) const;
100087
100093     Matrix operator*(const Matrix& other) const;
100094
100102     double& elementAt(int row, int col);
```

```
100103
100111     double elementAt(int row, int col) const;
100112
100116     void print() const;
100117
100118   private:
100119     std::array<std::array<double, 4>, 4> data;
100120   };
100121
100122 }  // namespace Math
```

5.17 Point3D.hpp

```
100001 /*
100002 ** EPITECH PROJECT, 2025
100003 ** B-OOP-400-NAN-4-1-bsraytracer-robin.schuffenecker
100004 ** File description:
100005 ** Point3D
100006 */
100007
100008 #pragma once
100009
100010 #include "Vector3D.hpp"
100011
100012 namespace Math {
100016   class Point3D {
100017   public:
100021     Point3D();
100022
100029     Point3D(double x, double y, double z);
100030
100034     ~Point3D() = default;
100035
100039     double x;
100040
100044     double y;
100045
100049     double z;
100050
100056     Point3D operator+(const Vector3D& vec) const;
100057
100063     Point3D& operator+=(const Vector3D& vec);
100064
100070     Vector3D operator-(const Point3D& point) const;
100071
100077     Point3D operator-(const Vector3D& vec) const;
100078   };
100079 }  // namespace Math
```

5.18 Ray.hpp

```
100001 /*
100002 ** EPITECH PROJECT, 2025
100003 ** project
100004 ** File description:
100005 ** Ray
100006 */
100007
100008 #pragma once
100009 #include "Point3D.hpp"
100010 #include "Vector3D.hpp"
100011
100012 namespace RayTracer {
100016   class Ray {
100017   public:
100021     Ray() = default;
100022
100028     Ray(const Math::Point3D& origin, const Math::Vector3D& direction);
100029
100033     ~Ray() = default;
100034
100038     Math::Point3D origin;
100039
100043     Math::Vector3D direction;
100044
100049     Math::Point3D getOrigin() const;
100050
100055     Math::Vector3D getDirection() const;
100056   };
100057 }  // namespace RayTracer
```

5.19 Rectangle3D.hpp

```
100001 /*
100002 ** EPITECH PROJECT, 2025
100003 ** B-OOP-400-NAN-4-1-bsraytracer-robin.schuffenecker
100004 ** File description:
100005 ** Rectangle3D
100006 */
100007
100008 #pragma once
100009 #include "Point3D.hpp"
100010 #include "Vector3D.hpp"
100011
100012 namespace Math {
100016   class Rectangle3D {
100017   public:
100021     Rectangle3D() = default;
100022
100029     Rectangle3D(const Point3D& origin, const Vector3D& bottom_side,
100030                 const Vector3D& left_side);
100031
100035     ~Rectangle3D() = default;
100036
100040     Point3D origin;
100041
100045     Vector3D bottom_side;
100046
100050     Vector3D left_side;
100051
100058     Point3D pointAt(double u, double v) const;
100059   };
100060 }  // namespace Math
```

5.20 Vector3D.hpp

```
100001 /*
100002 ** EPITECH PROJECT, 2025
100003 ** B-OOP-400-NAN-4-1-bsraytracer-robin.schuffenecker
100004 ** File description:
100005 ** Main
100006 */
100007
100008 #pragma once
100009
100010 #include <cmath>
100011
100012 namespace Math {
100016   class Vector3D {
100017   public:
100021     Vector3D();
100022
100029     Vector3D(double x, double y, double z);
100030
100034     ~Vector3D() = default;
100035
100036     double x;
100037     double y;
100038     double z;
100039
100044     double length() const;
100045
100046     Vector3D operator+(const Vector3D& other) const;
100047     Vector3D operator-(const Vector3D& other) const;
100048     Vector3D operator*(const Vector3D& other) const;
100049     Vector3D operator/(const Vector3D& other) const;
100050     Vector3D operator+=(const Vector3D& other);
100051     Vector3D operator-=(const Vector3D& other);
100052     Vector3D operator*=(const Vector3D& other);
100053     Vector3D operator/=(const Vector3D& other);
100054
100055     Vector3D operator*(double scalar) const;
100056     Vector3D operator/(double scalar) const;
100057     Vector3D operator*=(double scalar);
100058     Vector3D operator/=(double scalar);
100059
100060     Vector3D operator-() const;
100061
100067     double dot(const Vector3D& other) const;
100068
100074     Vector3D cross(const Vector3D& other) const;
100075
100080     Vector3D normalized() const;
100081   };
```

```
100082
100089   Vector3D operator*(double scalar, const Vector3D& vec);
100090
100091   template <std::size_t N>
100092   class Vector {
100093   public:
100094     Vector() : components{} {}
100095     explicit Vector(const std::array<double, N>& components)
100096         : components(components) {}
100097     ~Vector() = default;
100098
100099   private:
100100     std::array<double, N> components;
100101   };
100102 }  // namespace Math
```