RayTracer

Generated by Doxygen 1.9.7

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

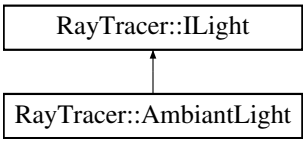
Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 RayTracer::AmbiantLight Class Reference

Inheritance diagram for RayTracer::AmbiantLight:



Public Member Functions

- **AmbiantLight** (const Color &color)
- virtual Color getAmbientColor (const RayTracer::Material &material) const override
- virtual Color getDiffuseColor (const Math::Vector3D &point, const Math::Vector3D &normal, const RayTracer::Material &material) const override
- virtual Math::Vector3D getDirection (const Math::Vector3D &point) const override
- virtual double getDistance (const Math::Vector3D &point) const override
- virtual Color getSpecularColor (const Math::Vector3D &point, const Math::Vector3D &viewDirection, const Math::Vector3D &normal, const RayTracer::Material &material) const override

- virtual Color **getAmbientColor** (const RayTracer::Material &material) const =0
- virtual Color **getDiffuseColor** (const Math::Vector3D &point, const Math::Vector3D &normal, const RayTracer::Material &material) const =0
- virtual Math::Vector3D **getDirection** (const Math::Vector3D &point) const =0
- virtual double **getDistance** (const Math::Vector3D &point) const =0
- virtual Color **getSpecularColor** (const Math::Vector3D &point, const Math::Vector3D &viewDirection, const Math::Vector3D &normal, const RayTracer::Material &material) const =0

Public Attributes

- Color **color_**

4.1.1 Member Function Documentation

4.1.1.1 getAmbientColor()

```
virtual Color RayTracer::AmbiantLight::getAmbientColor (
            const RayTracer::Material & material ) const  [inline], [override], [virtual]
```

Implements RayTracer::ILight.

4.1.1.2 getDiffuseColor()

```
virtual Color RayTracer::AmbiantLight::getDiffuseColor (
            const Math::Vector3D & point,
            const Math::Vector3D & normal,
            const RayTracer::Material & material ) const  [inline], [override], [virtual]
```

Implements RayTracer::ILight.

4.1.1.3 getDirection()

```
virtual Math::Vector3D RayTracer::AmbiantLight::getDirection (
            const Math::Vector3D & point ) const  [inline], [override], [virtual]
```

Implements RayTracer::ILight.

4.1.1.4 getDistance()

```
virtual double RayTracer::AmbiantLight::getDistance (
            const Math::Vector3D & point ) const  [inline], [override], [virtual]
```

Implements RayTracer::ILight.

4.1.1.5 getSpecularColor()

```
virtual Color RayTracer::AmbiantLight::getSpecularColor (
            const Math::Vector3D & point,
            const Math::Vector3D & viewDirection,
            const Math::Vector3D & normal,
            const RayTracer::Material & material ) const  [inline], [override], [virtual]
```

Implements RayTracer::ILight.

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

- include/AmbientLight.hpp

4.2 RayTracer::Config::AmbientLight_t Struct Reference

Public Attributes

- Color **color_**

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

- include/Config.hpp

4.3 RayTracer::Camera Class Reference

Public Member Functions

- **Camera** (const Math::Vector3D &origin_, const Rectangle3D &screen_)
- Ray **getRay** (double u, double v) const
- void **getCamera** (const std::string &file)
- RayTracer::Camera **setCamera** ()

Public Attributes

- Math::Vector3D **origin**
- Rectangle3D **screen**
- int **screen_width**
- int **screen_height**
- Math::Vector3D **camera_pos**
- Math::Vector3D **camera_rotation**
- double **camera_fov**

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

- include/Camera.hpp
- src/Camera.cpp

4.4 RayTracer::Color Class Reference

Public Member Functions

- **Color** (double red, double green, double blue)
- double **red** () const
- double **green** () const
- double **blue** () const
- Color **operator+** (const Color &c) const
- Color **operator*** (double k) const
- Color **operator*** (const Color &c) const
- Color **operator/** (double scalar) const
- Color **getColor** () const
- Color & **operator+=** (const Color &other)
- void **clamp** ()
- void **print** () const
- void **toRGB** ()

Public Attributes

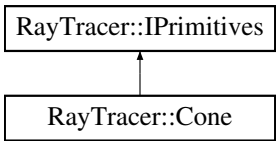
- double **r**
- double **g**
- double **b**

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

- include/Color.hpp

4.5 RayTracer::Cone Class Reference

Inheritance diagram for RayTracer::Cone:



Public Member Functions

- **Cone** (Math::Vector3D cone_pos, double radius, double height)
- std::optional< double > hits (const Ray &ray) const override
- Math::Vector3D getNormal (const Math::Vector3D &hit_point) const

- virtual std::optional< double > **hits** (const Ray &ray) const =0
- virtual Math::Vector3D **getNormal** (const Math::Vector3D &position) const =0

Public Attributes

- Math::Vector3D **cone_pos**
- double **radius**
- double **height**

Public Attributes inherited from RayTracer::IPrimitives

- double **distance**

4.5.1 Member Function Documentation

4.5.1.1 getNormal()

```
Math::Vector3D RayTracer::Cone::getNormal (
            const Math::Vector3D & hit_point ) const  [inline], [virtual]
```

Implements RayTracer::IPrimitives.

4.5.1.2 hits()

```
std::optional< double > RayTracer::Cone::hits (
            const Ray & ray ) const  [inline], [override], [virtual]
```

Implements RayTracer::IPrimitives.

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

- include/Cone.hpp

4.6 RayTracer::Config::Cone_t Struct Reference

Public Attributes

- Math::Vector3D **cone_pos**
- double **radius**
- double **height**

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

- include/Config.hpp

4.7 RayTracer::Config Class Reference

Classes

- struct AmbientLight_t
- struct Cone_t
- struct Cylinder_t
- struct DiffuseLight_t
- struct DirectionalLight_t
- struct Material_t
- struct Plane_t
- struct PointLight_t
- struct Sphere_t
- struct Torus_t

Public Member Functions

- void **getPrimitives** (const std::string &file)
- void **getLights** (const std::string &file)
- void **getMaterials** (const std::string &file)
- void **setPrimitives** ()
- void **setMaterials** ()
- void **setLights** ()

Public Attributes

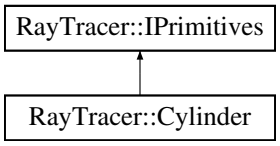
- std::vector< Sphere_t > **spheres**
- std::vector< Plane_t > **planes**
- std::vector< Cylinder_t > **cylinders**
- std::vector< Cone_t > **cones**
- std::vector< Torus_t > **toruses**
- std::vector< Material_t > **materials**
- std::vector< PointLight_t > **pointLights**
- std::vector< DirectionalLight_t > **directionalLights**
- std::vector< DiffuseLight_t > **diffuseLights**
- std::vector< AmbientLight_t > **ambientLights**
- std::vector< std::shared_ptr< IPrimitives > > **primitives**
- std::vector< std::shared_ptr< ILight > > **lights**
- std::vector< std::shared_ptr< Material > > **material**

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

- include/Config.hpp
- src/Config.cpp

4.8 RayTracer::Cylinder Class Reference

Inheritance diagram for RayTracer::Cylinder:



Public Member Functions

- **Cylinder** (Math::Vector3D cylinder_pos, double radius, double height)
- std::optional< double > hits (const Ray &ray) const override
- Math::Vector3D getNormal (const Math::Vector3D &point) const override

- virtual std::optional< double > **hits** (const Ray &ray) const =0
- virtual Math::Vector3D **getNormal** (const Math::Vector3D &position) const =0

Public Attributes

- Math::Vector3D **cylinder_pos**
- double **radius**
- double **height**

Public Attributes inherited from RayTracer::IPrimitives

- double **distance**

4.8.1 Member Function Documentation

4.8.1.1 getNormal()

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

Implements RayTracer::IPrimitives.

4.8.1.2 hits()

```
std::optional< double > RayTracer::Cylinder::hits (
            const Ray & ray ) const  [inline], [override], [virtual]
```

Implements RayTracer::IPrimitives.

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

- include/Cylinder.hpp

4.9 RayTracer::Config::Cylinder_t Struct Reference

Public Attributes

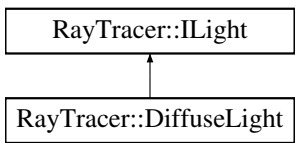
- Math::Vector3D **cylinder_pos**
- double **radius**
- double **height**

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

- include/Config.hpp

4.10 RayTracer::DiffuseLight Class Reference

Inheritance diagram for RayTracer::DiffuseLight:



Public Member Functions

- **DiffuseLight** (Color color, Math::Vector3D direction, double intensity)
- Color getAmbientColor (const RayTracer::Material &material) const override
- Color getDiffuseColor (const Math::Vector3D &point, const Math::Vector3D &normal, const RayTracer::Material &material) const override
- Math::Vector3D getDirection (const Math::Vector3D &point) const override
- double getDistance (const Math::Vector3D &point) const override
- virtual Color getSpecularColor (const Math::Vector3D &point, const Math::Vector3D &viewDirection, const Math::Vector3D &normal, const RayTracer::Material &material) const override

- virtual Color **getAmbientColor** (const RayTracer::Material &material) const =0
- virtual Color **getDiffuseColor** (const Math::Vector3D &point, const Math::Vector3D &normal, const RayTracer::Material &material) const =0
- virtual Math::Vector3D **getDirection** (const Math::Vector3D &point) const =0
- virtual double **getDistance** (const Math::Vector3D &point) const =0
- virtual Color **getSpecularColor** (const Math::Vector3D &point, const Math::Vector3D &viewDirection, const Math::Vector3D &normal, const RayTracer::Material &material) const =0

4.10.1 Member Function Documentation

4.10.1.1 getAmbientColor()

```
Color RayTracer::DiffuseLight::getAmbientColor (
            const RayTracer::Material & material ) const  [inline], [override], [virtual]
```

Implements RayTracer::ILight.

4.10.1.2 getDiffuseColor()

```
Color RayTracer::DiffuseLight::getDiffuseColor (
            const Math::Vector3D & point,
            const Math::Vector3D & normal,
            const RayTracer::Material & material ) const  [inline], [override], [virtual]
```

Implements RayTracer::ILight.

4.10.1.3 getDirection()

```
Math::Vector3D RayTracer::DiffuseLight::getDirection (
            const Math::Vector3D & point ) const  [inline], [override], [virtual]
```

Implements RayTracer::ILight.

4.10.1.4 getDistance()

```
double RayTracer::DiffuseLight::getDistance (
            const Math::Vector3D & point ) const  [inline], [override], [virtual]
```

Implements RayTracer::ILight.

4.10.1.5 getSpecularColor()

```
virtual Color RayTracer::DiffuseLight::getSpecularColor (
            const Math::Vector3D & point,
            const Math::Vector3D & viewDirection,
            const Math::Vector3D & normal,
            const RayTracer::Material & material ) const  [inline], [override], [virtual]
```

Implements RayTracer::ILight.

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

- include/DiffuseLight.hpp

4.11 RayTracer::Config::DiffuseLight_t Struct Reference

Public Attributes

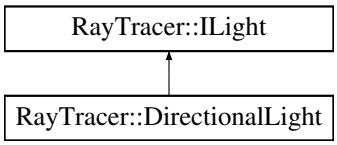
- Color **color_**
- Math::Vector3D **direction_**
- double **intensity_**

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

- include/Config.hpp

4.12 RayTracer::DirectionalLight Class Reference

Inheritance diagram for RayTracer::DirectionalLight:



Public Member Functions

- **DirectionalLight** (const Math::Vector3D &direction, const Color &color, double intensity)
- Color getAmbientColor (const Material &material) const override
- Color getDiffuseColor (const Math::Vector3D &point, const Math::Vector3D &normal, const Material &material) const override
- Math::Vector3D getDirection (const Math::Vector3D &point) const override
- double getDistance (const Math::Vector3D &point) const override
- Color getSpecularColor (const Math::Vector3D &point, const Math::Vector3D &viewDirection, const Math::Vector3D &normal, const Material &material) const override

- virtual Color **getAmbientColor** (const RayTracer::Material &material) const =0
- virtual Color **getDiffuseColor** (const Math::Vector3D &point, const Math::Vector3D &normal, const RayTracer::Material &material) const =0
- virtual Math::Vector3D **getDirection** (const Math::Vector3D &point) const =0
- virtual double **getDistance** (const Math::Vector3D &point) const =0
- virtual Color **getSpecularColor** (const Math::Vector3D &point, const Math::Vector3D &viewDirection, const Math::Vector3D &normal, const RayTracer::Material &material) const =0

4.12.1 Member Function Documentation

4.12.1.1 getAmbientColor()

```
Color RayTracer::DirectionalLight::getAmbientColor (
            const Material & material ) const  [inline], [override], [virtual]
```

Implements RayTracer::ILight.

4.12.1.2 getDiffuseColor()

```
Color RayTracer::DirectionalLight::getDiffuseColor (
            const Math::Vector3D & point,
            const Math::Vector3D & normal,
            const Material & material ) const  [inline], [override], [virtual]
```

Implements RayTracer::ILight.

4.12.1.3 getDirection()

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

Implements RayTracer::ILight.

4.12.1.4 getDistance()

```
double RayTracer::DirectionalLight::getDistance (
            const Math::Vector3D & point ) const  [inline], [override], [virtual]
```

Implements RayTracer::ILight.

4.12.1.5 getSpecularColor()

```
Color RayTracer::DirectionalLight::getSpecularColor (
            const Math::Vector3D & point,
            const Math::Vector3D & viewDirection,
            const Math::Vector3D & normal,
            const Material & material ) const  [inline], [override], [virtual]
```

Implements RayTracer::ILight.

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

- include/DirectionalLight.hpp

4.13 RayTracer::Config::DirectionalLight_t Struct Reference

Public Attributes

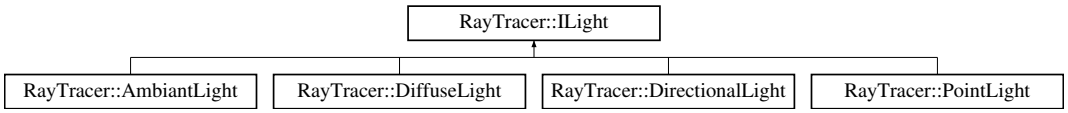
- Math::Vector3D **direction_**
- Color **color_**
- double **intensity**

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

- include/Config.hpp

4.14 RayTracer::ILight Class Reference

Inheritance diagram for RayTracer::ILight:



Public Member Functions

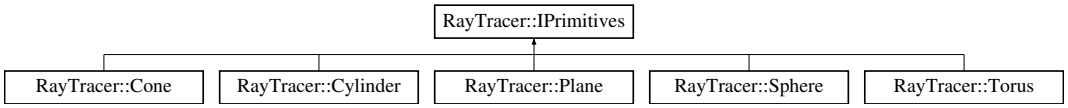
- virtual Color **getAmbientColor** (const RayTracer::Material &material) const =0
- virtual Color **getDiffuseColor** (const Math::Vector3D &point, const Math::Vector3D &normal, const RayTracer::Material &material) const =0
- virtual Math::Vector3D **getDirection** (const Math::Vector3D &point) const =0
- virtual double **getDistance** (const Math::Vector3D &point) const =0
- virtual Color **getSpecularColor** (const Math::Vector3D &point, const Math::Vector3D &viewDirection, const Math::Vector3D &normal, const RayTracer::Material &material) const =0

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

- include/ILight.hpp

4.15 RayTracer::IPrimitives Class Reference

Inheritance diagram for RayTracer::IPrimitives:



Public Member Functions

- virtual std::optional< double > **hits** (const Ray &ray) const =0
- virtual Math::Vector3D **getNormal** (const Math::Vector3D &position) const =0

Public Attributes

- double **distance**

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

- include/IPrimitives.hpp

4.16 RayTracer::Material Class Reference

Public Member Functions

- **Material** (const Color &color, double ambient, double diffuse, double specular, double shininess)
- Color **color** () const
- double **ambient** () const
- double **diffuse** () const
- double **specular** () const
- double **shininess** () const

Public Attributes

- Color **color_**
- double **ambient_**
- double **diffuse_**
- double **specular_**
- double **shininess_**

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

- include/Material.hpp

4.17 RayTracer::Config::Material_t Struct Reference

Public Attributes

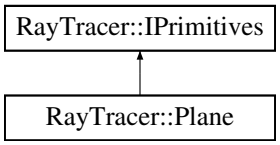
- Color **color_**
- int **sphere_r**
- int **sphere_g**
- int **sphere_b**
- double **ambient_**
- double **diffuse_**
- double **specular_**
- double **shininess_**

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

- include/Config.hpp

4.18 RayTracer::Plane Class Reference

Inheritance diagram for RayTracer::Plane:



Public Member Functions

- **Plane** (Math::Vector3D normal, Math::Vector3D position)
- std::optional< double > hits (const Ray &ray) const override
- Math::Vector3D getNormal (const Math::Vector3D &hit_point) const override

- virtual std::optional< double > **hits** (const Ray &ray) const =0
- virtual Math::Vector3D **getNormal** (const Math::Vector3D &position) const =0

Public Attributes

- Math::Vector3D **normal**
- Math::Vector3D **position**
- double **distance**

Public Attributes inherited from RayTracer::IPrimitives

- double **distance**

4.18.1 Member Function Documentation

4.18.1.1 getNormal()

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

Implements RayTracer::IPrimitives.

4.18.1.2 hits()

```
std::optional< double > RayTracer::Plane::hits (
            const Ray & ray ) const  [inline], [override], [virtual]
```

Implements RayTracer::IPrimitives.

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

- include/Plane.hpp

4.19 RayTracer::Config::Plane_t Struct Reference

Public Attributes

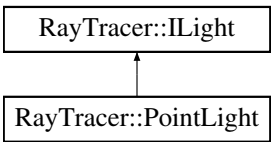
- Math::Vector3D **plane_pos**
- Math::Vector3D **plane_normal**

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

- include/Config.hpp

4.20 RayTracer::PointLight Class Reference

Inheritance diagram for RayTracer::PointLight:



Public Member Functions

- **PointLight** (const Math::Vector3D &position, const Color &color)
- virtual Color getAmbientColor (const RayTracer::Material &material) const override
- virtual Color getDiffuseColor (const Math::Vector3D &point, const Math::Vector3D &normal, const RayTracer::Material &material) const override
- virtual Math::Vector3D getDirection (const Math::Vector3D &point) const override
- virtual double getDistance (const Math::Vector3D &point) const override
- virtual Color getSpecularColor (const Math::Vector3D &point, const Math::Vector3D &viewDirection, const Math::Vector3D &normal, const RayTracer::Material &material) const override

- virtual Color **getAmbientColor** (const RayTracer::Material &material) const =0
- virtual Color **getDiffuseColor** (const Math::Vector3D &point, const Math::Vector3D &normal, const RayTracer::Material &material) const =0
- virtual Math::Vector3D **getDirection** (const Math::Vector3D &point) const =0
- virtual double **getDistance** (const Math::Vector3D &point) const =0
- virtual Color **getSpecularColor** (const Math::Vector3D &point, const Math::Vector3D &viewDirection, const Math::Vector3D &normal, const RayTracer::Material &material) const =0

Public Attributes

- Math::Vector3D **position_**
- Color **color_**
- double **attenuationConstant_** = 1.0
- double **attenuationLinear_** = 0.0045
- double **attenuationQuadratic_** = 0.0075

4.20.1 Member Function Documentation

4.20.1.1 getAmbientColor()

```
virtual Color RayTracer::PointLight::getAmbientColor (
            const RayTracer::Material & material ) const  [inline], [override], [virtual]
```

Implements RayTracer::ILight.

4.20.1.2 getDiffuseColor()

```
virtual Color RayTracer::PointLight::getDiffuseColor (
            const Math::Vector3D & point,
            const Math::Vector3D & normal,
            const RayTracer::Material & material ) const  [inline], [override], [virtual]
```

Implements RayTracer::ILight.

4.20.1.3 getDirection()

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

Implements RayTracer::ILight.

4.20.1.4 getDistance()

```
virtual double RayTracer::PointLight::getDistance (
            const Math::Vector3D & point ) const  [inline], [override], [virtual]
```

Implements RayTracer::ILight.

4.20.1.5 getSpecularColor()

```
virtual Color RayTracer::PointLight::getSpecularColor (
            const Math::Vector3D & point,
            const Math::Vector3D & viewDirection,
            const Math::Vector3D & normal,
            const RayTracer::Material & material ) const  [inline], [override], [virtual]
```

Implements RayTracer::ILight.

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

- include/PointLight.hpp

4.21 RayTracer::Config::PointLight_t Struct Reference

Public Attributes

- Math::Vector3D **position_**
- Color **color_**

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

- include/Config.hpp

4.22 RayTracer::Ray Class Reference

Public Member Functions

- **Ray** (const Math::Vector3D &origin, const Math::Vector3D &direction)
- **Ray** (const Ray &other)=default
- **Ray** (Ray &&other) noexcept=default
- Ray & **operator=** (const Ray &other)=default
- Ray & **operator=** (Ray &&other) noexcept=default
- Math::Vector3D **at** (double t) const

Public Attributes

- Math::Vector3D **origin**
- Math::Vector3D **direction**

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

- include/Ray.hpp

4.23 RayTracer::Rectangle3D Class Reference

Public Member Functions

- **Rectangle3D** (const Math::Vector3D &origin_, const Math::Vector3D &bottom_side_, const Math::Vector3D &left_side_)
- Math::Vector3D **pointAt** (double u, double v) const

Public Attributes

- Math::Vector3D **origin**
- Math::Vector3D **bottom_side**
- Math::Vector3D **left_side**

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

- include/Rectangle3D.hpp

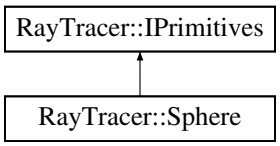
4.24 RayTracer::Scene Class Reference

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

- include/Scene.hpp

4.25 RayTracer::Sphere Class Reference

Inheritance diagram for RayTracer::Sphere:



Public Member Functions

- **Sphere** (const Math::Vector3D &center, double radius)
- Math::Vector3D **center** () const
- double **radius** () const
- std::optional< double > hits (const Ray &ray) const override
- Math::Vector3D getNormal (const Math::Vector3D &position) const

- virtual std::optional< double > **hits** (const Ray &ray) const =0
- virtual Math::Vector3D **getNormal** (const Math::Vector3D &position) const =0

Public Attributes

- double **distance**

Public Attributes inherited from RayTracer::IPrimitives

- double **distance**

4.25.1 Member Function Documentation

4.25.1.1 getNormal()

```
Math::Vector3D RayTracer::Sphere::getNormal (
            const Math::Vector3D & position ) const  [inline], [virtual]
```

Implements RayTracer::IPrimitives.

4.25.1.2 hits()

```
std::optional< double > RayTracer::Sphere::hits (
            const Ray & ray ) const  [inline], [override], [virtual]
```

Implements RayTracer::IPrimitives.

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

- include/Sphere.hpp

4.26 RayTracer::Config::Sphere_t Struct Reference

Public Attributes

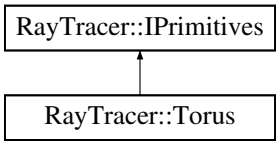
- Math::Vector3D **sphere_pos**
- double **radius**
- int **sphere_r**
- int **sphere_g**
- int **sphere_b**

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

- include/Config.hpp

4.27 RayTracer::Torus Class Reference

Inheritance diagram for RayTracer::Torus:



Public Member Functions

- **Torus** (Math::Vector3D torus_pos, double r, double R)
- std::optional< double > hits (const Ray &ray) const override
- Math::Vector3D getNormal (const Math::Vector3D &position) const

- virtual std::optional< double > **hits** (const Ray &ray) const =0
- virtual Math::Vector3D **getNormal** (const Math::Vector3D &position) const =0

Public Attributes

- Math::Vector3D **torus_pos**
- double **r**
- double **R**
- double **distance**

Public Attributes inherited from RayTracer::IPrimitives

- double **distance**

4.27.1 Member Function Documentation

4.27.1.1 getNormal()

```
Math::Vector3D RayTracer::Torus::getNormal (
            const Math::Vector3D & position ) const  [inline], [virtual]
```

Implements RayTracer::IPrimitives.

4.27.1.2 hits()

```
std::optional< double > RayTracer::Torus::hits (
            const Ray & ray ) const  [inline], [override], [virtual]
```

Implements RayTracer::IPrimitives.

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

- include/Torus.hpp

4.28 RayTracer::Config::Torus_t Struct Reference

Public Attributes

- Math::Vector3D **torus_pos**
- double **r**
- double **R**

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

- include/Config.hpp

4.29 Math::Vector3D Class Reference

Public Member Functions

- **Vector3D** (double x, double y, double z)
- **Vector3D** (const Vector3D &other)=default
- **Vector3D** (Vector3D &&other)=default
- Vector3D & **operator=** (const Vector3D &other)=default
- Vector3D & **operator=** (Vector3D &&other)=default
- double **length** () const
- double **dot** (const Vector3D &other) const
- Vector3D **operator+** (const Vector3D &other) const
- Vector3D & **operator+=** (const Vector3D &other)
- Vector3D **operator+** (double t) const
- Vector3D **operator-** (const Vector3D &other) const
- Vector3D **operator-** () const
- Vector3D & **operator-=** (const Vector3D &other)
- Vector3D **operator**∗ (double scalar) const
- Vector3D & **operator**∗**=** (double scalar)
- Vector3D **operator**∗ (const Vector3D &other)
- Vector3D **operator**/ (double scalar) const
- Vector3D & **operator/=** (double scalar)
- Vector3D **normalize** () const
- double **lengthSquared** () const

Public Attributes

- double **x**
- double **y**
- double **z**

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

- include/Vector3D.hpp

Chapter 5

File Documentation

5.1 AmbientLight.hpp

```
00001 /*
00002 ** EPITECH PROJECT, 2023
00003 ** Tek2
00004 ** File description:
00005 ** AmbientLight
00006 */
00007
00008 #pragma once
00009
00010 #include "ILight.hpp"
00011
00012 namespace RayTracer {
00013
00014     class AmbiantLight : public ILight {
00015     public:
00016         AmbiantLight(const Color& color)
00017             : color_(color) {}
00018
00019         virtual Color getAmbientColor(const RayTracer::Material& material) const override { return
      color_ * material.ambient_ * material.color_; }
00020         virtual Color getDiffuseColor(const Math::Vector3D& point,const Math::Vector3D& normal , const
      RayTracer::Material& material) const override { return Color(0, 0, 0); }
00021         virtual Math::Vector3D getDirection(const Math::Vector3D& point) const override { return
      Math::Vector3D(0, 0, 0); }
00022         virtual double getDistance(const Math::Vector3D& point) const override { return 0; }
00023         virtual Color getSpecularColor(const Math::Vector3D& point, const Math::Vector3D&
      viewDirection, const Math::Vector3D& normal, const RayTracer::Material& material) const override {
      return Color(0, 0, 0); }
00024
00025         Color color_;
00026     private:
00027     };
00028
00029 } // namespace RayTracer
```

5.2 Camera.hpp

```
00001 /*
00002 ** EPITECH PROJECT, 2023
00003 ** B-OOP-400-LYN-4-1-raytracer-maxime.gregoire
00004 ** File description:
00005 ** Camera.hpp
00006 */
00007
00008 #pragma once
00009
00010 #include "Ray.hpp"
00011 #include "Vector3D.hpp"
00012 #include "Config.hpp"
00013 #include "Rectangle3D.hpp"
00014 #include <libconfig.h++>
00015
00016 namespace RayTracer {
00017     class Camera {
```

```
00018         public:
00019
00020             Math::Vector3D origin;
00021             Rectangle3D screen;
00022
00023             // Camera variables
00024             int screen_width, screen_height;
00025             Math::Vector3D camera_pos;
00026             Math::Vector3D camera_rotation;
00027             double camera_fov;
00028
00029             ~Camera() = default;
00030             Camera() : origin(Math::Vector3D(0, 0, 0)), screen(Rectangle3D()) {}
00031             Camera(const Math::Vector3D& origin_, const Rectangle3D& screen_)
00032                 : origin(origin_), screen(screen_) {}
00033
00034             // get ray method
00035             Ray getRay(double u, double v) const {
00036                 Math::Vector3D dest = screen.pointAt(u, v);
00037                 Math::Vector3D dir = (dest - origin);
00038                 return Ray(origin, dir);
00039             }
00040
00041             // setter and getter
00042             void getCamera(const std::string& file);
00043             RayTracer::Camera setCamera();
00044
00045     };
00046 }
```

5.3 Color.hpp

```
00001 /*
00002 ** EPITECH PROJECT, 2023
00003 ** Tek2
00004 ** File description:
00005 ** Color
00006 */
00007
00008 #pragma once
00009
00010 #include <iostream>
00011 #include <algorithm>
00012
00013 namespace RayTracer {
00014     class Color  {
00015         public:
00016             // Attributs
00017             double r, g, b;
00018             // Constructeurs
00019             Color() : r(0.0), g(0.0), b(0.0) {}
00020             Color(double red, double green, double blue) : r(red), g(green), b(blue) {}
00021
00022             // Accesseurs
00023             double red() const { return r; }
00024             double green() const { return g; }
00025             double blue() const { return b; }
00026
00027             // Opérateurs arithmétiques
00028             Color operator+ (const Color& c) const { return Color(r + c.r, g + c.g, b + c.b); }
00029             Color operator* (double k) const { return Color(k * r, k * g, k * b); }
00030             Color operator* (const Color& c) const { return Color(r * c.r, g * c.g, b * c.b); }
00031             Color operator/(double scalar) const {
00032             return Color(r / scalar, g / scalar, b / scalar);
00033             }
00034
00035             Color getColor() const {
00036                 return *this;
00037             }
00038
00039             Color& operator+=(const Color& other)
00040             {
00041                 r += other.r;
00042                 g += other.g;
00043                 b += other.b;
00044                 return *this;
00045             }
00046             // Autres méthodes
00047             void clamp(); // Tronquer les valeurs des composantes à [0, 1]
00048             void print() const; // Afficher les valeurs des composantes
00049             void toRGB();
00050
00051         private:
```

```
00052         };
00053
00054         // Méthode pour tronquer les valeurs des composantes à [0, 1]
00055     inline void Color::clamp() {
00056         r = std::min(std::max(r / 255.0, 0.0), 1.0);
00057         g = std::min(std::max(g / 255.0, 0.0), 1.0);
00058         b = std::min(std::max(b / 255.0, 0.0), 1.0);
00059     }
00060
00061     inline void Color::toRGB() {
00062         r = r * 255;
00063         g = g * 255;
00064         b = b * 255;
00065     }
00066     // Méthode pour afficher les valeurs des composantes
00067     inline void Color::print() const {
00068         std::cout « "(" « r « ", " « g « ", " « b « ")" « std::endl;
00069     }
00070
00071 }
```

5.4 Cone.hpp

```
00001 /*
00002 ** EPITECH PROJECT, 2023
00003 ** cone header file
00004 ** File description:
00005 ** Cone
00006 */
00007
00008 #pragma once
00009
00010 #include "Ray.hpp"
00011 #include "Vector3D.hpp"
00012 #include "IPrimitives.hpp"
00013 #include <optional>
00014
00015 namespace RayTracer {
00016     class Cone : public IPrimitives {
00017         public:
00018             Math::Vector3D cone_pos;
00019             double radius;
00020             double height;
00021
00022             Cone(Math::Vector3D cone_pos, double radius, double height)
00023                 : cone_pos(cone_pos), radius(radius), height(height) {}
00024
00025             std::optional<double> hits(const Ray& ray) const override {
00026                 std::optional<double> ray_unit;
00027                 double a = pow(ray.direction.x, 2) + pow(ray.direction.y, 2) - pow(ray.direction.z, 2)
      * pow(radius / height, 2);
00028                 double b = 2 * (ray.direction.x * (ray.origin.x - cone_pos.x) + ray.direction.y *
      (ray.origin.y - cone_pos.y) - ray.direction.z * (ray.origin.z - cone_pos.z) * pow(radius / height,
      2));
00029                 double c = pow(ray.origin.x - cone_pos.x, 2) + pow(ray.origin.y - cone_pos.y, 2) -
      pow(ray.origin.z - cone_pos.z, 2) * pow(radius / height, 2);
00030                 double delta = pow(b, 2) - 4 * a * c;
00031                 if (delta < 0)
00032                     return std::nullopt;
00033                 double t1 = (-b - sqrt(delta)) / (2 * a);
00034                 double t2 = (-b + sqrt(delta)) / (2 * a);
00035                 if (t1 > 0 && t2 > 0) {
00036                     double hit_point = std::min(t1, t2);
00037                     Math::Vector3D point = ray.origin + hit_point * ray.direction;
00038                     if (point.z <= cone_pos.z && point.z >= cone_pos.z - height)
00039                         ray_unit = hit_point;
00040                 } else if (t1 > 0) {
00041                     Math::Vector3D point = ray.origin + t1 * ray.direction;
00042                     if (point.z <= cone_pos.z && point.z >= cone_pos.z - height)
00043                         ray_unit = t1;
00044                 } else if (t2 > 0) {
00045                     Math::Vector3D point = ray.origin + t2 * ray.direction;
00046                     if (point.z <= cone_pos.z && point.z >= cone_pos.z - height)
00047                         ray_unit = t2;
00048                 }
00049                 return ray_unit;
00050             }
00051
00052             Math::Vector3D getNormal(const Math::Vector3D& hit_point) const {
00053                 double k = radius / height;
00054                 double dx = 2 * (hit_point.x - cone_pos.x);
00055                 double dy = 2 * (hit_point.y - cone_pos.y);
00056                 double dz = 2 * (hit_point.z - cone_pos.z) * (1 - k*k) - 2 * (hit_point.z -
      cone_pos.z);
```

```
00057                 return Math::Vector3D(dx, dy, dz).normalize();
00058             }
00059
00060
00061         protected:
00062         private:
00063     };
00064 }
```

5.5 Config.hpp

```
00001 /*
00002 ** EPITECH PROJECT, 2023
00003 ** B-OOP-400-LYN-4-1-raytracer-maxime.gregoire
00004 ** File description:
00005 ** Config
00006 */
00007
00008
00009 #include "Plane.hpp"
00010 #include "Sphere.hpp"
00011 #include "Cylinder.hpp"
00012 #include "Cone.hpp"
00013 #include "Torus.hpp"
00014 #include "Vector3D.hpp"
00015 #include "Color.hpp"
00016 #include "IPrimitives.hpp"
00017 #include "ILight.hpp"
00018 #include <libconfig.h++>
00019 #include <iostream>
00020 #include <string>
00021 #include <fstream>
00022 #include <string>
00023 #include <memory>
00024 #include <vector>
00025
00026 #pragma once
00027
00028 namespace RayTracer {
00029     class Config {
00030         public:
00031             Config() = default;
00032             ~Config() = default;
00033
00034             // Sphere variables
00035             struct Sphere_t {
00036                 Math::Vector3D sphere_pos;
00037                 double radius;
00038                 int sphere_r, sphere_g, sphere_b;
00039             };
00040             std::vector<Sphere_t> spheres;
00041
00042             // Plane variables
00043             struct Plane_t {
00044                 Math::Vector3D plane_pos;
00045                 Math::Vector3D plane_normal;
00046             };
00047             std::vector<Plane_t> planes;
00048
00049             // Cylinder variables
00050             struct Cylinder_t {
00051                 Math::Vector3D cylinder_pos;
00052                 double radius;
00053                 double height;
00054             };
00055             std::vector<Cylinder_t> cylinders;
00056
00057             // Cone variables
00058             struct Cone_t {
00059                 Math::Vector3D cone_pos;
00060                 double radius;
00061                 double height;
00062             };
00063             std::vector<Cone_t> cones;
00064
00065             // Torus variables
00066             struct Torus_t {
00067                 Math::Vector3D torus_pos;
00068                 double r;
00069                 double R;
00070             };
00071             std::vector<Torus_t> toruses;
00072
```

```
00073             //Material variables
00074             struct Material_t {
00075                 Color color_;
00076                 int sphere_r, sphere_g, sphere_b;
00077                 double ambient_;
00078                 double diffuse_;
00079                 double specular_;
00080                 double shininess_;
00081             };
00082             std::vector<Material_t> materials;
00083 
00084             //PointLight variables
00085             struct PointLight_t {
00086                 Math::Vector3D position_;
00087                 Color color_;
00088             };
00089             std::vector<PointLight_t> pointLights;
00090 
00091             //DirectionalLight variables
00092             struct DirectionalLight_t {
00093                 Math::Vector3D direction_;
00094                 Color color_;
00095                 double intensity;
00096             };
00097             std::vector<DirectionalLight_t> directionalLights;
00098 
00099             //DiffuseLight variables
00100             struct DiffuseLight_t {
00101                 Color color_;
00102                 Math::Vector3D direction_;
00103                 double intensity_;
00104             };
00105             std::vector<DiffuseLight_t> diffuseLights;
00106 
00107             //AmbientLight variables
00108             struct AmbientLight_t {
00109                 Color color_;
00110             };
00111             std::vector<AmbientLight_t> ambientLights;
00112 
00113             std::vector<std::shared_ptr<IPrimitives» primitives;
00114             std::vector<std::shared_ptr<ILight» lights;
00115             std::vector<std::shared_ptr<Material» material;
00116 
00117             void getPrimitives(const std::string& file);
00118             void getLights(const std::string& file);
00119             void getMaterials(const std::string& file);
00120             void setPrimitives();
00121             void setMaterials();
00122             void setLights();
00123 
00124         protected:
00125         private:
00126     };
00127 }
```

5.6 Cylinder.hpp

```
00001 /*
00002 ** EPITECH PROJECT, 2023
00003 ** Cylinder function set up
00004 ** File description:
00005 ** Cylinder
00006 */
00007 
00008 #pragma once
00009 
00010 #include "Ray.hpp"
00011 #include "Vector3D.hpp"
00012 #include "IPrimitives.hpp"
00013 #include <optional>
00014 
00015 namespace RayTracer {
00016     class Cylinder : public IPrimitives{
00017         public:
00018             Math::Vector3D cylinder_pos;
00019             double radius;
00020             double height;
00021             Cylinder(Math::Vector3D cylinder_pos, double radius, double height)
00022                 : cylinder_pos(cylinder_pos), radius(radius), height(height) {};
00023 
00024             std::optional<double> hits(const Ray& ray) const override {
00025                 // Compute the discriminant of the quadratic equation for ray-cylinder intersection.
```

```
00026                 Math::Vector3D oc = ray.origin - cylinder_pos;
00027                 double a = ray.direction.x * ray.direction.x + ray.direction.y * ray.direction.y;
00028                 double b = 2 * (ray.direction.x * oc.x + ray.direction.y * oc.y);
00029                 double c = oc.x * oc.x + oc.y * oc.y - radius * radius;
00030                 double discriminant = b * b - 4 * a * c;
00031                 if (discriminant < 0)
00032                     return std::nullopt;
00033                 double t1 = (-b - sqrt(discriminant)) / (2 * a);
00034                 double t2 = (-b + sqrt(discriminant)) / (2 * a);
00035                 if (t1 < 0 && t2 < 0)
00036                     return std::nullopt;
00037                 double t = (t1 < t2 && t1 >= 0) ? t1 : t2;
00038                 double z = ray.origin.z + t * ray.direction.z;
00039                 if (z < cylinder_pos.z - height/2 || z > cylinder_pos.z + height/2)
00040                     return std::nullopt;
00041                 return t;
00042             }
00043
00044             Math::Vector3D getNormal(const Math::Vector3D& point) const override {
00045                 // Project the point onto the plane passing through the center of the cylinder and
      perpendicular to the y axis
00046                 Math::Vector3D projection = Math::Vector3D(point.x, cylinder_pos.y, point.z);
00047
00048                 // Calculate the normal vector as the difference between the projection and the
      cylinder's center
00049                 Math::Vector3D normal = projection - cylinder_pos;
00050
00051                 // Normalize the normal vector and return it
00052                 return normal.normalize();
00053             }
00054
00055
00056         protected:
00057         private:
00058     };
00059 }
```

5.7 DiffuseLight.hpp

```
00001 /*
00002 ** EPITECH PROJECT, 2023
00003 ** Tek2
00004 ** File description:
00005 ** DiffuseLight
00006 */
00007
00008 #pragma once
00009
00010 #include "ILight.hpp"
00011
00012 namespace RayTracer {
00013
00014     class DiffuseLight : public ILight {
00015     public:
00016
00017         DiffuseLight(Color color, Math::Vector3D direction, double intensity)
00018             : color_(std::move(color)), direction_(std::move(direction)), intensity_(intensity) {}
00019
00020         Color getAmbientColor(const RayTracer::Material& material) const override {
00021             return Color(0, 0, 0);
00022         }
00023
00024         Color getDiffuseColor(const Math::Vector3D& point,const Math::Vector3D& normal , const
      RayTracer::Material& material) const override {
00025             return color_ * (intensity_ * std::max(0.0, direction_.dot(-getDirection(point)))) *
      material.diffuse_;
00026         }
00027
00028         Math::Vector3D getDirection(const Math::Vector3D& point) const override {
00029             return direction_;
00030         }
00031
00032         double getDistance(const Math::Vector3D& point) const override {
00033             return std::numeric_limits<double>::infinity();
00034         }
00035
00036         virtual Color getSpecularColor(const Math::Vector3D& point, const Math::Vector3D&
      viewDirection, const Math::Vector3D& normal, const RayTracer::Material& material) const override
00037         {
00038             return Color(0, 0, 0);
00039         }
00040
00041     private:
```

```
00042         Color color_;
00043         Math::Vector3D direction_;
00044         double intensity_;
00045     };
00046
00047 } // namespace RayTracer
```

5.8 DirectionalLight.hpp

```
00001 /*
00002 ** EPITECH PROJECT, 2023
00003 ** Tek2
00004 ** File description:
00005 ** DirectionalLight
00006 */
00007
00008 #pragma once
00009
00010 #include "ILight.hpp"
00011 #include "Color.hpp"
00012 #include "Vector3D.hpp"
00013
00014 namespace RayTracer {
00015
00016     class DirectionalLight : public ILight {
00017     public:
00018         DirectionalLight(const Math::Vector3D& direction, const Color& color, double intensity)
00019             : direction_(direction.normalize()), color_(color), intensity_(intensity) {}
00020
00021         Color getAmbientColor(const Material& material) const override {
00022             return Color(0, 0, 0);
00023         }
00024
00025         Color getDiffuseColor(const Math::Vector3D& point, const Math::Vector3D& normal, const
      Material& material) const override {
00026             double diffuseIntensity = std::max(0.0, normal.dot(-direction_));
00027             return color_ * (material.diffuse_ * diffuseIntensity * intensity_);
00028         }
00029
00030         Math::Vector3D getDirection(const Math::Vector3D& point) const override {
00031             return -direction_;
00032         }
00033
00034         double getDistance(const Math::Vector3D& point) const override {
00035             return std::numeric_limits<double>::infinity();
00036         }
00037
00038         Color getSpecularColor(const Math::Vector3D& point, const Math::Vector3D& viewDirection, const
      Math::Vector3D& normal, const Material& material) const override {
00039             Math::Vector3D reflectedDirection = Math::reflect(direction_, normal);
00040             double specularIntensity = std::max(0.0, reflectedDirection.dot(viewDirection));
00041             return color_ * (material.specular_ * pow(specularIntensity, material.shininess_) *
      intensity_);
00042         }
00043
00044     private:
00045         Math::Vector3D direction_;
00046         Color color_;
00047         double intensity_;
00048     };
00049
00050 } // namespace RayTracer
```

5.9 ILight.hpp

```
00001 /*
00002 ** EPITECH PROJECT, 2023
00003 ** Tek2
00004 ** File description:
00005 ** ILight
00006 */
00007
00008 #pragma once
00009 #pragma once
00010
00011 #pragma once
00012
00013 #pragma once
00014
```

```
00015 #include "Ray.hpp"
00016 #include "Color.hpp"
00017 #include "Vector3D.hpp"
00018 #include "Material.hpp"
00019 #include <iostream>
00020 #include <string>
00021 #include <cmath>
00022 #include <algorithm>
00023
00024 namespace RayTracer {
00025
00026     class ILight {
00027     public:
00028         virtual Color getAmbientColor(const RayTracer::Material& material) const = 0;
00029         virtual Color getDiffuseColor(const Math::Vector3D& point,const Math::Vector3D& normal , const
      RayTracer::Material& material) const = 0;
00030         virtual Math::Vector3D getDirection(const Math::Vector3D& point) const = 0;
00031         virtual double getDistance(const Math::Vector3D& point) const = 0;
00032         virtual Color getSpecularColor(const Math::Vector3D& point, const Math::Vector3D&
      viewDirection, const Math::Vector3D& normal, const RayTracer::Material& material) const = 0;
00033     };
00034
00035 } // namespace RayTracer
```

5.10 IPrimitives.hpp

```
00001 /*
00002 ** EPITECH PROJECT, 2023
00003 ** Epitech
00004 ** File description:
00005 ** IPrimitives.hpp
00006 */
00007
00008 #pragma once
00009
00010 #include <string>
00011 #include <libconfig.h++>
00012 #include <optional>
00013 #include "Ray.hpp"
00014
00015 namespace RayTracer {
00016     class IPrimitives  {
00017         public:
00018             virtual ~IPrimitives() = default;
00019             virtual std::optional<double> hits(const Ray& ray) const = 0;
00020             virtual Math::Vector3D getNormal(const Math::Vector3D& position) const = 0;
00021             double distance;
00022         protected:
00023         private:
00024     };
00025 }
```

5.11 Material.hpp

```
00001 /*
00002 ** EPITECH PROJECT, 2023
00003 ** Tek2
00004 ** File description:
00005 ** Material
00006 */
00007
00008 #pragma once
00009
00010 #include "Color.hpp"
00011
00012 namespace RayTracer {
00013
00014     class Material {
00015     public:
00016         Material(const Color& color, double ambient, double diffuse, double specular, double
      shininess)
00017             : color_{color}, ambient_{ambient}, diffuse_{diffuse}, specular_{specular},
      shininess_{shininess}
00018         {}
00019
00020         Color color() const { return color_; }
00021         double ambient() const { return ambient_; }
00022         double diffuse() const { return diffuse_; }
00023         double specular() const { return specular_; }
```

```
00024         double shininess() const { return shininess_; }
00025
00026         Color color_;
00027         double ambient_;
00028         double diffuse_;
00029         double specular_;
00030         double shininess_;
00031     private:
00032     };
00033
00034 } // namespace RayTracer
```

5.12 Matrix.hpp

```
00001 // /*
00002 // ** EPITECH PROJECT, 2023
00003 // ** Tek2
00004 // ** File description:
00005 // ** Matrix
00006 // */
00007
00008 // #pragma once
00009
00010 // #include "Vector3D.hpp"
00011 // #include <cstddef>
00012
00013 // namespace Math {
00014
00015 //     class Matrix {
00016 //         public:
00017 //             Matrix() {
00018 //                 for (int i = 0; i < 4; i++) {
00019 //                     for (int j = 0; j < 4; j++) {
00020 //                         m_data[i][j] = 0;
00021 //                     }
00022 //                 }
00023 //             }
00024
00025 //             Matrix(double arr[4][4]) {
00026 //                 for (int i = 0; i < 4; i++) {
00027 //                     for (int j = 0; j < 4; j++) {
00028 //                         m_data[i][j] = arr[i][j];
00029 //                     }
00030 //                 }
00031 //             }
00032
00033 //             Matrix operator*(const Matrix& other) const {
00034 //                 Matrix result;
00035
00036 //                 for (int i = 0; i < 4; i++) {
00037 //                     for (int j = 0; j < 4; j++) {
00038 //                         for (int k = 0; k < 4; k++) {
00039 //                             result.m_data[i][j] += m_data[i][k] * other.m_data[k][j];
00040 //                         }
00041 //                     }
00042 //                 }
00043
00044 //                 return result;
00045 //             }
00046
00047 //             Vector3D operator*(const Vector3D& p) const {
00048 //                 return Vector3D(
00049 //                     m_data[0][0]*p.x + m_data[0][1]*p.y + m_data[0][2]*p.z + m_data[0][3],
00050 //                     m_data[1][0]*p.x + m_data[1][1]*p.y + m_data[1][2]*p.z + m_data[1][3],
00051 //                     m_data[2][0]*p.x + m_data[2][1]*p.y + m_data[2][2]*p.z + m_data[2][3]
00052 //                 );
00053 //             }
00054
00055 //             Vector3D operator*(const Vector3D& v) const {
00056 //                 return Vector3D(
00057 //                     m_data[0][0]*v.x + m_data[0][1]*v.y + m_data[0][2]*v.z,
00058 //                     m_data[1][0]*v.x + m_data[1][1]*v.y + m_data[1][2]*v.z,
00059 //                     m_data[2][0]*v.x + m_data[2][1]*v.y + m_data[2][2]*v.z
00060 //                 );
00061 //             }
00062
00063 //             static Matrix translate(double x, double y, double z) {
00064 //                 double arr[4][4] = {
00065 //                     {1, 0, 0, x},
00066 //                     {0, 1, 0, y},
00067 //                     {0, 0, 1, z},
00068 //                     {0, 0, 0, 1}
00069 //                 };
```

```
00070
00071 //                 return Matrix(arr);
00072 //             }
00073
00074 //             static Matrix scale(double x, double y, double z) {
00075 //                 double arr[4][4] = {
00076 //                     {x, 0, 0, 0},
00077 //                     {0, y, 0, 0},
00078 //                     {0, 0, z, 0},
00079 //                     {0, 0, 0, 1}
00080 //                 };
00081
00082 //                 return Matrix(arr);
00083 //             }
00084
00085 //             static Matrix rotateX(double angle) {
00086 //                 double s = std::sin(angle);
00087 //                 double c = std::cos(angle);
00088
00089 //                 double arr[4][4] = {
00090 //                     {1, 0, 0, 0},
00091 //                     {0, c, -s, 0},
00092 //                     {0, s, c, 0},
00093 //                     {0, 0, 0, 1}
00094 //                 };
00095
00096 //                 return Matrix(arr);
00097 //             }
00098
00099 //             static Matrix rotateY(double angle) {
00100 //                 double s = std::sin(angle);
00101 //                 double c = std::cos(angle);
00102
00103 //                 double arr[4][4] = {
00104 //                     {c, 0, s, 0},
00105 //                     {0, 1, 0, 0},
00106 //                     {-s, 0, c, 0},
00107 //                     {0, 0, 0, 1}
00108 //                 };
00109
00110 //                 return Matrix(arr);
00111 //             }
00112
00113 //             static Matrix rotateZ(double angle) {
00114 //                 double s = std::sin(angle);
00115 //                 double c = std::cos(angle);
00116
00117 //                 double arr[4][4] = {
00118 //                     {c, -s, 0, 0},
00119 //                     {s, c, 0, 0},
00120 //                     {0, 0, 1, 0},
00121 //                     {0, 0, 0, 1}
00122 //                 };
00123
00124 //                 return Matrix(arr);
00125 //             }
00126
00127 //         private:
00128 //             double m_data[4][4];
00129 //         };
00130
00131 // }
```

5.13 Plane.hpp

```
00001 /*
00002 ** EPITECH PROJECT, 2023
00003 ** Tek2
00004 ** File description:
00005 ** Plane
00006 */
00007
00008 #pragma once
00009
00010 #include "Ray.hpp"
00011 #include "Vector3D.hpp"
00012 #include "IPrimitives.hpp"
00013 #include <optional>
00014
00015 namespace RayTracer {
00016     class Plane : public IPrimitives {
00017         public:
00018             Math::Vector3D normal;
```

```
00019             Math::Vector3D position;
00020             double distance;
00021
00022             Plane(Math::Vector3D normal, Math::Vector3D position)
00023                 : normal(normal), position(position) {}
00024
00025             std::optional<double> hits(const Ray& ray) const override {
00026                 std::optional<double> ray_unit;
00027                 double denom = normal.dot(ray.direction);
00028                 if (std::abs(denom) > 1e-6) {
00029                     Math::Vector3D v = position - ray.origin;
00030                     double t = v.dot(normal) / denom;
00031                     if (t <= 0) {
00032                         return -1.0;
00033                     }
00034                     ray_unit = t;
00035                 }
00036                 return ray_unit;
00037             }
00038
00039             Math::Vector3D getNormal(const Math::Vector3D& hit_point) const override {
00040                 return normal;
00041             }
00042         };
00043 } // namespace RayTracer
```

5.14 PointLight.hpp

```
00001 /*
00002 ** EPITECH PROJECT, 2023
00003 ** Tek2
00004 ** File description:
00005 ** PointLight
00006 */
00007 #pragma once
00008
00009 #include "ILight.hpp"
00010
00011 namespace RayTracer {
00012
00013     class PointLight : public ILight {
00014     public:
00015         PointLight(const Math::Vector3D& position, const Color& color)
00016             : position_(position), color_(color) {}
00017
00018         virtual Color getAmbientColor(const RayTracer::Material& material) const override { return
      Color(0, 0, 0); }
00019
00020         virtual Color getDiffuseColor(const Math::Vector3D& point,const Math::Vector3D& normal , const
      RayTracer::Material& material) const override {
00021             Math::Vector3D lightDirection = getDirection(point);
00022             double diffuseIntensity = lightDirection.dot(normal);
00023             if (diffuseIntensity > 0) {
00024                 diffuseIntensity *= material.diffuse_;
00025                 return color_ * diffuseIntensity;
00026             }
00027             return Color(0, 0, 0);
00028         }
00029
00030         virtual Math::Vector3D getDirection(const Math::Vector3D& point) const override { return
      (position_ - point).normalize(); }
00031
00032         virtual double getDistance(const Math::Vector3D& point) const override { return (position_ -
      point).length(); }
00033
00034         virtual Color getSpecularColor(const Math::Vector3D& point, const Math::Vector3D&
      viewDirection, const Math::Vector3D& normal, const RayTracer::Material& material) const override {
00035             Math::Vector3D lightDirection = getDirection(point);
00036             Math::Vector3D reflectionDirection = Math::reflect(lightDirection, normal);
00037             double specularIntensity = reflectionDirection.dot(viewDirection);
00038             if (specularIntensity > 0) {
00039                 specularIntensity = pow(specularIntensity, material.shininess_);
00040                 return color_ * material.specular_ * specularIntensity;
00041             }
00042             return Color(0, 0, 0);
00043         }
00044
00045         Math::Vector3D position_;
00046         Color color_;
00047
00048         double attenuationConstant_ = 1.0;
00049         double attenuationLinear_ = 0.0045;
00050         double attenuationQuadratic_ = 0.0075;
```

```
00051     };
00052
00053 } // namespace RayTracer
```

5.15 Ray.hpp

```
00001 /*
00002 ** EPITECH PROJECT, 2023
00003 ** Tek2
00004 ** File description:
00005 ** Ray
00006 */
00007
00008 #ifndef RAY_HPP_
00009 #define RAY_HPP_
00010
00011 #include "Vector3D.hpp"
00012 #include "Matrix.hpp"
00013 #include <cmath>
00014
00015 namespace RayTracer {
00016
00017     class Ray {
00018         public:
00019             Math::Vector3D origin;
00020             Math::Vector3D direction;
00021
00022             Ray() = default;
00023             Ray(const Math::Vector3D& origin, const Math::Vector3D& direction) : origin(origin),
      direction(direction) {}
00024
00025             // Copy constructor
00026             Ray(const Ray& other) = default;
00027
00028             // Move constructor
00029             Ray(Ray&& other) noexcept = default;
00030
00031             // Copy assignment operator
00032             Ray& operator=(const Ray& other) = default;
00033
00034             // Move assignment operator
00035             Ray& operator=(Ray&& other) noexcept = default;
00036
00037
00038             Math::Vector3D at(double t) const {
00039                 return Math::Vector3D((origin + direction) * t);
00040             }
00041
00042
00043             // void transform(const Math::Matrix& m) {
00044             //     direction = m * direction;
00045             //     origin = m * origin;
00046             // }
00047     };
00048 }
00049
00050 #endif /* !RAY_HPP_ */
```

5.16 Rectangle3D.hpp

```
00001 /*
00002 ** EPITECH PROJECT, 2023
00003 ** B-OOP-400-LYN-4-1-raytracer-maxime.gregoire
00004 ** File description:
00005 ** Rectangle3D
00006 */
00007
00008 #pragma once
00009
00010 #include <cmath>
00011 #include "Vector3D.hpp"
00012
00013 namespace RayTracer {
00014     class Rectangle3D {
00015         public:
00016             Math::Vector3D origin;
00017             Math::Vector3D bottom_side;
00018             Math::Vector3D left_side;
00019
```

```
00020             Rectangle3D() = default;
00021             Rectangle3D(const Math::Vector3D& origin_, const Math::Vector3D& bottom_side_, const
      Math::Vector3D& left_side_)
00022                 : origin(origin_), bottom_side(bottom_side_), left_side(left_side_) {}
00023
00024             Math::Vector3D pointAt(double u, double v) const {
00025                 return origin + bottom_side * u + left_side * v;
00026             }
00027     };
00028 }
```

5.17 Scene.hpp

```
00001 /*
00002 ** EPITECH PROJECT, 2023
00003 ** scene drawing functions
00004 ** File description:
00005 ** Scene
00006 */
00007
00008 #include <libconfig.h++>
00009 #include "Config.hpp"
00010 #include <iostream>
00011 #include <vector>
00012 #include <optional>
00013
00014 #pragma once
00015
00016 namespace RayTracer {
00017     class Scene {
00018         public:
00019         protected:
00020         private:
00021     };
00022 }
```

5.18 Sphere.hpp

```
00001 /*
00002 ** EPITECH PROJECT, 2023
00003 ** Tek2
00004 ** File description:
00005 ** Sphere
00006 */
00007
00008 #include "Vector3D.hpp"
00009 #include "Matrix.hpp"
00010 #include "IPrimitives.hpp"
00011 #include "Ray.hpp"
00012 #include <optional>
00013
00014 #pragma once
00015
00016 namespace RayTracer {
00017
00018     class Sphere : public IPrimitives {
00019         private:
00020             Math::Vector3D m_center;
00021             double m_radius;
00022         public:
00023             double distance;
00024
00025             Sphere(const Math::Vector3D &center, double radius)
00026                 : m_center(center), m_radius(radius) {}
00027
00028             Math::Vector3D center() const
00029             {
00030                 return m_center;
00031             }
00032
00033             double radius() const
00034             {
00035                 return m_radius;
00036             }
00037
00038             std::optional<double> hits(const Ray& ray) const override
00039             {
00040                 Math::Vector3D oc = ray.origin - this->m_center;
00041                 double a = ray.direction.dot(ray.direction);
```

```
00042                 double b = 2.0 * oc.dot(ray.direction);
00043                 double c = oc.dot(oc) - this->m_radius * this->m_radius;
00044                 double discriminant = b * b - 4.0 * a * c;
00045                 std::optional<double> ray_unit1;
00046                 std::optional<double> ray_unit2;
00047                 if (discriminant < 0.0) {
00048                     return -1.0;
00049                 } else {
00050                     ray_unit1 = (-b - std::sqrt(discriminant)) / (2.0 * a);
00051                     ray_unit2 = (-b + std::sqrt(discriminant)) / (2.0 * a);
00052                     if (ray_unit1 <= ray_unit2) {
00053                         return ray_unit1;
00054                     } else {
00055                         return ray_unit2;
00056                     }
00057                 }
00058             }
00059
00060             Math::Vector3D getNormal(const Math::Vector3D& position) const
00061             {
00062                 return (position - m_center).normalize();
00063             }
00064     };
00065
00066 } // namespace RayTracer
```

5.19 Torus.hpp

```
00001 /*
00002 ** EPITECH PROJECT, 2023
00003 ** toruses hit function file
00004 ** File description:
00005 ** Torus
00006 */
00007
00008 #pragma once
00009
00010 #include "Ray.hpp"
00011 #include "Vector3D.hpp"
00012 #include "IPrimitives.hpp"
00013 #include <optional>
00014 #include <vector>
00015
00016 namespace RayTracer {
00017     class Torus : public IPrimitives{
00018         public:
00019             Math::Vector3D torus_pos;
00020             double r;
00021             double R;
00022             double distance;
00023             Torus(Math::Vector3D torus_pos, double r, double R)
00024                 : torus_pos(torus_pos), r(r), R(R) {};
00025
00026             std::optional<double> hits(const Ray& ray) const override {
00027                 Math::Vector3D origin = ray.origin - torus_pos;
00028                 double a = pow(ray.direction.z, 2) + pow(ray.direction.y, 2) + pow(ray.direction.x,
      2);
00029                 double b = 2 * (ray.direction.z * origin.z + ray.direction.y * origin.y +
      ray.direction.x * origin.x);
00030                 double c = pow(origin.z, 2) + pow(origin.y, 2) + pow(origin.x, 2) - pow(r, 2) - pow(R,
      2);
00031                 double delta = pow(b, 2) - 4 * a * c;
00032                 std::optional<double> ray_unit;
00033                 if (delta < 0)
00034                     return std::nullopt;
00035                 double t1 = (-b - sqrt(delta)) / (2 * a);
00036                 double t2 = (-b + sqrt(delta)) / (2 * a);
00037                 if (t1 > 0 && t2 > 0) {
00038                     double hit_point = std::min(t1, t2);
00039                     Math::Vector3D point = ray.origin + hit_point * ray.direction;
00040                     if (point.x <= torus_pos.x && point.x >= torus_pos.x - R)
00041                         ray_unit = hit_point;
00042                 } else if (t1 > 0) {
00043                     Math::Vector3D point = ray.origin + t1 * ray.direction;
00044                     if (point.x <= torus_pos.x && point.x >= torus_pos.x - R)
00045                         ray_unit = t1;
00046                 } else if (t2 > 0) {
00047                     Math::Vector3D point = ray.origin + t2 * ray.direction;
00048                     if (point.x <= torus_pos.x && point.x >= torus_pos.x - R)
00049                         ray_unit = t2;
00050                 }
00051                 return ray_unit;
00052             }
```

```
00053
00054             Math::Vector3D getNormal(const Math::Vector3D& position) const
00055             {
00056                 double x = position.x - torus_pos.x;
00057                 double y = position.y - torus_pos.y;
00058                 double z = position.z - torus_pos.z;
00059                 double fx = 4 * x * (x * x + y * y + z * z + R * R - r * r) - 8 * R * x * x;
00060                 double fy = 4 * y * (x * x + y * y + z * z + R * R - r * r) - 8 * R * y * y;
00061                 double fz = 4 * z * (x * x + y * y + z * z + R * R - r * r);
00062                 Math::Vector3D normal(fx, fy, fz);
00063                 return normal.normalize();
00064             }
00065
00066
00067         protected:
00068         private:
00069     };
00070 }
```

5.20 Vector3D.hpp

```
00001 /*
00002 ** EPITECH PROJECT, 2023
00003 ** Tek2
00004 ** File description:
00005 ** Point3D
00006 */
00007
00008 #pragma once
00009
00010 #include <cmath>
00011
00012 namespace Math {
00013     class Vector3D {
00014         public:
00015             double x, y, z;
00016
00017             Vector3D() : x(0), y(0), z(0) {}
00018             Vector3D(double x, double y, double z) : x(x), y(y), z(z) {}
00019             Vector3D(const Vector3D& other) = default;
00020             Vector3D(Vector3D&& other) = default;
00021             Vector3D& operator=(const Vector3D& other) = default;
00022             Vector3D& operator=(Vector3D&& other) = default;
00023             ~Vector3D() = default;
00024
00025             double length() const {
00026                 return std::sqrt(x * x + y * y + z * z);
00027             }
00028
00029             double dot(const Vector3D& other) const {
00030                 return x * other.x + y * other.y + z * other.z;
00031             }
00032
00033             Vector3D operator+(const Vector3D& other) const {
00034                 return Vector3D(x + other.x, y + other.y, z + other.z);
00035             }
00036
00037             Vector3D& operator+=(const Vector3D& other) {
00038                 x += other.x;
00039                 y += other.y;
00040                 z += other.z;
00041                 return *this;
00042             }
00043
00044             Vector3D operator+(double t) const {
00045                 return Vector3D(x + t, y + t, z + t);
00046             }
00047
00048             Vector3D operator-(const Vector3D& other) const {
00049                 return Vector3D(x - other.x, y - other.y, z - other.z);
00050             }
00051
00052             Vector3D operator-() const {
00053                 return Vector3D(-x, -y, -z);
00054             }
00055
00056             Vector3D& operator-=(const Vector3D& other) {
00057                 x -= other.x;
00058                 y -= other.y;
00059                 z -= other.z;
00060                 return *this;
00061             }
00062
```

```
00063             Vector3D operator*(double scalar) const {
00064                 return Vector3D(x * scalar, y * scalar, z * scalar);
00065             }
00066
00067             Vector3D& operator*=(double scalar) {
00068                 x *= scalar;
00069                 y *= scalar;
00070                 z *= scalar;
00071                 return *this;
00072             }
00073
00074             Vector3D operator*(const Vector3D& other) {
00075                 x *= other.x;
00076                 y *= other.y;
00077                 z *= other.z;
00078                 return *this;
00079             }
00080
00081             Vector3D operator/(double scalar) const {
00082                 return Vector3D(x / scalar, y / scalar, z / scalar);
00083             }
00084
00085             Vector3D& operator/=(double scalar) {
00086                 x /= scalar;
00087                 y /= scalar;
00088                 z /= scalar;
00089                 return *this;
00090             }
00091
00092             Vector3D normalize() const {
00093                 double len = length();
00094                 if (len != 0) {
00095                     return *this / len;
00096                 }
00097                 return Vector3D();
00098             }
00099
00100             double lengthSquared() const {
00101                 return x * x + y * y + z * z;
00102             }
00103
00104
00105
00106     };
00107
00108         inline Vector3D operator-=(const Vector3D& p1, const Vector3D& p2) {
00109             return Vector3D(p1.x - p2.x, p1.y - p2.y, p1.z - p2.z);
00110         }
00111
00112         inline Vector3D operator*(const double scalar, const Vector3D& vec)
00113         {
00114             return Vector3D(scalar * vec.x, scalar * vec.y, scalar * vec.z);
00115         }
00116
00117         inline Vector3D reflect(const Math::Vector3D& incident, const Math::Vector3D& normal)
00118         {
00119             return incident - 2 * incident.dot(incident) * normal;
00120         }
00121 }
```

Index