Ray Tracer

1.0

Generated by Doxygen 1.9.1

Chapter 1

Raytracer Project

The raytracer project is an implementation of a rendering engine that utilizes the raytracing technique to generate realistic images by simulating the behavior of light. Raytracing is a powerful method used in computer graphics to produce highly realistic renders by tracing the path of light rays through a virtual scene.

This project, implemented in C++, encompasses a raytracing rendering engine that allows for the creation of images by calculating the interactions between light and objects in the scene. It employs optical principles such as reflection, refraction, and light diffusion to simulate the effects of light on object surfaces.

Key features of the project include 3D geometry modeling, application of textures and materials to objects, implementation of light sources, shadow handling, simulation of reflection and refraction phenomena, as well as management of depth of field and motion blur effects. It may also encompass advanced techniques such as global illumination computation and simulation of complex optical phenomena like caustics.

The raytracer project requires a solid understanding of mathematics, including geometry, linear algebra, and vector calculus. It is implemented in C++ to provide a high-performance and flexible environment for developing the raytracing engine.

Ultimately, the raytracing rendering engine produces realistic and compelling images by employing intricate calculations to simulate the behavior of light. It finds applications in numerous fields, such as cinematic animation, visual effects, video games, architectural design, and virtual reality, to create immersive and visually stunning virtual environments.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

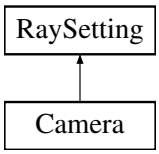
Class Documentation

4.1 Camera Class Reference

[Camera] The camera is the point of view of the scene It contains the position, the rotation, the field of view and the resolution It also contains the pixels of the camera It inherits from RaySetting

`#include <Camera.hpp>`

Inheritance diagram for Camera:



Public Member Functions

- Camera ()

  Default construct a new Camera object.
- Camera (Vector3D position, Vector3D rotation, double fov, sf::Vector2i resolution)

  Construct a new Camera object.
- Camera (const Camera &camera)=default

  Copy constructor of Camera.
- Camera (Camera &camera)=default

  Move constructor of Camera.
- ~Camera ()=default

  Destroy the Camera object.
- Camera & operator= (Camera &camera)=default

  Operator = of Camera (copy the Camera)
- Camera & operator= (const Camera &camera)=default

  Move operator of Camera (move the Camera)
- Vector3D getPosition () const

  Get the position of the camera.
- Vector3D getRotation () const

  Get the rotation of the camera.

- double getFov () const

  Get the Fov of the camera (in degrees)
- sf::Vector2i getResolution () const

  Get the Resolution of the camera.
- std::vector< Color > getPixels () const

  *Get the Pixels of the camera (width * height) (r, g, b)*
- Color getPixel (int x, int y) const

  Get the Pixel of the camera.
- Color getPixel (int index) const

  Get the Pixel of the camera.
- void setPosition (Vector3D position)

  Set the Position of the camera.
- void setRotation (Vector3D rotation)

  Set the Rotation of the camera.
- void setFov (double fov)

  Set the Fov of the camera.
- void setResolution (sf::Vector2i resolution)

  Set the Resolution of the camera.
- void setPixels (std::vector< Color > pixels)

  Set the Pixels of the camera.
- void setPixel (int x, int y, Color color)

  Set the Pixel of the camera at the index.
- void setPixel (int index, Color color)

  Set the Pixel of the camera at the index.
- Ray getRay (float x, float y) const

  Get the Ray from the camera to the pixel (x, y) on the screen.
- std::string getType () override

  Get the Type of the object.

4.1.1 Detailed Description

[Camera] The camera is the point of view of the scene It contains the position, the rotation, the field of view and the resolution It also contains the pixels of the camera It inherits from RaySetting

Parameters

_position	position of the camera (x, y, z)
_rotation	rotation of the camera (in degrees: x, y, z)
_fov	field of view of the camera (in degrees)
_resolution	resolution of the camera (width, height)

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Camera() [1/3]

```
Camera::Camera (
            Vector3D position,
```

```
            Vector3D rotation,
            double fov,
            sf::Vector2i resolution )
```

Construct a new Camera object.

Parameters

position	position of the camera (x, y, z)
rotation	rotation of the camera (in degrees: x, y, z)
fov	field of view of the camera (in degrees)
resolution	resolution of the camera (width, height)

4.1.2.2 Camera() [2/3]

```
Camera::Camera (
            const Camera & camera )  [default]
```

Copy constructor of Camera.

Parameters

Camera	

4.1.2.3 Camera() [3/3]

```
Camera::Camera (
            Camera & camera )  [default]
```

Move constructor of Camera.

Parameters

Camera	

4.1.3 Member Function Documentation

4.1.3.1 getFov()

```
double Camera::getFov ( ) const
```

Get the Fov of the camera (in degrees)

Returns

double

4.1.3.2 getPixel() [1/2]

```
Color Camera::getPixel (
            int index ) const
```

Get the Pixel of the camera.

Parameters

index	of the pixel on the screen (width ∗ y + x)

Returns

Color

4.1.3.3 getPixel() [2/2]

```
Color Camera::getPixel (
            int x,
            int y ) const
```

Get the Pixel of the camera.

Parameters

x	coordinate of the pixel on the screen (width)
y	coordinate of the pixel on the screen (height)

Returns

Color

4.1.3.4 getPixels()

```
std::vector<Color> Camera::getPixels ( ) const
```

Get the Pixels of the camera (width ∗ height) (r, g, b)

Returns

std::vector<Color>

4.1.3.5 getPosition()

```
Vector3D Camera::getPosition ( ) const
```

Get the position of the camera.

Returns

Vector3D

4.1.3.6 getRay()

```
Ray Camera::getRay (
            float x,
            float y ) const
```

Get the Ray from the camera to the pixel (x, y) on the screen.

Parameters

x	coordinate of the pixel on the screen (width)
y	coordinate of the pixel on the screen (height)

Returns

Ray

4.1.3.7 getResolution()

```
sf::Vector2i Camera::getResolution ( ) const
```

Get the Resolution of the camera.

Returns

sf::Vector2i (width = x, height = y)

4.1.3.8 getRotation()

```
Vector3D Camera::getRotation ( ) const
```

Get the rotation of the camera.

Returns

Vector3D (in degrees)

4.1.3.9 getType()

```
std::string Camera::getType ( )  [inline], [override], [virtual]
```

Get the Type of the object.

Returns

"Camera"

Implements RaySetting.

4.1.3.10 operator=() [1/2]

```
Camera& Camera::operator= (
            Camera & camera )  [default]
```

Operator = of Camera (copy the Camera)

Parameters

Camera	The Camera to copy

Returns

Camera&

4.1.3.11 operator=() [2/2]

```
Camera& Camera::operator= (
            const Camera & camera )  [default]
```

Move operator of Camera (move the Camera)

Parameters

Camera	The Camera to move

Returns

Camera&

4.1.3.12 setFov()

```
void Camera::setFov (
            double fov )
```

Set the Fov of the camera.

Parameters

fov	(in degrees)

4.1.3.13 setPixel() [1/2]

```
void Camera::setPixel (
            int index,
            Color color )
```

Set the Pixel of the camera at the index.

Parameters

index	of the pixel in the vector
color	of the pixel

4.1.3.14 setPixel() [2/2]

```
void Camera::setPixel (
            int x,
            int y,
            Color color )
```

Set the Pixel of the camera at the index.

Parameters

x	(wigth)
y	(height)
color	of the pixel

4.1.3.15 setPixels()

```
void Camera::setPixels (
            std::vector< Color > pixels )
```

Set the Pixels of the camera.

Parameters

pixels	

4.1.3.16 setPosition()

```
void Camera::setPosition (
            Vector3D position )
```

Set the Position of the camera.

Parameters

position	

4.1.3.17 setResolution()

```
void Camera::setResolution (
            sf::Vector2i resolution )
```

Set the Resolution of the camera.

Parameters

resolution	(width, height)

4.1.3.18 setRotation()

```
void Camera::setRotation (
            Vector3D rotation )
```

Set the Rotation of the camera.

Parameters

rotation	

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

- inc/Camera.hpp

4.2 Color Class Reference

[Color] class This class is used to represent a color It is composed of 4 unsigned char (r, g, b, a) and a are between 0 and 255 it's possible to use the color class with sfml or openGL

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

Public Member Functions

- Color ()=default

  Defaultc construct a new Color object.
- Color (unsigned char r, unsigned char g, unsigned char b, unsigned char a=255)

  Construct a new Color object.
- Color (const Color &other)=default

  Copy construct a new Color object.
- Color (Color &&other)=default

  move construct a new Color object
- ~Color ()=default

  Destroy the Color object.
- Color & operator= (const Color &other)=default

  [operator=] copy assignment operator
- Color & operator= (Color &&other)=default

  [operator=] move assignment operator
- Color & operator+ (const Color &other)

  [operator+] add two colors
- Color & operator- (const Color &other)

  [operator-] substract two colors
- Color & operator* (const Color &other)

  [operator] multiply two colors*
- Color & operator/ (const Color &other)

  [operator/] divide two colors
- Color & operator+ (const double scalar)

  [operator+] add a scalar to a color
- Color & operator- (const double scalar)

  [operator-] substract a scalar to a color
- Color & operator* (const double scalar)

  [operator] multiply a scalar to a color*
- Color & operator/ (const double scalar)

  [operator/] divide a scalar to a color
- Color & operator+= (const Color &other)

  [operator+=] add two colors
- Color & operator-= (const Color &other)

  [operator-=] substract two colors
- Color & operator*= (const Color &other)

  [operator=] multiply two colors*
- Color & operator/= (const Color &other)

  [operator/=] divide two colors
- Color & operator+= (const double scalar)

  [operator+=] add a scalar to a color
- Color & operator-= (const double scalar)

[operator-=] substract a scalar to a color

- Color & operator∗= (const double scalar)

  [operator∗=] multiply a scalar to a color
- Color & operator/= (const double scalar)

  [operator/=] divide a scalar to a color
- bool operator== (const Color &other) const

  compare two colors
- bool operator!= (const Color &other) const

  compare two colors
- std::ostream & operator<< (std::ostream &os)

  print a color
- unsigned char getR () const

  Getter for the red value.
- unsigned char getG () const

  Getter for the green value.
- unsigned char getB () const

  Getter for the blue value.
- unsigned char getA () const

  Getter for the alpha value.
- void setR (unsigned char r)

  Setter for the red value.
- void setG (unsigned char g)

  Setter for the green value.
- void setB (unsigned char b)

  Setter for the blue value.
- void setA (unsigned char a)

  Setter for the alpha value.
- void gammaCorrection (double gamma)

  Apply gamma correction to a color.
- Color blend (const Color &other)

  blend two colors

Static Public Member Functions

- static Color mix (const Color &color1, const Color &color2, double factor)

  Mix two colors.
- static Color lerp (const Color &color1, const Color &color2, double factor)

  Linear interpolation between two colors.

4.2.1 Detailed Description

[Color] class This class is used to represent a color It is composed of 4 unsigned char (r, g, b, a) and a are between 0 and 255 it's possible to use the color class with sfml or openGL

Attention

: the color is stored in RGBA format

Parameters

r	red
g	green
b	blue
a	alpha

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Color() [1/3]

```
Color::Color (
            unsigned char r,
            unsigned char g,
            unsigned char b,
            unsigned char a = 255 )  [inline]
```

Construct a new Color object.

Parameters

r	
g	
b	
a	

4.2.2.2 Color() [2/3]

```
Color::Color (
            const Color & other )  [default]
```

Copy construct a new Color object.

Parameters

other	

4.2.2.3 Color() [3/3]

```
Color::Color (
            Color && other )  [default]
```

move construct a new Color object

Parameters

other	

4.2.3 Member Function Documentation

4.2.3.1 blend()

```
Color Color::blend (
            const Color & other )
```

blend two colors

Parameters

other	

Returns

the blended Color

4.2.3.2 gammaCorrection()

```
void Color::gammaCorrection (
            double gamma )
```

Apply gamma correction to a color.

Parameters

gamma	(gamma $>$ 0)

4.2.3.3 getA()

```
unsigned char Color::getA ( ) const
```

Getter for the alpha value.

Returns

unsigned char

4.2.3.4 getB()

```
unsigned char Color::getB ( ) const
```

Getter for the blue value.

Returns

unsigned char

4.2.3.5 getG()

```
unsigned char Color::getG ( ) const
```

Getter for the green value.

Returns

unsigned char

4.2.3.6 getR()

```
unsigned char Color::getR ( ) const
```

Getter for the red value.

Returns

unsigned char

4.2.3.7 lerp()

```
static Color Color::lerp (
            const Color & color1,
            const Color & color2,
            double factor )  [static]
```

Linear interpolation between two colors.

Parameters

color1	
color2	
factor	

Returns

Color the interpolated color

4.2.3.8 mix()

```
static Color Color::mix (
            const Color & color1,
            const Color & color2,
            double factor )  [static]
```

Mix two colors.

Parameters

color1	
color2	
factor	

Returns

Color

4.2.3.9 operator"!=()

```
bool Color::operator!= (
            const Color & other ) const
```

compare two colors

Parameters

other	

Returns

bool: true if the colors are different, false otherwise

4.2.3.10 operator∗() [1/2]

```
Color& Color::operator* (
            const Color & other )
```

[operator∗] multiply two colors

Parameters

other	

Returns

Color

4.2.3.11 operator∗() [2/2]

```
Color& Color::operator* (
            const double scalar )
```

[operator∗] multiply a scalar to a color

Parameters

scalar	

Returns

Color

4.2.3.12 operator∗=() [1/2]

```
Color& Color::operator*= (
            const Color & other )
```

[operator∗=] multiply two colors

Parameters

other	

Returns

Color

4.2.3.13 operator∗=() [2/2]

```
Color& Color::operator*= (
            const double scalar )
```

[operator∗=] multiply a scalar to a color

Parameters

scalar	

Returns

Color

4.2.3.14 operator+() [1/2]

```
Color& Color::operator+ (
            const Color & other )
```

[operator+] add two colors

Parameters

other	

Returns

Color

4.2.3.15 operator+() [2/2]

```
Color& Color::operator+ (
            const double scalar )
```

[operator+] add a scalar to a color

Parameters

scalar	

Returns

Color

4.2.3.16 operator+=() [1/2]

```
Color& Color::operator+= (
            const Color & other )
```

[operator+=] add two colors

Parameters

other	

Returns

Color

4.2.3.17 operator+=() [2/2]

```
Color& Color::operator+= (
            const double scalar )
```

[operator+=] add a scalar to a color

Parameters

scalar	

Returns

Color

4.2.3.18 operator-() [1/2]

```
Color& Color::operator- (
            const Color & other )
```

[operator-] substract two colors

Parameters

other	

Returns

Color

4.2.3.19 operator-() [2/2]

```
Color& Color::operator- (
            const double scalar )
```

[operator-] substract a scalar to a color

Parameters

scalar	

Returns

Color

4.2.3.20 operator-=() [1/2]

```
Color& Color::operator-= (
            const Color & other )
```

[operator-=] substract two colors

Parameters

other	

Returns

Color

4.2.3.21 operator-=() [2/2]

```
Color& Color::operator-= (
            const double scalar )
```

[operator-=] substract a scalar to a color

Parameters

scalar	

Returns

Color

4.2.3.22 operator/() [1/2]

```
Color& Color::operator/ (
            const Color & other )
```

[operator/] divide two colors

Parameters

other	

Returns

Color

4.2.3.23 operator/() [2/2]

```
Color& Color::operator/ (
            const double scalar )
```

[operator/] divide a scalar to a color

Parameters

scalar	

Returns

Color

4.2.3.24 operator/=() [1/2]

```
Color& Color::operator/= (
            const Color & other )
```

[operator/=] divide two colors

Parameters

other	

Returns

Color

4.2.3.25 operator/=() [2/2]

```
Color& Color::operator/= (
            const double scalar )
```

[operator/=] divide a scalar to a color

Parameters

scalar	

Returns

Color

4.2.3.26 operator<<()

```
std::ostream& Color::operator<< (
            std::ostream & os )
```

print a color

Parameters

os	the output stream

Returns

std::ostream

4.2.3.27 operator=() [1/2]

```
Color& Color::operator= (
            Color && other )  [default]
```

[operator=] move assignment operator

Parameters

other	

Returns

Color

4.2.3.28 operator=() [2/2]

```
Color& Color::operator= (
            const Color & other )  [default]
```

[operator=] copy assignment operator

Parameters

other	

Returns

Color

4.2.3.29 operator==()

```
bool Color::operator== (
            const Color & other ) const
```

compare two colors

Parameters

other	

Returns

bool: true if the colors are the same, false otherwise

4.2.3.30 setA()

```
void Color::setA (
            unsigned char a )
```

Setter for the alpha value.

Parameters

a	

4.2.3.31 setB()

```
void Color::setB (
            unsigned char b )
```

Setter for the blue value.

Parameters

b	

4.2.3.32 setG()

```
void Color::setG (
            unsigned char g )
```

Setter for the green value.

Parameters

g	

4.2.3.33 setR()

```
void Color::setR (
            unsigned char r )
```

Setter for the red value.

Parameters

r	

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

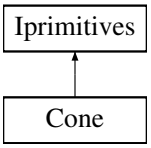
- inc/Color.hpp

4.3 Cone Class Reference

[Cone] class of the cone primitive (inherits from Iprimitives) This class is used to define the cone primitive and to define the methods that the cone must have (ex: intersect, ...)

`#include <Cone.hpp>`

Inheritance diagram for Cone:



Public Member Functions

- Cone ()
  default construct a new Cone object
- Cone (const Vector3D &position, const Vector3D &direction, float radius, float height, Color color)
  Construct a new Cone object.
- Cone (const Cone &other)=default
  Copy construct a new Cone object.
- Cone (Cone &&other)=default
  Move construct a new Cone object.
- ~Cone ()=default
  Destroy the Cone object.
- Cone & operator= (const Cone &other)=default
  Copy assignment operator.
- Cone & operator= (Cone &&other)=default
  Move assignment operator.
- Vector3D getPosition () const
  Get the Position object.
- Vector3D getDirection () const
  Get the Direction object.
- float getRadius () const
  Get the Radius object.
- float getHeight () const
  Get the Height object.
- void setPosition (const Vector3D &position)
  Set the Position object.
- void setDirection (const Vector3D &direction)
  Set the Direction object.
- void setRadius (float radius)
  Set the Radius object.
- void setHeight (float height)
  Set the Height object.
- bool intersect (const Ray &ray, float &t) const override
  method to know if a ray intersect the cone
- Vector3D normalize () const override
  method to normalize the cone
- Vector3D getNormal (const Vector3D &intersection) const override
  Get the Normal object.
- const Color & getColor () const override
  Get the Color object.

4.3.1 Detailed Description

[Cone] class of the cone primitive (inherits from Iprimitives) This class is used to define the cone primitive and to define the methods that the cone must have (ex: intersect, ...)

Parameters

_position	the position of the cone (Vector3D)
_direction	the direction of the cone (Vector3D)
_radius	the radius of the cone (float)
_height	the height of the cone (float)
_color	the color of the cone (Color)

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Cone() [1/3]

```
Cone::Cone (
            const Vector3D & position,
            const Vector3D & direction,
            float radius,
            float height,
            Color color )
```

Construct a new Cone object.

Parameters

position	
direction	
radius	
height	
color	

4.3.2.2 Cone() [2/3]

```
Cone::Cone (
            const Cone & other )  [default]
```

Copy construct a new Cone object.

Parameters

other	

4.3.2.3 Cone() [3/3]

```
Cone::Cone (
            Cone && other )  [default]
```

Move construct a new Cone object.

Parameters

other	

4.3.3 Member Function Documentation

4.3.3.1 getColor()

```
const Color& Cone::getColor ( ) const  [override], [virtual]
```

Get the Color object.

Returns

const Color

Implements Iprimitives.

4.3.3.2 getDirection()

```
Vector3D Cone::getDirection ( ) const
```

Get the Direction object.

Returns

Vector3D

4.3.3.3 getHeight()

```
float Cone::getHeight ( ) const
```

Get the Height object.

Returns

float

4.3.3.4 getNormal()

```
Vector3D Cone::getNormal (
            const Vector3D & intersection ) const  [override], [virtual]
```

Get the Normal object.

Parameters

intersection	

Returns

Vector3D

Implements Iprimitives.

4.3.3.5 getPosition()

```
Vector3D Cone::getPosition ( ) const
```

Get the Position object.

Returns

Vector3D

4.3.3.6 getRadius()

```
float Cone::getRadius ( ) const
```

Get the Radius object.

Returns

float (radius)

4.3.3.7 intersect()

```
bool Cone::intersect (
            const Ray & ray,
            float & t ) const  [override], [virtual]
```

method to know if a ray intersect the cone

Parameters

ray	to test
t	the distance between the ray origin and the intersection (if there is an intersection)

Returns

true|false if the ray intersect the cone

Implements Iprimitives.

4.3.3.8 normalize()

```
Vector3D Cone::normalize ( ) const  [override], [virtual]
```

method to normalize the cone

Returns

Vector3D

Implements Iprimitives.

4.3.3.9 operator=() [1/2]

```
Cone& Cone::operator= (
            Cone && other )  [default]
```

Move assignment operator.

Parameters

other	the cone to move

Returns

Cone

4.3.3.10 operator=() [2/2]

```
Cone& Cone::operator= (
            const Cone & other )  [default]
```

Copy assignment operator.

Parameters

other	the cone to copy

Returns

Cone

4.3.3.11 setDirection()

```
void Cone::setDirection (
            const Vector3D & direction )
```

Set the Direction object.

Parameters

direction	

4.3.3.12 setHeight()

```
void Cone::setHeight (
            float height )
```

Set the Height object.

Parameters

height	

4.3.3.13 setPosition()

```
void Cone::setPosition (
            const Vector3D & position )
```

Set the Position object.

Parameters

position	

4.3.3.14 setRadius()

```
void Cone::setRadius (
            float radius )
```

Set the Radius object.

Parameters

radius	

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

- inc/Cone.hpp

4.4 Config Class Reference

[Config] class of the config This class is used to define the config of the raytracer method factory and thiw class use the libconfig library to parse the config file and to define the methods that the config must have (ex: getRaySetting, ...)

```
#include <Config.hpp>
```

Public Member Functions

- Config (const char *path)

  Construct a new Config object.
- ~Config ()=default

  Destroy the Config object.
- RaySetting * getRaySetting (const std::string name)

  Get the Ray Setting object by name (ex: "Camera", "Light")
- std::vector< Iprimitives * > * getRaySetting ()

  Get the Ray Setting object (ex: "Iprimitives")

4.4.1 Detailed Description

[Config] class of the config This class is used to define the config of the raytracer method factory and thiw class use the libconfig library to parse the config file and to define the methods that the config must have (ex: getRaySetting, ...)

Attention

the config file must be in the format of the libconfig library and install the libconfig library

Parameters

_config	the config (libconfig::Config)

4.4.2 Constructor & Destructor Documentation

4.4.2.1 Config()

```
Config::Config (
            const char * path )
```

Construct a new Config object.

Parameters

path	name of the config file

4.4.3 Member Function Documentation

4.4.3.1 getRaySetting() [1/2]

```
std::vector<Iprimitives *>* Config::getRaySetting ( )
```

Get the Ray Setting object (ex: "Iprimitives")

Returns

std::vector<Iprimitives *>*

4.4.3.2 getRaySetting() [2/2]

```
RaySetting* Config::getRaySetting (
            const std::string name )
```

Get the Ray Setting object by name (ex: "Camera", "Light")

Parameters

name	of the ray setting

Returns

RaySetting∗

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

- inc/Config.hpp

4.5 Core Class Reference

[Core] class of the core This class is used to define the core of the raytracer and to define the methods that the core must have (ex: render, ...)

`#include <Core.hpp>`

Public Member Functions

- Core (const char ∗path)

  Construct a new Core object.
- ∼Core ()

  Destroy the Core object.

4.5.1 Detailed Description

[Core] class of the core This class is used to define the core of the raytracer and to define the methods that the core must have (ex: render, ...)

Parameters

_config	the config of the raytracer (Config)
_camera	the camera of the raytracer (Camera)
_light	the light of the raytracer (Light)
_image	the image of the raytracer (ImagePPM)
_scene	the scene of the raytracer (Scene)
_primitives	the primitives of the raytracer (std::vector<Iprimitives ∗>)

4.5.2 Constructor & Destructor Documentation

4.5.2.1 Core()

```
Core::Core (
            const char * path )
```

Construct a new Core object.

Parameters

path	the path of the config file

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

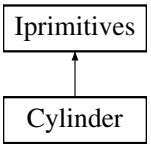
- inc/Core.hpp

4.6 Cylinder Class Reference

[Cylinder] class of the cylinder primitive (inherits from Iprimitives) This class is used to define the cylinder primitive and to define the methods that the cylinder must have (ex: intersect, ...)

```
#include <Cylinder.hpp>
```

Inheritance diagram for Cylinder:



Public Member Functions

- Cylinder ()

  Defaultc onstruct a new Cylinder object.
- Cylinder (const Vector3D &position, const Vector3D &direction, float radius, float height, Color color)

  Construct a new Cylinder object.
- Cylinder (const Cylinder &other)=default

  Copy construct a new Cylinder object.
- Cylinder (Cylinder &&other)=default

  Move construct a new Cylinder object.
- ~Cylinder ()=default

  Destroy the Cylinder object.
- Cylinder & operator= (const Cylinder &other)=default

  Copy assignment operator.
- Cylinder & operator= (Cylinder &&other)=default

  Move assignment operator.
- Vector3D getPosition () const

  Get the Position value.
- float getRadius () const

Get the Radius value.

- float getHeight () const

  Get the Height value.

- const Color & getColor () const override

  Get the Color value.

- void setPosition (const Vector3D &position)

  Set the Position value.

- void setRadius (float radius)

  Set the Radius value.

- void setHeight (float height)

  Set the Height value.

- void setColor (const Color &color)

  Set the Color value.

- bool intersect (const Ray &ray, float &t) const override

  method to know if a ray intersects the cylinder

- Vector3D getNormal (const Vector3D &intersection) const override

  method to get the normal of the cylinder at a given point

- Vector3D normalize () const override

  method to normalize the cylinder

4.6.1 Detailed Description

[Cylinder] class of the cylinder primitive (inherits from Iprimitives) This class is used to define the cylinder primitive and to define the methods that the cylinder must have (ex: intersect, ...)

Parameters

_position	the position of the cylinder (Vector3D)
_radius	the radius of the cylinder (float)
_height	the height of the cylinder (float)
_color	the color of the cylinder (Color)

4.6.2 Constructor & Destructor Documentation

4.6.2.1 Cylinder() [1/3]

```
Cylinder::Cylinder (
            const Vector3D & position,
            const Vector3D & direction,
            float radius,
            float height,
            Color color )
```

Construct a new Cylinder object.

Parameters

position	of the cylinder (Vector3D)
direction	of the cylinder (Vector3D)
radius	of the cylinder (float)
height	of the cylinder (float)
color	of the cylinder (Color)

4.6.2.2 Cylinder() [2/3]

```
Cylinder::Cylinder (
            const Cylinder & other )  [default]
```

Copy construct a new Cylinder object.

Parameters

other	to copy

4.6.2.3 Cylinder() [3/3]

```
Cylinder::Cylinder (
            Cylinder && other )  [default]
```

Move construct a new Cylinder object.

Parameters

other	to move

4.6.3 Member Function Documentation

4.6.3.1 getColor()

```
const Color& Cylinder::getColor ( ) const  [override], [virtual]
```

Get the Color value.

Returns

const Color&

Implements Iprimitives.

4.6.3.2 getHeight()

```
float Cylinder::getHeight ( ) const
```

Get the Height value.

Returns

float

4.6.3.3 getNormal()

```
Vector3D Cylinder::getNormal (
            const Vector3D & intersection ) const  [override], [virtual]
```

method to get the normal of the cylinder at a given point

Parameters

intersection	the point where we want the normal (Vector3D)

Returns

Vector3D the normal of the cylinder at the given point

Implements Iprimitives.

4.6.3.4 getPosition()

```
Vector3D Cylinder::getPosition ( ) const
```

Get the Position value.

Returns

Vector3D

4.6.3.5 getRadius()

```
float Cylinder::getRadius ( ) const
```

Get the Radius value.

Returns

float

4.6.3.6 intersect()

```
bool Cylinder::intersect (
            const Ray & ray,
            float & t ) const  [override], [virtual]
```

method to know if a ray intersects the cylinder

Parameters

ray	to test (Ray)
t	the distance between the ray origin and the intersection (float) (if there is one)

Returns

true|false if the ray intersects the cylinder or not (bool)

Implements Iprimitives.

4.6.3.7 normalize()

```
Vector3D Cylinder::normalize ( ) const  [override], [virtual]
```

method to normalize the cylinder

Returns

Vector3D the normalized cylinder

Implements Iprimitives.

4.6.3.8 operator=() [1/2]

```
Cylinder& Cylinder::operator= (
            const Cylinder & other )  [default]
```

Copy assignment operator.

Parameters

other	to copy

Returns

Cylinder

4.6.3.9 operator=() [2/2]

```
Cylinder& Cylinder::operator= (
            Cylinder && other )  [default]
```

Move assignment operator.

Parameters

other	to move

Returns

Cylinder

4.6.3.10 setColor()

```
void Cylinder::setColor (
            const Color & color )
```

Set the Color value.

Parameters

color	

4.6.3.11 setHeight()

```
void Cylinder::setHeight (
            float height )
```

Set the Height value.

Parameters

height	

4.6.3.12 setPosition()

```
void Cylinder::setPosition (
            const Vector3D & position )
```

Set the Position value.

Parameters

position	

4.6.3.13 setRadius()

```
void Cylinder::setRadius (
            float radius )
```

Set the Radius value.

Parameters

radius	

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

- inc/Cylinder.hpp

4.7 ImagePPM Class Reference

[ImagePPM] class of the image (PPM format) (P3) (ASCII) (RGB) This class is used to define the image and to define the methods that the image must have (ex: createImage, ...)

```
#include <ImagePPM.hpp>
```

Public Member Functions

- ImagePPM ()
  Default construct a new Image PP.
- ImagePPM (int width, int height)
  Construct a new Image PPM.
- ImagePPM (const ImagePPM &copy)=default
  Copy construct a new Image PPM.
- ImagePPM (ImagePPM &&other)=default
  Construct a new Image PPM.
- ~ImagePPM ()=default
  Destroy the Image PPM.

- ImagePPM & operator= (const ImagePPM &copy)=default

  Copy a ImagePPM.
- ImagePPM & operator= (ImagePPM &&other)=default

  Move a ImagePPM.
- void createImage (const char *path)

  Create a Image object when the raytracer is finished.
- void setSize (int width, int height)

  Set the Size value.
- void setWidth (int width)

  Set the Width value.
- void setHeight (int height)

  Set the Height value.
- void setPixels (std::vector< Color * > pixels)

  Set the Pixels value.
- void setPixel (int x, int y, Color *color)

  Set the Pixel value.
- void setPixel (int index, Color *color)

  Set the Pixel value.
- int getWidth () const

  Get the Width value.
- int getHeight () const

  Get the Height value.
- std::vector< Color * > getPixels () const

  Get the Pixels value.
- Color * getPixel (int x, int y) const

  Get the Pixel value.
- Color * getPixel (int index) const

  Get the Pixel value.

4.7.1 Detailed Description

[ImagePPM] class of the image (PPM format) (P3) (ASCII) (RGB) This class is used to define the image and to define the methods that the image must have (ex: createImage, ...)

Parameters

_width	the width of the image (the number of columns)
_height	the height of the image (the number of lines)
_pixels	the pixels of the image (vector of Color)

4.7.2 Constructor & Destructor Documentation

4.7.2.1 ImagePPM() [1/3]

```
ImagePPM::ImagePPM (
            int width,
            int height )  [inline]
```

Construct a new Image PPM.

Parameters

width	of the image
height	of the image

4.7.2.2 ImagePPM() [2/3]

```
ImagePPM::ImagePPM (
            const ImagePPM & copy )  [default]
```

Copy construct a new Image PPM.

Parameters

copy	

4.7.2.3 ImagePPM() [3/3]

```
ImagePPM::ImagePPM (
            ImagePPM && other )  [default]
```

Construct a new Image PPM.

Parameters

other	

4.7.3 Member Function Documentation

4.7.3.1 createImage()

```
void ImagePPM::createImage (
            const char * path )
```

Create a Image object when the raytracer is finished.

Parameters

path	of the image to create

4.7.3.2 getHeight()

```
int ImagePPM::getHeight ( ) const
```

Get the Height value.

Returns

int

4.7.3.3 getPixel() [1/2]

```
Color* ImagePPM::getPixel (
            int index ) const
```

Get the Pixel value.

Parameters

index	of the pixel

Returns

Color*

4.7.3.4 getPixel() [2/2]

```
Color* ImagePPM::getPixel (
            int x,
            int y ) const
```

Get the Pixel value.

Parameters

x	position of the pixel
y	position of the pixel

Returns

Color*

4.7.3.5 getPixels()

```
std::vector<Color *> ImagePPM::getPixels ( ) const
```

Get the Pixels value.

Returns

std::vector<Color *>

4.7.3.6 getWidth()

```
int ImagePPM::getWidth ( ) const
```

Get the Width value.

Returns

int

4.7.3.7 operator=() [1/2]

```
ImagePPM& ImagePPM::operator= (
            const ImagePPM & copy )  [default]
```

Copy a ImagePPM.

Parameters

copy	

Returns

ImagePPM

4.7.3.8 operator=() [2/2]

```
ImagePPM& ImagePPM::operator= (
            ImagePPM && other )  [default]
```

Move a ImagePPM.

Parameters

other	

Returns

ImagePPM

4.7.3.9 setHeight()

```
void ImagePPM::setHeight (
            int height )
```

Set the Height value.

Parameters

height	of the image

4.7.3.10 setPixel() [1/2]

```
void ImagePPM::setPixel (
            int index,
            Color * color )
```

Set the Pixel value.

Parameters

index	of the pixel
color	of the pixel

4.7.3.11 setPixel() [2/2]

```
void ImagePPM::setPixel (
            int x,
```

```
            int y,
            Color * color )
```

Set the Pixel value.

Parameters

x	position of the pixel
y	position of the pixel
color	of the pixel

4.7.3.12 setPixels()

```
void ImagePPM::setPixels (
            std::vector< Color * > pixels )
```

Set the Pixels value.

Parameters

pixels	of the image

4.7.3.13 setSize()

```
void ImagePPM::setSize (
            int width,
            int height )
```

Set the Size value.

Parameters

width	of the image
height	of the image

4.7.3.14 setWidth()

```
void ImagePPM::setWidth (
            int width )
```

Set the Width value.

Parameters

width	of the image

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

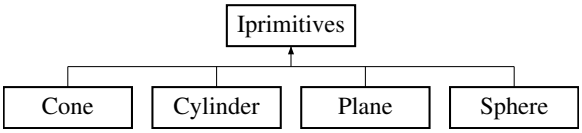
- inc/ImagePPM.hpp

4.8 Iprimitives Class Reference

[Iprimitives] Abstract class of primitives (used to define the type of the primitive) ex: Sphere, Plane, Cube, ... This class is used to define the type of the primitive and to define the methods that the primitive must have (ex: intersect, normalize, getNormal, ...)

`#include <Iprimitives.hpp>`

Inheritance diagram for Iprimitives:



Public Member Functions

- virtual ~Iprimitives ()=default

  Destroy the Iprimitives (virtual because it's an abstract class)
- virtual bool intersect (const Ray &ray, float &t) const =0

  Check if there is an intersection between the ray and the scene.
- virtual Vector3D normalize () const =0

  Normalize the vector to a length of 1 for the calculations of the intersection point and the normal.
- virtual Vector3D getNormal (const Vector3D &intersection) const =0

  Get the Normal of the primitive at the intersection point.
- virtual const Color & getColor () const =0

  Get the Color of the primitive.

4.8.1 Detailed Description

[Iprimitives] Abstract class of primitives (used to define the type of the primitive) ex: Sphere, Plane, Cube, ... This class is used to define the type of the primitive and to define the methods that the primitive must have (ex: intersect, normalize, getNormal, ...)

4.8.2 Member Function Documentation

4.8.2.1 getColor()

```
virtual const Color& Iprimitives::getColor ( ) const  [pure virtual]
```

Get the Color of the primitive.

Returns

const Color&

Implemented in Sphere, Plane, Cylinder, and Cone.

4.8.2.2 getNormal()

```
virtual Vector3D Iprimitives::getNormal (
            const Vector3D & intersection ) const  [pure virtual]
```

Get the Normal of the primitive at the intersection point.

Parameters

intersection	

Returns

Vector3D

Implemented in Sphere, Plane, Cylinder, and Cone.

4.8.2.3 intersect()

```
virtual bool Iprimitives::intersect (
            const Ray & ray,
            float & t ) const  [pure virtual]
```

Check if there is an intersection between the ray and the scene.

Parameters

ray	to check the intersection with the scene
t	the distance between the ray and the intersection point (if there is one)

Returns

true|false if there is an intersection

Implemented in Sphere, Plane, Cylinder, and Cone.

4.8.2.4 normalize()

```
virtual Vector3D Iprimitives::normalize ( ) const  [pure virtual]
```

Normalize the vector to a length of 1 for the calculations of the intersection point and the normal.

Returns

Vector3D

Implemented in Sphere, Plane, Cylinder, and Cone.

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

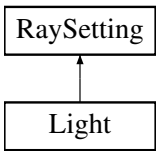
- inc/Iprimitives.hpp

4.9 Light Class Reference

[Light] class of the light (inherits from RaySetting) This class is used to define the light and to define the methods that the light must have (ex: addition, subtraction, ...)

```
#include <Light.hpp>
```

Inheritance diagram for Light:



Public Member Functions

- Light ()=default

  Default construct a new Light object.
- Light (double ambient, double diffuse, const std::vector< Vector3D > &pointLights, const std::vector< Vector3D > &directionalLights)

  Construct a new Light object.
- Light (const Light &other)=default

  Copy construct a new Light object.
- Light (Light &&other)=default

  Move construct a new Light object.
- ~Light ()=default

  Destroy the Light object.
- Light & operator= (const Light &other)=default

Copy a Light object.

- Light & operator= (Light &&other)=default

  Move a Light object.
- Light & operator+= (const Light &other)

  addition of two lights
- Light & operator-= (const Light &other)

  subtraction of two lights
- Light operator+ (const Light &other) const

  addition of two lights
- Light operator- (const Light &other) const

  subtraction of two lights
- bool operator== (const Light &other) const

  compare two lights
- bool operator!= (const Light &other) const

  compare two lights
- double getAmbient () const

  Get the Ambient value.
- double getDiffuse () const

  Get the Diffuse value.
- const std::vector< Vector3D > & getPointLights () const

  Get the Point Lights.
- const std::vector< Vector3D > & getDirectionalLights () const

  Get the Directional Lights.
- void setAmbient (double ambient)

  Set the Ambient value.
- void setDiffuse (double diffuse)

  Set the Diffuse value.
- void setPointLights (const std::vector< Vector3D > &pointLights)

  Set the Point Lights.
- void setDirectionalLights (const std::vector< Vector3D > &directionalLights)

  Set the Directional Lights.
- void addPointLight (const Vector3D &pointLight)

  Add new point light source.
- void addDirectionalLight (const Vector3D &directionalLight)

  Add new directional light source.
- void clear ()

  Clear the light.
- Color getColor (const Vector3D &point, const Vector3D &normal, Color &color) const

  Get the Color of a point with the light settings.
- std::string getType () override

  Get the Type object.

4.9.1 Detailed Description

[Light] class of the light (inherits from RaySetting) This class is used to define the light and to define the methods that the light must have (ex: addition, subtraction, ...)

Parameters

_ambient	the ambient light (0.0 - 1.0)
_diffuse	the diffuse light (0.0 - 1.0)
_pointLights	the point light sources (Vector3D)
_directionalLights	the directional light sources (Vector3D)

4.9.2 Constructor & Destructor Documentation

4.9.2.1 Light() [1/3]

```
Light::Light (
            double ambient,
            double diffuse,
            const std::vector< Vector3D > & pointLights,
            const std::vector< Vector3D > & directionalLights )  [inline]
```

Construct a new Light object.

Parameters

ambient	light (0.0 - 1.0)
diffuse	light (0.0 - 1.0)
pointLights	sources
directionalLights	sources

4.9.2.2 Light() [2/3]

```
Light::Light (
            const Light & other )  [default]
```

Copy construct a new Light object.

Parameters

other	

4.9.2.3 Light() [3/3]

```
Light::Light (
            Light && other )  [default]
```

Move construct a new Light object.

Parameters

other	

4.9.3 Member Function Documentation

4.9.3.1 addDirectionalLight()

```
void Light::addDirectionalLight (
            const Vector3D & directionalLight )
```

Add new directional light source.

Parameters

directionalLight	

4.9.3.2 addPointLight()

```
void Light::addPointLight (
            const Vector3D & pointLight )
```

Add new point light source.

Parameters

pointLight	

4.9.3.3 getAmbient()

```
double Light::getAmbient ( ) const
```

Get the Ambient value.

Returns

double

4.9.3.4 getColor()

```
Color Light::getColor (
            const Vector3D & point,
            const Vector3D & normal,
            Color & color ) const
```

Get the Color of a point with the light settings.

Parameters

point	of the object
normal	of the object
color	of the object

Returns

Color of the object

4.9.3.5 getDiffuse()

```
double Light::getDiffuse ( ) const
```

Get the Diffuse value.

Returns

double

4.9.3.6 getDirectionalLights()

```
const std::vector<Vector3D>& Light::getDirectionalLights ( ) const
```

Get the Directional Lights.

Returns

const std::vector<Vector3D>&

4.9.3.7 getPointLights()

```
const std::vector<Vector3D>& Light::getPointLights ( ) const
```

Get the Point Lights.

Returns

const std::vector<Vector3D>&

4.9.3.8 getType()

```
std::string Light::getType ( )  [inline], [override], [virtual]
```

Get the Type object.

Returns

std::string

Implements RaySetting.

4.9.3.9 operator"!=()

```
bool Light::operator!= (
            const Light & other ) const
```

compare two lights

Parameters

other	

Returns

true|false if the two lights are not the same or not

4.9.3.10 operator+()

```
Light Light::operator+ (
            const Light & other ) const
```

addition of two lights

Parameters

other	

Returns

Light

4.9.3.11 operator+=()

```
Light& Light::operator+= (
            const Light & other )
```

addition of two lights

Parameters

other	

Returns

Light&

4.9.3.12 operator-()

```
Light Light::operator- (
            const Light & other ) const
```

subtraction of two lights

Parameters

other	

Returns

Light

4.9.3.13 operator-=()

```
Light& Light::operator-= (
            const Light & other )
```

subtraction of two lights

Parameters

other	

Returns

Light&

4.9.3.14 operator=() [1/2]

```
Light& Light::operator= (
            const Light & other )  [default]
```

Copy a Light object.

Parameters

other	

Returns

Light&

4.9.3.15 operator=() [2/2]

```
Light& Light::operator= (
            Light && other )  [default]
```

Move a Light object.

Parameters

other	

Returns

Light&

4.9.3.16 operator==()

```
bool Light::operator== (
            const Light & other ) const
```

compare two lights

Parameters

other	

Returns

true|false if the two lights are the same or not

4.9.3.17 setAmbient()

```
void Light::setAmbient (
            double ambient )
```

Set the Ambient value.

Parameters

ambient	

4.9.3.18 setDiffuse()

```
void Light::setDiffuse (
            double diffuse )
```

Set the Diffuse value.

Parameters

diffuse	

4.9.3.19 setDirectionalLights()

```
void Light::setDirectionalLights (
            const std::vector< Vector3D > & directionalLights )
```

Set the Directional Lights.

Parameters

directionalLights	

4.9.3.20 setPointLights()

```
void Light::setPointLights (
            const std::vector< Vector3D > & pointLights )
```

Set the Point Lights.

Parameters

pointLights	

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

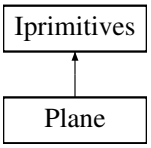
- inc/Light.hpp

4.10 Plane Class Reference

[Plane] class of the plane primitive (inherits from Iprimitives) This class is used to define the plane primitive and to define the methods that the plane must have (ex: intersect, ...)

`#include <Plane.hpp>`

Inheritance diagram for Plane:



Public Member Functions

- Plane ()=default

  Default construct a new Plane object.
- Plane (const int &axis, const double &position, const Color &color)

  Construct a new Plane object.
- Plane (const Plane &other)=default

  Copy construct a new Plane object.
- Plane (Plane &&other)=default

  Move construct a new Plane object.
- ~Plane ()=default

  Destroy the Plane object.
- Plane & operator= (const Plane &other)=default

  Copy assignment operator.
- Plane & operator= (Plane &&other)=default

  Move assignment operator.
- int getAxis () const

  Get the Axis value.
- double getPosition () const

  Get the Position value.
- const Color & getColor () const override

  Get the Color value.
- void setAxis (const int &axis)

Set the Axis value.

- void setPosition (const double &position)

  Set the Position value.

- void setColor (const Color &color)

  Set the Color value.

- bool intersect (const Ray &ray, float &t) const override

  method to check if a ray intersect with the plane

- Vector3D normalize () const override

  method to get the normal of the plane

- Vector3D getNormal (const Vector3D &point) const override

  Get the Normal object at a given point.

4.10.1 Detailed Description

[Plane] class of the plane primitive (inherits from Iprimitives) This class is used to define the plane primitive and to define the methods that the plane must have (ex: intersect, ...)

Parameters

_axis	the axis of the plane (0 = x, 1 = y, 2 = z)
_position	the position of the plane on the axis (double)
_color	the color of the plane (Color)

4.10.2 Constructor & Destructor Documentation

4.10.2.1 Plane() [1/3]

```
Plane::Plane (
            const int & axis,
            const double & position,
            const Color & color )
```

Construct a new Plane object.

Parameters

axis	of the plane (0 = x, 1 = y, 2 = z)
position	of the plane on the axis
color	of the plane

4.10.2.2 Plane() [2/3]

```
Plane::Plane (
```

```
            const Plane & other )  [default]
```

Copy construct a new Plane object.

Parameters

other	

4.10.2.3 Plane() [3/3]

```
Plane::Plane (
            Plane && other )  [default]
```

Move construct a new Plane object.

Parameters

other	

4.10.3 Member Function Documentation

4.10.3.1 getAxis()

```
int Plane::getAxis ( ) const
```

Get the Axis value.

Returns

int

4.10.3.2 getColor()

```
const Color& Plane::getColor ( ) const  [override], [virtual]
```

Get the Color value.

Returns

const Color&

Implements Iprimitives.

4.10.3.3 getNormal()

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

Get the Normal object at a given point.

Parameters

point	to get the normal from

Returns

Vector3D

Implements Iprimitives.

4.10.3.4 getPosition()

```
double Plane::getPosition ( ) const
```

Get the Position value.

Returns

double

4.10.3.5 intersect()

```
bool Plane::intersect (
            const Ray & ray,
            float & t ) const  [override], [virtual]
```

method to check if a ray intersect with the plane

Parameters

ray	to hnow if it intersect with the plane
t	the intersection point if there is one

Returns

true|false if there is an intersection or not

Implements Iprimitives.

4.10.3.6 normalize()

```
Vector3D Plane::normalize ( ) const  [override], [virtual]
```

method to get the normal of the plane

Returns

Vector3D

Implements Iprimitives.

4.10.3.7 operator=() [1/2]

```
Plane& Plane::operator= (
            const Plane & other )  [default]
```

Copy assignment operator.

Parameters

other	

Returns

Plane&

4.10.3.8 operator=() [2/2]

```
Plane& Plane::operator= (
            Plane && other )  [default]
```

Move assignment operator.

Parameters

other	

Returns

Plane&

4.10.3.9 setAxis()

```
void Plane::setAxis (
            const int & axis )
```

Set the Axis value.

Parameters

axis	

4.10.3.10 setColor()

```
void Plane::setColor (
            const Color & color )
```

Set the Color value.

Parameters

color	

4.10.3.11 setPosition()

```
void Plane::setPosition (
            const double & position )
```

Set the Position value.

Parameters

position	

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

- inc/Plane.hpp

4.11 Ray Class Reference

[Ray] Class of Ray (used to define a ray) A ray is defined by an origin and a direction (ex: (0, 0, 0) and (1, 0, 0)) The ray is used to check if there is an intersection between the ray and the scene (ex: if there is an intersection between the ray and a sphere) The ray is also used to check if there is an intersection between the ray and the light (ex: if there is an intersection between the ray and the light, the point is in the shadow) The ray is also used to check if there is an intersection between the ray and the camera (ex: if there is an intersection between the ray and the camera, the point is visible) The ray is also used to check if there is an intersection between the ray and all the objects in the scene (ex: if there is an intersection between the ray and a sphere, the point is visible)

```
#include <Ray.hpp>
```

Public Member Functions

- Ray ()

  Default constructor of Ray (create a ray with origin (0, 0, 0) and direction (0, 0, 0) (null vector))
- Ray (const Vector3D &origin, const Vector3D &direction)

  Construct a new Ray object.
- Ray (const Ray &other)=default

  Copy constructor of Ray.
- Ray (Ray &&other)

  Move constructor of Ray.
- ~Ray ()=default

  Destroy the Ray object.
- Ray & operator= (const Ray &other)=default

  Move operator of Ray.
- Ray & operator= (Ray &&other)

  Move operator of Ray.
- const Vector3D & getOrigin () const

  Get the Origin value.
- const Vector3D & getDirection () const

  Get the Direction value.
- void setOrigin (const Vector3D &origin)

  Set the Origin value.
- void setDirection (const Vector3D &direction)

  Set the Direction value.

4.11.1 Detailed Description

[Ray] Class of Ray (used to define a ray) A ray is defined by an origin and a direction (ex: (0, 0, 0) and (1, 0, 0)) The ray is used to check if there is an intersection between the ray and the scene (ex: if there is an intersection between the ray and a sphere) The ray is also used to check if there is an intersection between the ray and the light (ex: if there is an intersection between the ray and the light, the point is in the shadow) The ray is also used to check if there is an intersection between the ray and the camera (ex: if there is an intersection between the ray and the camera, the point is visible) The ray is also used to check if there is an intersection between the ray and all the objects in the scene (ex: if there is an intersection between the ray and a sphere, the point is visible)

Parameters

_origin	the origin of the ray (ex: (0, 0, 0)) (Vector3D)
_direction	the direction of the ray (ex: (1, 0, 0)) (Vector3D) (must be normalized)

4.11.2 Constructor & Destructor Documentation

4.11.2.1 Ray() [1/3]

```
Ray::Ray (
            const Vector3D & origin,
            const Vector3D & direction )
```

Construct a new Ray object.

Parameters

origin	of the ray
direction	of the ray

4.11.2.2 Ray() [2/3]

```
Ray::Ray (
            const Ray & other )  [default]
```

Copy constructor of Ray.

Parameters

other	

4.11.2.3 Ray() [3/3]

```
Ray::Ray (
            Ray && other )
```

Move constructor of Ray.

Parameters

other	

4.11.3 Member Function Documentation

4.11.3.1 getDirection()

```
const Vector3D& Ray::getDirection ( ) const
```

Get the Direction value.

Returns

const Vector3D&

4.11.3.2 getOrigin()

```
const Vector3D& Ray::getOrigin ( ) const
```

Get the Origin value.

Returns

const Vector3D&

4.11.3.3 operator=() [1/2]

```
Ray& Ray::operator= (
            const Ray & other )  [default]
```

Move operator of Ray.

Parameters

other	

Returns

Ray&

4.11.3.4 operator=() [2/2]

```
Ray& Ray::operator= (
            Ray && other )
```

Move operator of Ray.

Parameters

other	

Returns

Ray&

4.11.3.5 setDirection()

```
void Ray::setDirection (
            const Vector3D & direction )
```

Set the Direction value.

Parameters

direction	

4.11.3.6 setOrigin()

```
void Ray::setOrigin (
            const Vector3D & origin )
```

Set the Origin value.

Parameters

origin	

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

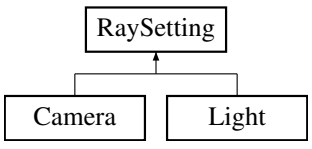
- inc/Ray.hpp

4.12 RaySetting Class Reference

[RaySetting] class it's abstract class of the ray setting This class is used to define Camera and Light

```
#include <RaySetting.hpp>
```

Inheritance diagram for RaySetting:



Public Member Functions

- virtual ~RaySetting ()=default

  Destroy the RaySetting object (it's virtual because it's abstract class)
- virtual std::string getType ()=0

  Get the Type object.

4.12.1 Detailed Description

[RaySetting] class it's abstract class of the ray setting This class is used to define Camera and Light

4.12.2 Member Function Documentation

4.12.2.1 getType()

```
virtual std::string RaySetting::getType ( )  [pure virtual]
```

Get the Type object.

Returns

std::string

Implemented in Light, and Camera.

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

- inc/RaySetting.hpp

4.13 Scene Class Reference

[Scene] class of the scene This class is used to define the scene (the camera, the lights and the primitives) and to define the methods that the scene must have (ex: addPrimitive, render, ...)

```
#include <Scene.hpp>
```

Public Member Functions

- Scene ()=default
  Construct a new Scene object.
- Scene (const Scene &scene)=default
  Copy construct a new Scene object.
- Scene (Scene &scene)=default
  Move construct a new Scene object.
- ~Scene ()=default
  Destroy the Scene object.
- Scene & operator= (Scene &scene)=default
  Copy assignment operator.
- Scene & operator= (const Scene &scene)=default
  Move assignment operator.
- Camera getCamera () const
  Get the Camera object.

- Light getLight () const

  Get the Light object.
- std::vector< Iprimitives * > * getIprimitives () const

  Get the Iprimitives object.
- void setCamera (Camera camera)

  Set the Camera object.
- void setLight (Light light)

  Set the Light object.
- void setIprimitives (std::vector< Iprimitives * > *iprimitives)

  Set the Iprimitives object.
- void addPrimitive (Iprimitives *primitive)

  Add new primitive to the scene.
- bool intersect (const Ray &ray, Vector3D &intersection) const

  Method for all primitives in the scene.
- bool intersect (const Ray &ray, Vector3D &intersection, Vector3D &noraml, Color &color) const

  Method for all primitives in the scene.
- Color trace (const Ray &ray, int depth) const

  trace method for all primitives in the scene
- void render (ImagePPM &image) const

  Render the scene.
- void putRendering (int x, int y, int width, int height) const

  Put the rendering of the scene.

4.13.1 Detailed Description

[Scene] class of the scene This class is used to define the scene (the camera, the lights and the primitives) and to define the methods that the scene must have (ex: addPrimitive, render, ...)

Parameters

_camera	the camera of the scene
_light	the light of the scene
_iprimitives	the primitives of the scene

4.13.2 Constructor & Destructor Documentation

4.13.2.1 Scene() [1/2]

```
Scene::Scene (
            const Scene & scene )  [default]
```

Copy construct a new Scene object.

Parameters

scene	

4.13.2.2 Scene() [2/2]

```
Scene::Scene (
            Scene & scene )  [default]
```

Move construct a new Scene object.

Parameters

scene	

4.13.3 Member Function Documentation

4.13.3.1 addPrimitive()

```
void Scene::addPrimitive (
            Iprimitives * primitive )
```

Add new primitive to the scene.

Parameters

primitive	

4.13.3.2 getCamera()

```
Camera Scene::getCamera ( ) const
```

Get the Camera object.

Returns

Camera

4.13.3.3 getIprimitives()

```
std::vector<Iprimitives *>* Scene::getIprimitives ( ) const
```

Get the Iprimitives object.

Returns

std::vector<Iprimitives *>

4.13.3.4 getLight()

```
Light Scene::getLight ( ) const
```

Get the Light object.

Returns

Light

4.13.3.5 intersect() [1/2]

```
bool Scene::intersect (
            const Ray & ray,
            Vector3D & intersection ) const
```

Method for all primitives in the scene.

Parameters

ray	to intersect with the scene
intersection	point if there is one

Returns

true|false if there is an intersection or not

4.13.3.6 intersect() [2/2]

```
bool Scene::intersect (
            const Ray & ray,
            Vector3D & intersection,
            Vector3D & noraml,
            Color & color ) const
```

Method for all primitives in the scene.

Parameters

ray	to intersect with the scene
intersection	point if there is one
normal	of the primitive at the intersection point if there is one
color	of the primitive at the intersection point if there is one

Returns

true|false if there is an intersection or not

4.13.3.7 operator=() [1/2]

```
Scene& Scene::operator= (
            const Scene & scene )  [default]
```

Move assignment operator.

Parameters

scene	

Returns

Scene&

4.13.3.8 operator=() [2/2]

```
Scene& Scene::operator= (
            Scene & scene )  [default]
```

Copy assignment operator.

Parameters

scene	

Returns

Scene&

4.13.3.9 putRendering()

```
void Scene::putRendering (
            int x,
            int y,
            int width,
            int height ) const
```

Put the rendering of the scene.

Parameters

x	coordinate of the pixel
y	coordinate of the pixel
width	the pixel
height	of the pixel

4.13.3.10 render()

```
void Scene::render (
            ImagePPM & image ) const
```

Render the scene.

Parameters

image	to render (the image will be modified in this method)

4.13.3.11 setCamera()

```
void Scene::setCamera (
            Camera camera )
```

Set the Camera object.

Parameters

camera	

4.13.3.12 setIprimitives()

```
void Scene::setIprimitives (
            std::vector< Iprimitives * > * iprimitives )
```

Set the Iprimitives object.

Parameters

iprimitives	

4.13.3.13 setLight()

```
void Scene::setLight (
            Light light )
```

Set the Light object.

Parameters

light	

4.13.3.14 trace()

```
Color Scene::trace (
            const Ray & ray,
            int depth ) const
```

trace method for all primitives in the scene

Parameters

ray	to trace with the scene (from the camera)
depth	of the ray (for reflection)

Returns

Color at this pixel (the color of the primitive)

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

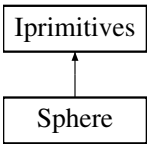
- inc/Scene.hpp

4.14 Sphere Class Reference

[Sphere] class of the sphere primitive (inherits from Iprimitives) This class is used to define the sphere primitive and to define the methods that the sphere must have (ex: intersect, ...)

```
#include <Sphere.hpp>
```

Inheritance diagram for Sphere:



Public Member Functions

- Sphere ()

  Default construct a new Sphere object.
- Sphere (const Vector3D &center, const double &radius, Color color)

  Construct a new Sphere object.
- Sphere (const Sphere &other)=default

  Copy construt a new Sphere object.
- Sphere (Sphere &&other)=default

  Move construct a new Sphere object.
- ~Sphere ()=default

  Destroy the Sphere object.
- Sphere & operator= (const Sphere &other)=default

  Copy assignment operator.
- Sphere & operator= (Sphere &&other)=default

  Move assignment operator.
- const Vector3D & getCenter () const

  Get the Center of the sphere.
- const double & getRadius () const

  Get the Radius of the sphere.
- void setCenter (const Vector3D &center)

  Set the Center of the sphere.
- void setRadius (const double &radius)

  Set the Radius of the sphere.
- const Color & getColor () const override

  Get the Color of the sphere.
- bool intersect (const Ray &ray, float &t) const override

  method to intersect a ray with a sphere (if there is an intersection) and to get the intersection point
- Vector3D normalize () const override

  method to normalize the sphere
- Vector3D getNormal (const Vector3D &point) const override

  method to get the normal of the sphere at a given point

4.14.1 Detailed Description

[Sphere] class of the sphere primitive (inherits from Iprimitives) This class is used to define the sphere primitive and to define the methods that the sphere must have (ex: intersect, ...)

Parameters

_center	the center of the sphere (Vector3D)
_radius	the radius of the sphere (double)
_color	the color of the sphere (Color)

4.14.2 Constructor & Destructor Documentation

4.14.2.1 Sphere() [1/3]

```
Sphere::Sphere (
            const Vector3D & center,
            const double & radius,
            Color color )  [inline]
```

Construct a new Sphere object.

Parameters

center	
radius	
color	

4.14.2.2 Sphere() [2/3]

```
Sphere::Sphere (
            const Sphere & other )  [default]
```

Copy construt a new Sphere object.

Parameters

other	

4.14.2.3 Sphere() [3/3]

```
Sphere::Sphere (
            Sphere && other )  [default]
```

Move construct a new Sphere object.

Parameters

other	

4.14.3 Member Function Documentation

4.14.3.1 getCenter()

```
const Vector3D& Sphere::getCenter ( ) const
```

Get the Center of the sphere.

Returns

const Vector3D&

4.14.3.2 getColor()

```
const Color& Sphere::getColor ( ) const  [inline], [override], [virtual]
```

Get the Color of the sphere.

Returns

const Color&

Implements Iprimitives.

4.14.3.3 getNormal()

```
Vector3D Sphere::getNormal (
            const Vector3D & point ) const  [inline], [override], [virtual]
```

method to get the normal of the sphere at a given point

Parameters

point	

Returns

Vector3D

Implements Iprimitives.

4.14.3.4 getRadius()

```
const double& Sphere::getRadius ( ) const
```

Get the Radius of the sphere.

Returns

const double&

4.14.3.5 intersect()

```
bool Sphere::intersect (
            const Ray & ray,
            float & t ) const  [inline], [override], [virtual]
```

method to intersect a ray with a sphere (if there is an intersection) and to get the intersection point

Parameters

ray	to intersect with the scene
intersection	point if there is one

Returns

true|false if there is an intersection or not

Implements Iprimitives.

4.14.3.6 normalize()

```
Vector3D Sphere::normalize ( ) const  [inline], [override], [virtual]
```

method to normalize the sphere

Returns

Vector3D

Implements Iprimitives.

4.14.3.7 operator=() [1/2]

```
Sphere& Sphere::operator= (
            const Sphere & other )  [default]
```

Copy assignment operator.

Parameters

other	

Returns

Sphere&

4.14.3.8 operator=() [2/2]

```
Sphere& Sphere::operator= (
            Sphere && other )  [default]
```

Move assignment operator.

Parameters

other	

Returns

Sphere&

4.14.3.9 setCenter()

```
void Sphere::setCenter (
            const Vector3D & center )
```

Set the Center of the sphere.

Parameters

center	

4.14.3.10 setRadius()

```
void Sphere::setRadius (
            const double & radius )
```

Set the Radius of the sphere.

Parameters

radius	

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

- inc/Sphere.hpp

4.15 Vector3D Class Reference

[Vector3D] class of the vector3D This class is used to define the vector3D and to define the methods that the vector3D must have (ex: addition, subtraction, ...)

```
#include <Vector3D.hpp>
```

Public Member Functions

- Vector3D (double x=0, double y=0, double z=0)

  Construct a new Vector3D object.
- Vector3D (const Vector3D &other)=default

  Construct a new Vector3D object.
- Vector3D (Vector3D &&other)=default

  Construct a new Vector3D object.
- ~Vector3D ()=default

  Destroy the Vector3D object.
- Vector3D & operator= (const Vector3D &other)=default

  Copy assignment operator.
- Vector3D & operator= (Vector3D &&other)=default

  Move assignment operator.
- Vector3D operator+ (const Vector3D &other) const

  Add two vectors.
- Vector3D operator- (const Vector3D &other) const

  Substract two vectors.
- Vector3D operator* (const Vector3D &other) const

  Multiply two vectors.
- Vector3D operator/ (const Vector3D &other) const

  Divide two vectors.
- Vector3D & operator+= (const Vector3D &other)

  Add two vectors.
- Vector3D & operator-= (const Vector3D &other)

  Substract two vectors.
- Vector3D & operator*= (const Vector3D &other)

  Multiply two vectors.
- Vector3D & operator/= (const Vector3D &other)

  Divide two vectors.
- Vector3D operator+ (double scalar) const

  Add a scalar to a vector.
- Vector3D operator- (double scalar) const

  Substract a scalar to a vector.
- Vector3D operator* (double scalar) const

  Multiply a scalar to a vector.
- Vector3D operator/ (double scalar) const

  Divide a scalar to a vector.
- Vector3D & operator+= (double scalar)

  Add a scalar to a vector.
- Vector3D & operator-= (double scalar)

  Substract a scalar to a vector.
- Vector3D & operator*= (double scalar)

Multiply a scalar to a vector.

- Vector3D & operator/= (double scalar)

  Divide a scalar to a vector.
- Vector3D operator- () const

  Negate a vector.
- bool operator== (const Vector3D &other) const

  Compare two vectors.
- bool operator!= (const Vector3D &other) const

  Compare two vectors.
- bool operator< (const Vector3D &other) const

  Compare two vectors.
- bool operator<= (const Vector3D &other) const

  Compare two vectors.
- bool operator> (const Vector3D &other) const

  Compare two vectors.
- bool operator>= (const Vector3D &other) const

  Compare two vectors.
- std::ostream & operator<< (std::ostream &os) const

  Overload for std::ostream.
- std::vector< double > getVector () const

  Get the Vector object.
- double getX () const

  Get the X object.
- double getY () const

  Get the Y object.
- double getZ () const

  Get the Z object.
- void setVector (double x, double y, double z)

  Set the Vector object.
- void setX (double x)

  Set X in the vector.
- void setY (double y)

  Set Y in the vector.
- void setZ (double z)

  Set Z in the vector.
- void translate (const Vector3D &v)

  translate the vector with another vector
- void rotateX (double angle)

  rotate the vector on the x axis
- void rotateY (double angle)

  rotate the vector on the y axis
- void rotateZ (double angle)

  rotate the vector on the z axis
- Vector3D normalize () const

  normalize the vector
- Vector3D cross (const Vector3D &v) const

  calculate the cross product of two vectors
- float dot (const Vector3D &other) const

  calculate the dot product of two vectors
- double length () const

  calculate the length of the vector

- Vector3D squaredLength () const

  calculate the squared length of the vector (faster than length)
- Vector3D reflect (const Vector3D &normal) const

  calculate the reflection vector of this vector
- Vector3D magnitude () const

  calculate the magnitude vector of this vector
- Vector3D lerp (const Vector3D &v, float t) const

  linear interpolation between two vectors
- float distance (const Vector3D &v) const

  calculate the distance between two vectors
- float angle (const Vector3D &v) const

  calculate the angle between two vectors
- Vector3D min (const Vector3D &v) const

  return the minimum values between two vectors
- Vector3D max (const Vector3D &v) const

  return the maximum values between two vectors
- Vector3D clamp (const Vector3D &min, const Vector3D &max)

  clamp the values of the vector between min and max
- float somme () const

  calculate the sum of the vector
- float substract () const

  calculate the substraction of the vector
- float produit () const

  calculate the product of the vector
- float division () const

  calculate the division of the vector

4.15.1 Detailed Description

[Vector3D] class of the vector3D This class is used to define the vector3D and to define the methods that the vector3D must have (ex: addition, subtraction, ...)

Parameters

↩ _↩ *x*	the x coordinate
↩ _↩ *y*	the y coordinate
↩ _↩ *z*	the z coordinate

4.15.2 Constructor & Destructor Documentation

4.15.2.1 Vector3D() [1/3]

```
Vector3D::Vector3D (
            double x = 0,
            double y = 0,
            double z = 0 )  [inline]
```

Construct a new Vector3D object.

Parameters

x	coordinate
y	coordinate
z	coordinate

4.15.2.2 Vector3D() [2/3]

```
Vector3D::Vector3D (
            const Vector3D & other )  [default]
```

Construct a new Vector3D object.

Parameters

other	

4.15.2.3 Vector3D() [3/3]

```
Vector3D::Vector3D (
            Vector3D && other )  [default]
```

Construct a new Vector3D object.

Parameters

other	

4.15.3 Member Function Documentation

4.15.3.1 angle()

```
float Vector3D::angle (
            const Vector3D & v ) const
```

calculate the angle between two vectors

Parameters

v	

Returns

float

4.15.3.2 clamp()

```
Vector3D Vector3D::clamp (
            const Vector3D & min,
            const Vector3D & max )
```

clamp the values of the vector between min and max

Parameters

min	
max	

Returns

Vector3D

4.15.3.3 cross()

```
Vector3D Vector3D::cross (
            const Vector3D & v ) const
```

calculate the cross product of two vectors

Parameters

v	

Returns

Vector3D

4.15.3.4 distance()

```
float Vector3D::distance (
            const Vector3D & v ) const
```

calculate the distance between two vectors

Parameters

v	

Returns

float

4.15.3.5 division()

```
float Vector3D::division ( ) const
```

calculate the division of the vector

Returns

float

4.15.3.6 dot()

```
float Vector3D::dot (
            const Vector3D & other ) const
```

calculate the dot product of two vectors

Parameters

other	

Returns

float

4.15.3.7 getVector()

```
std::vector<double> Vector3D::getVector ( ) const
```

Get the Vector object.

Returns

std::vector<double>

4.15.3.8 getX()

```
double Vector3D::getX ( ) const
```

Get the X object.

Returns

double

4.15.3.9 getY()

```
double Vector3D::getY ( ) const
```

Get the Y object.

Returns

double

4.15.3.10 getZ()

```
double Vector3D::getZ ( ) const
```

Get the Z object.

Returns

double

4.15.3.11 length()

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

calculate the length of the vector

Returns

double

4.15.3.12 lerp()

```
Vector3D Vector3D::lerp (
            const Vector3D & v,
            float t ) const
```

linear interpolation between two vectors

Parameters

v	other vector to interpolate with this one
t	interpolation value between 0.0f and 1.0f

Returns

Vector3D

4.15.3.13 magnitude()

```
Vector3D Vector3D::magnitude ( ) const
```

calculate the magnitude vector of this vector

Returns

Vector3D

4.15.3.14 max()

```
Vector3D Vector3D::max (
            const Vector3D & v ) const
```

return the maximum values between two vectors

Parameters

v	

Returns

Vector3D

4.15.3.15 min()

```
Vector3D Vector3D::min (
            const Vector3D & v ) const
```

return the minimum values between two vectors

Parameters

v	

Returns

Vector3D

4.15.3.16 normalize()

```
Vector3D Vector3D::normalize ( ) const
```

normalize the vector

Returns

Vector3D

4.15.3.17 operator"!=()

```
bool Vector3D::operator!= (
            const Vector3D & other ) const
```

Compare two vectors.

Parameters

other	

Returns

true|false if the two vectors are different or not

4.15.3.18 operator*() [1/2]

```
Vector3D Vector3D::operator* (
            const Vector3D & other ) const
```

Multiply two vectors.

Parameters

other	

Returns

Vector3D

4.15.3.19 operator*() [2/2]

```
Vector3D Vector3D::operator* (
            double scalar ) const
```

Multiply a scalar to a vector.

Parameters

scalar	

Returns

Vector3D

4.15.3.20 operator*=() [1/2]

```
Vector3D& Vector3D::operator*= (
            const Vector3D & other )
```

Multiply two vectors.

Parameters

other	

Returns

Vector3D&

4.15.3.21 operator*=() [2/2]

```
Vector3D& Vector3D::operator*= (
            double scalar )
```

Multiply a scalar to a vector.

Parameters

scalar	

Returns

Vector3D&

4.15.3.22 operator+() [1/2]

```
Vector3D Vector3D::operator+ (
            const Vector3D & other ) const
```

Add two vectors.

Parameters

other	

Returns

Vector3D

4.15.3.23 operator+() [2/2]

```
Vector3D Vector3D::operator+ (
            double scalar ) const
```

Add a scalar to a vector.

Parameters

scalar	

Returns

Vector3D

4.15.3.24 operator+=() [1/2]

```
Vector3D& Vector3D::operator+= (
            const Vector3D & other )
```

Add two vectors.

Parameters

other	

Returns

Vector3D&

4.15.3.25 operator+=() [2/2]

```
Vector3D& Vector3D::operator+= (
            double scalar )
```

Add a scalar to a vector.

Parameters

scalar	

Returns

Vector3D&

4.15.3.26 operator-() [1/3]

```
Vector3D Vector3D::operator- ( ) const
```

Negate a vector.

Returns

Vector3D

4.15.3.27 operator-() [2/3]

```
Vector3D Vector3D::operator- (
            const Vector3D & other ) const
```

Substract two vectors.

Parameters

other	

Returns

Vector3D

4.15.3.28 operator-() [3/3]

```
Vector3D Vector3D::operator- (
            double scalar ) const
```

Substract a scalar to a vector.

Parameters

scalar	

Returns

Vector3D

4.15.3.29 operator-=() [1/2]

```
Vector3D& Vector3D::operator-= (
            const Vector3D & other )
```

Substract two vectors.

Parameters

other	

Returns

Vector3D&

4.15.3.30 operator-=() [2/2]

```
Vector3D& Vector3D::operator-= (
            double scalar )
```

Substract a scalar to a vector.

Parameters

scalar	

Returns

Vector3D&

4.15.3.31 operator/() [1/2]

```
Vector3D Vector3D::operator/ (
            const Vector3D & other ) const
```

Divide two vectors.

Parameters

other	

Returns

Vector3D

4.15.3.32 operator/() [2/2]

```
Vector3D Vector3D::operator/ (
            double scalar ) const
```

Divide a scalar to a vector.

Parameters

scalar	

Returns

Vector3D

4.15.3.33 operator/=() [1/2]

```
Vector3D& Vector3D::operator/= (
            const Vector3D & other )
```

Divide two vectors.

Parameters

other	

Returns

Vector3D&

4.15.3.34 operator/=() [2/2]

```
Vector3D& Vector3D::operator/= (
            double scalar )
```

Divide a scalar to a vector.

Parameters

scalar	

Returns

Vector3D&

4.15.3.35 operator<()

```
bool Vector3D::operator< (
            const Vector3D & other ) const
```

Compare two vectors.

Parameters

other	

Returns

true|false if the first vector is smaller than the second or not

4.15.3.36 operator<<()

```
std::ostream& Vector3D::operator<< (
            std::ostream & os ) const
```

Overload for std::ostream.

Parameters

os	is the stream to write in

Returns

std::ostream&

4.15.3.37 operator<=()

```
bool Vector3D::operator<= (
            const Vector3D & other ) const
```

Compare two vectors.

Parameters

other	

Returns

true|false if the first vector is smaller or equal than the second or not

4.15.3.38 operator=() [1/2]

```
Vector3D& Vector3D::operator= (
            const Vector3D & other )  [default]
```

Copy assignment operator.

Parameters

other	

Returns

Vector3D&

4.15.3.39 operator=() [2/2]

```
Vector3D& Vector3D::operator= (
            Vector3D && other )  [default]
```

Move assignment operator.

Parameters

other	

Returns

Vector3D&

4.15.3.40 operator==()

```
bool Vector3D::operator== (
            const Vector3D & other ) const
```

Compare two vectors.

Parameters

other	

Returns

true|false if the two vectors are equal or not

4.15.3.41 operator>()

```
bool Vector3D::operator> (
            const Vector3D & other ) const
```

Compare two vectors.

Parameters

other	

Returns

true|false if the first vector is greater than the second or not

4.15.3.42 operator>=()

```
bool Vector3D::operator>= (
            const Vector3D & other ) const
```

Compare two vectors.

Parameters

other	

Returns

true|false if the first vector is greater or equal than the second or not

4.15.3.43 produit()

```
float Vector3D::produit ( ) const
```

calculate the product of the vector

Returns

float

4.15.3.44 reflect()

```
Vector3D Vector3D::reflect (
            const Vector3D & normal ) const
```

calculate the reflection vector of this vector

Parameters

normal	

Returns

Vector3D

4.15.3.45 rotateX()

```
void Vector3D::rotateX (
            double angle )
```

rotate the vector on the x axis

Parameters

angle	

4.15.3.46 rotateY()

```
void Vector3D::rotateY (
            double angle )
```

rotate the vector on the y axis

Parameters

angle	

4.15.3.47 rotateZ()

```
void Vector3D::rotateZ (
            double angle )
```

rotate the vector on the z axis

Parameters

angle	

4.15.3.48 setVector()

```
void Vector3D::setVector (
            double x,
            double y,
            double z )
```

Set the Vector object.

Parameters

x	
y	
z	

4.15.3.49 setX()

```
void Vector3D::setX (
            double x )
```

Set X in the vector.

Parameters

x	

4.15.3.50 setY()

```
void Vector3D::setY (
            double y )
```

Set Y in the vector.

Parameters

y	

4.15.3.51 setZ()

```
void Vector3D::setZ (
            double z )
```

Set Z in the vector.

Parameters

z	

4.15.3.52 somme()

```
float Vector3D::somme ( ) const
```

calculate the sum of the vector

Returns

float

4.15.3.53 squaredLength()

```
Vector3D Vector3D::squaredLength ( ) const
```

calculate the squared length of the vector (faster than length)

Returns

Vector3D

4.15.3.54 substract()

```
float Vector3D::substract ( ) const
```

calculate the substraction of the vector

Returns

float

4.15.3.55 translate()

```
void Vector3D::translate (
            const Vector3D & v )
```

translate the vector with another vector

Parameters

v	

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

- inc/Vector3D.hpp

Index