Raytracer

Generated by Doxygen 1.10.0

Chapter 1

raytracer

This project is a raytracer written in C++. The goal of this project is to implement a raytracer that can render scenes with `spheres`, `planes`, `lights`... The raytracer supports features such as `translation`, `rotation`, and drop `shadows`. The raytracer outputs the rendered image to a PPM file.

1.1 development

1. Clone the repository.

```
git clone git@github.com:Jabolol/raytracer.git .
```

1. Create a build directory.

```
mkdir build && cd build
```

1. Build the project.

```
cmake .. -GNinja -DBUILD_DOC=OFF -DBUILD_TEST=OFF
```

1. Run the raytracer with a provided scene.

```
./raytracer --config ../scenes/cornell.cfg > cornell.ppm
```

1.2 features

1.2.1 must

[!TIP]\ The config file can be found at `config/must.cfg`

- [x] sphere
- [x] plane
- [x] translation
- [x] directional light
- [x] ambient light
- [x] flat color
- [x] add primitive to scene
- [x] set up lighting
- [x] set up camera
- [x] output to ppm

1.2.2 should

[!TIP]\ The config file can be found at `config/should.cfg`

- [x] cylinder
- [x] cone
- [x] rotation
- [x] drop shadows

1.2.3 could

[!TIP]\ The config file can be found at `config/could.cfg`

- [x] multiple directional lights (0.5)
- [x] colored lights (0.5)
- [x] transparency (0.5)
- [x] reflection (1)
- [x] refraction (1)
- [x] texturing from file (1)
- [x] texturing from procedural chessboard (1)
- [x] texturing from procedural perlin noise (1)
- [x] import a scene in a scene (2)
- [x] set up antialiasing through supersampling (0.5)
- [x] space partitioning (2)
- [x] scene preview using a fast renderer (2)

1.2.4 bonus

[!TIP]\ The config file can be found at `config/bonus.cfg`

- [x] quad box
- [x] motion blur
- [x] depth of field
- [x] quadrilaterals
- [x] constant medium (fog)
- [x] tinted dielectric

1.2.5 cornell box

[!TIP]\ The config file can be found at `config/cornell.cfg`

Chapter 2

Concept Index

2.1 Concepts

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

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

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

Chapter 6

Concept Documentation

6.1 Raytracer::Config::isValidEnum Concept Reference

6.1.1 Concept definition

```
template<typename T, typename E>
concept Raytracer::Config::isValidEnum =  std::is_enum_v<T> && std::is_same_v<T, E>
```

6.2 Raytracer::Utils::isNumerical Concept Reference

6.2.1 Concept definition

```
template<typename T>
concept Raytracer::Utils::isNumerical =  requires(T) { std::is_arithmetic_v<T>; }
```

6.3 Raytracer::Utils::isPositive Concept Reference

6.3.1 Concept definition

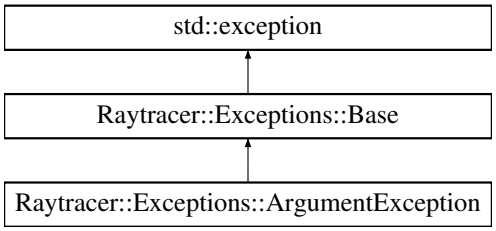
```
template<typename T>
concept Raytracer::Utils::isPositive =  requires(T t) { t > 0; }
```

Chapter 7

Class Documentation

7.1 Raytracer::Exceptions::ArgumentException Class Reference

Inheritance diagram for Raytracer::Exceptions::ArgumentException:



Public Member Functions

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

Public Member Functions inherited from Raytracer::Exceptions::Base

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

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

- /Users/riosj1/Code/raytracer/include/exceptions/Argument.hpp

7.2 Raytracer::Utils::AxisAlignedBBox Class Reference

Public Member Functions

- AxisAlignedBBox (const Interval &x, const Interval &y, const Interval &z)

  Construct a new AxisAlignedBBox object.
- AxisAlignedBBox (const Point3 &a, const Point3 &b)

  Construct a new AxisAlignedBBox object.
- AxisAlignedBBox (const AxisAlignedBBox &a, const AxisAlignedBBox &b)

  Construct a new AxisAlignedBBox object.
- const Interval & axisInterval (int n) const

  Get the interval along the x-axis.
- bool hit (const Core::Ray &ray, Interval interval) const

  Check if the ray hits the AxisAlignedBBox.
- int longestAxis () const

  Get the longest axis of the AxisAlignedBBox.
- void padToMinimum ()

  Pad the AxisAlignedBBox to the minimum size.

Static Public Attributes

- static const AxisAlignedBBox Empty

  Empty AxisAlignedBBox.
- static const AxisAlignedBBox Universe

  Universe AxisAlignedBBox.

7.2.1 Constructor & Destructor Documentation

7.2.1.1 AxisAlignedBBox() [1/3]

```
Raytracer::Utils::AxisAlignedBBox::AxisAlignedBBox (
            const Interval & x,
            const Interval & y,
            const Interval & z )
```

Construct a new AxisAlignedBBox object.

This function constructs a new AxisAlignedBBox object with the given x, y, and z intervals. The AxisAlignedBBox is an axis-aligned bounding box that represents a box in 3D space. The x, y, and z intervals represent the intervals of the box along the x, y, and z axes.

Parameters

x	The interval along the x-axis.
y	The interval along the y-axis.
z	The interval along the z-axis.

Returns

A new AxisAlignedBBox object.

7.2.1.2 AxisAlignedBBox() [2/3]

```
Raytracer::Utils::AxisAlignedBBox::AxisAlignedBBox (
            const Point3 & a,
            const Point3 & b )
```

Construct a new AxisAlignedBBox object.

This function constructs a new AxisAlignedBBox object with the given points. The AxisAlignedBBox is an axis-aligned bounding box that represents a box in 3D space. The box is defined by the two points.

Parameters

a	The first point of the box.
b	The second point of the box.

Returns

A new AxisAlignedBBox object.

7.2.1.3 AxisAlignedBBox() [3/3]

```
Raytracer::Utils::AxisAlignedBBox::AxisAlignedBBox (
            const AxisAlignedBBox & a,
            const AxisAlignedBBox & b )
```

Construct a new AxisAlignedBBox object.

This function constructs a new AxisAlignedBBox object by combining the given AxisAlignedBBoxes. The new AxisAlignedBBox is the smallest AxisAlignedBBox that contains both of the given AxisAlignedBBoxes.

Parameters

a	The first AxisAlignedBBox.
b	The second AxisAlignedBBox.

Returns

A new AxisAlignedBBox object.

7.2.2 Member Function Documentation

7.2.2.1 axisInterval()

```
const Raytracer::Utils::Interval & Raytracer::Utils::AxisAlignedBBox::axisInterval (
            int n ) const
```

Get the interval along the x-axis.

This function returns the interval along the x-axis.

Parameters

n	The axis to get the interval for.

Returns

The interval along the x-axis.

7.2.2.2 hit()

```
bool Raytracer::Utils::AxisAlignedBBox::hit (
            const Core::Ray & ray,
            Interval interval ) const
```

Check if the ray hits the AxisAlignedBBox.

This function checks if the ray hits the AxisAlignedBBox. The function returns true if the ray hits the AxisAlignedBBox. The function returns false if the ray does not hit the AxisAlignedBBox.

Parameters

ray	The ray to check for hits.
interval	The interval to check for hits.

Returns

True if the ray hits the AxisAlignedBBox, false otherwise.

7.2.2.3 longestAxis()

```
int Raytracer::Utils::AxisAlignedBBox::longestAxis ( ) const
```

Get the longest axis of the AxisAlignedBBox.

This function returns the index of the longest axis of the AxisAlignedBBox. The function returns 0 if the x-axis is the longest axis. The function returns 1 if the y-axis is the longest axis. The function returns 2 if the z-axis is the longest axis.

Returns

The index of the longest axis of the AxisAlignedBBox.

7.2.2.4 padToMinimum()

```
void Raytracer::Utils::AxisAlignedBBox::padToMinimum ( )
```

Pad the AxisAlignedBBox to the minimum size.

This function pads the AxisAlignedBBox to the minimum size. The function expands the intervals of the AxisAlignedBBox to the minimum size if the intervals are smaller than the minimum size.

Returns

void

7.2.3 Member Data Documentation

7.2.3.1 Empty

```
const Raytracer::Utils::AxisAlignedBBox Raytracer::Utils::AxisAlignedBBox::Empty  [static]
```

Initial value:

```
=
        AxisAlignedBBox(Interval::Empty, Interval::Empty, Interval::Empty)
```

Empty AxisAlignedBBox.

This constant represents an empty AxisAlignedBBox.

7.2.3.2 Universe

```
const Raytracer::Utils::AxisAlignedBBox Raytracer::Utils::AxisAlignedBBox::Universe  [static]
```

Initial value:

```
= AxisAlignedBBox(
        Interval::Universe, Interval::Universe, Interval::Universe)
```

Universe AxisAlignedBBox.

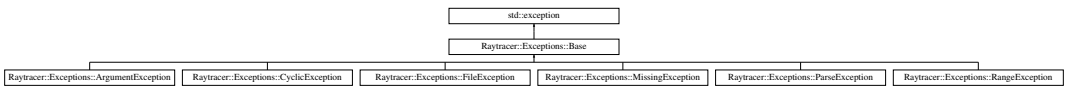
This constant represents the universe AxisAlignedBBox.

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

- /Users/riosj1/Code/raytracer/include/utils/AxisAlignedBBox.hpp
- /Users/riosj1/Code/raytracer/sources/utils/AxisAlignedBBox.cpp

7.3 Raytracer::Exceptions::Base Class Reference

Inheritance diagram for Raytracer::Exceptions::Base:



Public Member Functions

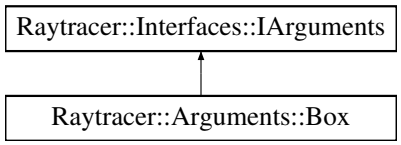
- **Base** (const std::string &message)
- virtual const char * **what** () const noexcept override

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

- /Users/riosj1/Code/raytracer/include/exceptions/Base.hpp

7.4 Raytracer::Arguments::Box Class Reference

Inheritance diagram for Raytracer::Arguments::Box:



Public Member Functions

- **Box** (Utils::Point3 pointOne, Utils::Point3 pointTwo, std::shared_ptr< Interfaces::IMaterial > material)
- **GET_SET** (Utils::Point3, pointOne)
- **GET_SET** (Utils::Point3, pointTwo)
- **GET_SET** (std::shared_ptr< Interfaces::IMaterial >, material)
- **ARG_KIND** (ArgumentKind::ARG_BOX)

Public Member Functions inherited from Raytracer::Interfaces::IArguments

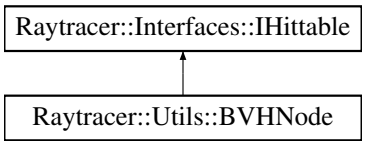
- virtual Arguments::ArgumentKind **kind** () const =0

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

- /Users/riosj1/Code/raytracer/include/arguments/Shapes.hpp

7.5 Raytracer::Utils::BVHNode Class Reference

Inheritance diagram for Raytracer::Utils::BVHNode:



Public Member Functions

- BVHNode (Core::Scene list)

  Construct a new BVHNode object.
- **BVHNode** (std::vector< std::shared_ptr< Interfaces::IHittable > > &objects, size_t start, size_t end)
- bool hit (const Core::Ray &ray, Interval interval, Core::Payload &payload) const override

  Check if the ray hits the BVHNode.
- AxisAlignedBBox boundingBox () const override

  Get the bounding box of the BVHNode.

Static Public Member Functions

- static bool boxCompare (const std::shared_ptr< Interfaces::IHittable > &a, const std::shared_ptr< Interfaces::IHittable > &b, int axis)

  Compare two objects based on the given axis.
- static bool boxXCompare (const std::shared_ptr< Interfaces::IHittable > &a, const std::shared_ptr< Interfaces::IHittable > &b)

  Compare two objects based on the x-axis.
- static bool boxYCompare (const std::shared_ptr< Interfaces::IHittable > &a, const std::shared_ptr< Interfaces::IHittable > &b)

  Compare two objects based on the y-axis.
- static bool boxZCompare (const std::shared_ptr< Interfaces::IHittable > &a, const std::shared_ptr< Interfaces::IHittable > &b)

  Compare two objects based on the z-axis.

7.5.1 Constructor & Destructor Documentation

7.5.1.1 BVHNode()

```
Raytracer::Utils::BVHNode::BVHNode (
            Core::Scene list )
```

Construct a new BVHNode object.

This function constructs a new BVHNode object with the given list of objects. The BVHNode is a bounding volume hierarchy node that represents a node in a BVH tree. The BVH tree is a binary tree that is used to accelerate ray tracing. The BVHNode is constructed from the given list of objects.

Parameters

list	The list of objects.

Returns

A new BVHNode object.

7.5.2 Member Function Documentation

7.5.2.1 boundingBox()

```
Raytracer::Utils::AxisAlignedBBox Raytracer::Utils::BVHNode::boundingBox ( ) const  [override],
```

[virtual]

Get the bounding box of the BVHNode.

This function returns the bounding box of the BVHNode.

Returns

The bounding box of the BVHNode.

Implements Raytracer::Interfaces::IHittable.

7.5.2.2 boxCompare()

```
bool Raytracer::Utils::BVHNode::boxCompare (
            const std::shared_ptr< Interfaces::IHittable > & a,
            const std::shared_ptr< Interfaces::IHittable > & b,
            int axis )  [static]
```

Compare two objects based on the given axis.

This function compares two objects based on the given axis. The function returns true if the first object is less than the second object based on the given axis. The function returns false otherwise.

Parameters

a	The first object.
b	The second object.
axis	The axis to compare the objects on.

Returns

true if the first object is less than the second object based on the given axis, false otherwise.

7.5.2.3 boxXCompare()

```
bool Raytracer::Utils::BVHNode::boxXCompare (
            const std::shared_ptr< Interfaces::IHittable > & a,
            const std::shared_ptr< Interfaces::IHittable > & b )  [static]
```

Compare two objects based on the x-axis.

This function compares two objects based on the x-axis. The function returns true if the first object is less than the second object based on the x-axis. The function returns false otherwise.

Parameters

a	The first object.
b	The second object.

Returns

true if the first object is less than the second object based on the x-axis, false otherwise.

7.5.2.4 boxYCompare()

```
bool Raytracer::Utils::BVHNode::boxYCompare (
            const std::shared_ptr< Interfaces::IHittable > & a,
            const std::shared_ptr< Interfaces::IHittable > & b )  [static]
```

Compare two objects based on the y-axis.

This function compares two objects based on the y-axis. The function returns true if the first object is less than the second object based on the y-axis. The function returns false otherwise.

Parameters

a	The first object.
b	The second object.

Returns

true if the first object is less than the second object based on the y-axis, false otherwise.

7.5.2.5 boxZCompare()

```
bool Raytracer::Utils::BVHNode::boxZCompare (
            const std::shared_ptr< Interfaces::IHittable > & a,
            const std::shared_ptr< Interfaces::IHittable > & b )  [static]
```

Compare two objects based on the z-axis.

This function compares two objects based on the z-axis. The function returns true if the first object is less than the second object based on the z-axis. The function returns false otherwise.

Parameters

a	The first object.
b	The second object.

Returns

true if the first object is less than the second object based on the z-axis, false otherwise.

7.5.2.6 hit()

```
bool Raytracer::Utils::BVHNode::hit (
            const Core::Ray & ray,
```

```
        Interval interval,
        Core::Payload & payload ) const  [override], [virtual]
```

Check if the ray hits the BVHNode.

This function checks if the ray hits the BVHNode. The function returns true if the ray hits the BVHNode. The function returns false if the ray does not hit the BVHNode. The function updates the payload with the hit information.

Parameters

ray	The ray to check for hits.
interval	The interval to check for hits.
payload	The payload to update with the hit information.

Returns

true if the ray hits the BVHNode, false otherwise.

Implements Raytracer::Interfaces::IHittable.

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

- /Users/riosj1/Code/raytracer/include/utils/BVHNode.hpp
- /Users/riosj1/Code/raytracer/sources/utils/BVHNode.cpp

7.6 Raytracer::Core::Camera Class Reference

Public Member Functions

- void setup ()
  Set up the camera with the given parameters.
- void render (const Interfaces::IHittable &world)
  Render the scene with the given camera.
- Core::Ray getRay (double u, double v) const
  Get the ray for the given pixel.
- Utils::Vec3 sampleSquare () const
  Sample a square.
- Utils::Vec3 sampleDisk (double radius) const
  Sample a disk.
- Utils::Vec3 sampleDefocusDisk () const
  Sample the defocus disk.
- Utils::Color rayColor (const Ray ray, int depth, const Interfaces::IHittable &world) const
  Get the color of the ray.
- void progress (const std::chrono::steady_clock::time_point &start, int j) const
  Print the progress of the rendering.

7.6.1 Member Function Documentation

7.6.1.1 getRay()

```
Raytracer::Core::Ray Raytracer::Core::Camera::getRay (
            double i,
            double j ) const
```

Get the ray for the given pixel.

This function calculates the sample location based on the pixel location and the pixel delta u and v vectors. The origin is set to the center of the camera if the defocus angle is less than or equal to 0. Otherwise, the origin is set to a point on the defocus disk. The direction is set to the sample location minus the origin. The time is set to a random double.

Parameters

i	The x coordinate of the pixel.
j	The y coordinate of the pixel.

Returns

The ray for the given pixel.

7.6.1.2 progress()

```
void Raytracer::Core::Camera::progress (
            const std::chrono::steady_clock::time_point & start,
            int j ) const
```

Print the progress of the rendering.

This function prints the progress of the rendering to the standard error stream. The progress is printed as a percentage and the time elapsed is printed in seconds.

Parameters

start	The start time of the rendering.
j	The y coordinate of the pixel.

Returns

void

7.6.1.3 rayColor()

```
Raytracer::Utils::Color Raytracer::Core::Camera::rayColor (
            const Ray ray,
```

```
            int depth,
            const Interfaces::IHittable & world ) const
```

Get the color of the ray.

This function returns the color of the ray based on the depth and the world. If the depth is less than or equal to 0, the function returns black. If the ray does not hit anything in the world, the function returns the background color. If the ray scatters, the function returns the emission color plus the scatter color.

Parameters

ray	The ray to get the color of.
depth	The depth of the ray.
world	The world to get the color from.

Returns

The color of the ray.

7.6.1.4 render()

```
void Raytracer::Core::Camera::render (
            const Interfaces::IHittable & world )
```

Render the scene with the given camera.

This function sets up the camera and prints the PPM header. It then loops through each pixel in the image and calculates the pixel color based on the number of samples per pixel. The pixel color is then written to the output stream.

Parameters

world	The world to render.

Returns

void

7.6.1.5 sampleDefocusDisk()

```
Raytracer::Utils::Vec3 Raytracer::Core::Camera::sampleDefocusDisk ( ) const
```

Sample the defocus disk.

This function returns a random point in the defocus disk.

Returns

A random point in the defocus disk.

7.6.1.6 sampleDisk()

```
Raytracer::Utils::Vec3 Raytracer::Core::Camera::sampleDisk (
            double radius ) const
```

Sample a disk.

This function returns a random point in a disk centered at the origin with a radius of 1.

Parameters

radius	The radius of the disk.

Returns

A random point in a disk.

7.6.1.7 sampleSquare()

```
Raytracer::Utils::Vec3 Raytracer::Core::Camera::sampleSquare ( ) const
```

Sample a square.

This function returns a random point in a square centered at the origin with a side length of 1.

Returns

A random point in a square.

7.6.1.8 setup()

```
void Raytracer::Core::Camera::setup ( )
```

Set up the camera with the given parameters.

This function calculates the image height based on the aspect ratio and the image width. It also calculates the viewport height and width based on the vertical field of view and the focus distance. The pixel sample scale is calculated based on the number of samples per pixel. The camera's center is set to the look from point. The camera's u, v, and w vectors are calculated based on the look from, look at, and v up points. The viewport u and v vectors are calculated based on the viewport width and height and the u and v vectors. The pixel delta u and v vectors are calculated based on the viewport u and v vectors and the image width and height. The pixel zero location is calculated based on the viewport upper left corner and the pixel delta u and v vectors. The defocus disk u and v vectors are calculated based on the defocus angle and the u and v vectors.

Returns

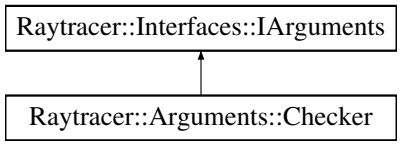
void

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

- /Users/riosj1/Code/raytracer/include/core/Camera.hpp
- /Users/riosj1/Code/raytracer/sources/core/Camera.cpp

7.7 Raytracer::Arguments::Checker Class Reference

Inheritance diagram for Raytracer::Arguments::Checker:



Public Member Functions

- **Checker** (double scale, Utils::Vec3 color1, Utils::Vec3 color2)
- **Checker** (double scale, std::shared_ptr< Interfaces::ITexture > texture1, std::shared_ptr< Interfaces::ITexture > texture2)
- **GET_SET** (double, scale)
- **GET_SET** (Utils::Vec3, color1)
- **GET_SET** (Utils::Vec3, color2)
- **GET_SET** (std::shared_ptr< Interfaces::ITexture >, texture1)
- **GET_SET** (std::shared_ptr< Interfaces::ITexture >, texture2)
- **ARG_KIND** (_kind)

Public Member Functions inherited from Raytracer::Interfaces::IArguments

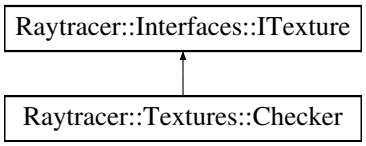
- virtual Arguments::ArgumentKind **kind** () const =0

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

- /Users/riosj1/Code/raytracer/include/arguments/Textures.hpp

7.8 Raytracer::Textures::Checker Class Reference

Inheritance diagram for Raytracer::Textures::Checker:



Public Member Functions

- Checker (double scale, std::shared_ptr< Interfaces::ITexture > even, std::shared_ptr< Interfaces::ITexture > odd)

  Construct a new Checker object.
- Checker (double scale, const Utils::Color &a, const Utils::Color &b)

  Construct a new Checker object.
- Utils::Color value (double u, double v, const Utils::Point3 &point) const override

  Get the value of the checker texture.

7.8.1 Constructor & Destructor Documentation

7.8.1.1 Checker() [1/2]

```
Raytracer::Textures::Checker::Checker (
            double scale,
            std::shared_ptr< Interfaces::ITexture > even,
            std::shared_ptr< Interfaces::ITexture > odd )
```

Construct a new Checker object.

This function constructs a new Checker object with the given scale, even texture, and odd texture.

Parameters

scale	The scale of the checker texture.
even	The even texture of the checker texture.
odd	The odd texture of the checker texture.

Returns

A new Checker object.

7.8.1.2 Checker() [2/2]

```
Raytracer::Textures::Checker::Checker (
            double scale,
            const Utils::Color & a,
            const Utils::Color & b )
```

Construct a new Checker object.

This function constructs a new Checker object with the given scale, even color, and odd color.

Parameters

scale	The scale of the checker texture.
a	The even color of the checker texture.
b	The odd color of the checker texture.

Returns

A new Checker object.

7.8.2 Member Function Documentation

7.8.2.1 value()

```
Raytracer::Utils::Color Raytracer::Textures::Checker::value (
            double u,
```

```
double v,
const Utils::Point3 & point ) const  [override], [virtual]
```

Get the value of the checker texture.

This function returns the value of the checker texture at the given UV coordinates and point.

Parameters

u	The U coordinate.
v	The V coordinate.
point	The point.

Returns

The value of the checker texture.

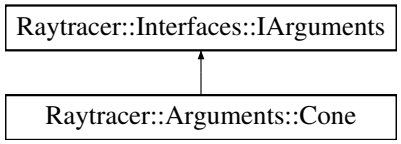
Implements Raytracer::Interfaces::ITexture.

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

- /Users/riosj1/Code/raytracer/include/textures/Checker.hpp
- /Users/riosj1/Code/raytracer/sources/textures/Checker.cpp

7.9 Raytracer::Arguments::Cone Class Reference

Inheritance diagram for Raytracer::Arguments::Cone:



Public Member Functions

- **Cone** (Utils::Point3 &center, double radius, double height, std::shared_ptr< Interfaces::IMaterial > material)
- **GET_SET** (Utils::Point3, center)
- **GET_SET** (double, radius)
- **GET_SET** (double, height)
- **GET_SET** (std::shared_ptr< Interfaces::IMaterial >, material)
- **ARG_KIND** (ArgumentKind::ARG_CONE)

Public Member Functions inherited from Raytracer::Interfaces::IArguments

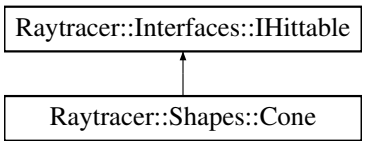
- virtual Arguments::ArgumentKind **kind** () const =0

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

- /Users/riosj1/Code/raytracer/include/arguments/Shapes.hpp

7.10 Raytracer::Shapes::Cone Class Reference

Inheritance diagram for Raytracer::Shapes::Cone:



Public Member Functions

- Cone (const Utils::Point3 &center, double radius, double height, std::shared_ptr< Interfaces::IMaterial > material)

  Construct a new Cone object.
- virtual bool hit (const Core::Ray &ray, Utils::Interval interval, Core::Payload &payload) const override

  Check if the ray hits the cone.
- virtual Utils::AxisAlignedBBox boundingBox () const override

  Get the bounding box of the cone.

7.10.1 Constructor & Destructor Documentation

7.10.1.1 Cone()

```
Raytracer::Shapes::Cone::Cone (
            const Utils::Point3 & center,
            double radius,
            double height,
            std::shared_ptr< Interfaces::IMaterial > material )
```

Construct a new Cone object.

This function constructs a new Cone object with the given center, radius, height, and material. The cone is centered at the given center with the given radius, height, and material.

Parameters

center	The center of the cone.
radius	The radius of the cone.
height	The height of the cone.
material	The material of the cone.

Returns

A new Cone object.

7.10.2 Member Function Documentation

7.10.2.1 boundingBox()

```
Raytracer::Utils::AxisAlignedBBox Raytracer::Shapes::Cone::boundingBox ( ) const  [override],
[virtual]
```

Get the bounding box of the cone.

This function returns the bounding box of the cone.

Returns

The bounding box of the cone.

Implements Raytracer::Interfaces::IHittable.

7.10.2.2 hit()

```
bool Raytracer::Shapes::Cone::hit (
            const Core::Ray & ray,
            Utils::Interval interval,
            Core::Payload & payload ) const  [override], [virtual]
```

Check if the ray hits the cone.

This function checks if the ray hits the cone. The function returns true if the ray hits the cone. The function returns false if the ray does not hit the cone. The function updates the payload with the hit information.

Parameters

ray	The ray to check for hits.
interval	The interval to check for hits.
payload	The payload to update with the hit information.

Returns

true if the ray hits the cone, false otherwise.

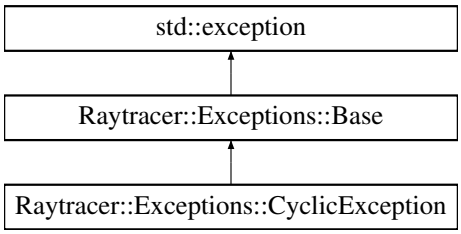
Implements Raytracer::Interfaces::IHittable.

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

- /Users/riosj1/Code/raytracer/include/shapes/Cone.hpp
- /Users/riosj1/Code/raytracer/sources/shapes/Cone.cpp

7.11 Raytracer::Exceptions::CyclicException Class Reference

Inheritance diagram for Raytracer::Exceptions::CyclicException:



Public Member Functions

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

Public Member Functions inherited from Raytracer::Exceptions::Base

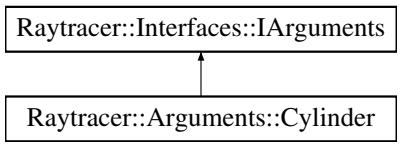
- **Base** (const std::string &message)
- virtual const char ∗ **what** () const noexcept override

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

- /Users/riosj1/Code/raytracer/include/exceptions/Cyclic.hpp

7.12 Raytracer::Arguments::Cylinder Class Reference

Inheritance diagram for Raytracer::Arguments::Cylinder:



Public Member Functions

- **Cylinder** (Utils::Point3 &center, double radius, double height, std::shared_ptr< Interfaces::IMaterial > material)
- **GET_SET** (Utils::Point3, center)
- **GET_SET** (double, radius)
- **GET_SET** (double, height)
- **GET_SET** (std::shared_ptr< Interfaces::IMaterial >, material)
- **ARG_KIND** (ArgumentKind::ARG_CYLINDER)

Public Member Functions inherited from Raytracer::Interfaces::IArguments

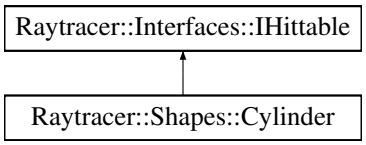
- virtual Arguments::ArgumentKind **kind** () const =0

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

- /Users/riosj1/Code/raytracer/include/arguments/Shapes.hpp

7.13 Raytracer::Shapes::Cylinder Class Reference

Inheritance diagram for Raytracer::Shapes::Cylinder:



Public Member Functions

- Cylinder (const Utils::Point3 &center, double radius, double height, std::shared_ptr< Interfaces::IMaterial > material)

  Construct a new Cylinder object.
- virtual bool hit (const Core::Ray &ray, Utils::Interval interval, Core::Payload &payload) const override

  Check if the ray hits the cylinder.
- virtual Utils::AxisAlignedBBox boundingBox () const override

  Get the bounding box of the cylinder.

7.13.1 Constructor & Destructor Documentation

7.13.1.1 Cylinder()

```
Raytracer::Shapes::Cylinder::Cylinder (
            const Utils::Point3 & center,
            double radius,
            double height,
            std::shared_ptr< Interfaces::IMaterial > material )
```

Construct a new Cylinder object.

This function constructs a new Cylinder object with the given center, radius, height, and material. The cylinder is centered at the given center with the given radius, height, and material.

Parameters

center	The center of the cylinder.
radius	The radius of the cylinder.
height	The height of the cylinder.
material	The material of the cylinder.

Returns

A new Cylinder object.

7.13.2 Member Function Documentation

7.13.2.1 boundingBox()

```
Raytracer::Utils::AxisAlignedBBox Raytracer::Shapes::Cylinder::boundingBox ( ) const  [override], [virtual]
```

Get the bounding box of the cylinder.

This function returns the bounding box of the cylinder.

Returns

The bounding box of the cylinder.

Implements Raytracer::Interfaces::IHittable.

7.13.2.2 hit()

```
bool Raytracer::Shapes::Cylinder::hit (
            const Core::Ray & ray,
            Utils::Interval interval,
            Core::Payload & payload ) const  [override], [virtual]
```

Check if the ray hits the cylinder.

This function checks if the ray hits the cylinder. The function returns true if the ray hits the cylinder. The function returns false if the ray does not hit the cylinder. The function updates the payload with the hit information.

Parameters

ray	The ray to check for hits.
interval	The interval to check for hits.
payload	The payload to update with the hit information.

Returns

true if the ray hits the cylinder, false otherwise.

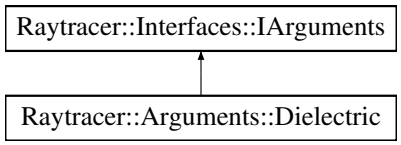
Implements Raytracer::Interfaces::IHittable.

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

- /Users/riosj1/Code/raytracer/include/shapes/Cylinder.hpp
- /Users/riosj1/Code/raytracer/sources/shapes/Cylinder.cpp

7.14 Raytracer::Arguments::Dielectric Class Reference

Inheritance diagram for Raytracer::Arguments::Dielectric:



Public Member Functions

- **Dielectric** (double refractionIndex)
- **Dielectric** (double refractionIndex, Utils::Color color)
- **GET_SET** (double, refractionIndex)
- **GET_SET** (Utils::Color, color)
- **ARG_KIND** (_kind)

Public Member Functions inherited from Raytracer::Interfaces::IArguments

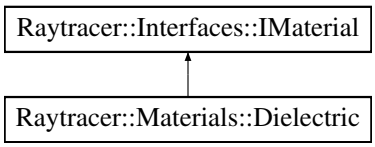
- virtual Arguments::ArgumentKind **kind** () const =0

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

- /Users/riosj1/Code/raytracer/include/arguments/Materials.hpp

7.15 Raytracer::Materials::Dielectric Class Reference

Inheritance diagram for Raytracer::Materials::Dielectric:



Public Member Functions

- Dielectric (double refractionIndex)

  Construct a new Dielectric object.
- Dielectric (double refractionIndex, const Utils::Color &albedo)

  Construct a new Dielectric object.
- bool scatter (const Core::Ray &ray, const Core::Payload &payload, Utils::Color &attenuation, Core::Ray &scattered) const override

  Scatter the ray with the dielectric material.
- Utils::Color emitted (double u, double v, const Utils::Point3 &point) const override

  Calculate the refracted ray.

Static Public Member Functions

- static double reflectance (double cosine, double index)

  Calculate the reflectance of the dielectric material.

7.15.1 Constructor & Destructor Documentation

7.15.1.1 Dielectric() [1/2]

```
Raytracer::Materials::Dielectric::Dielectric (
            double refractionIndex )
```

Construct a new Dielectric object.

This function constructs a new Dielectric object with the given refraction index.

Parameters

refractionIndex	The refraction index of the dielectric.

Returns

A new Dielectric object.

7.15.1.2 Dielectric() [2/2]

```
Raytracer::Materials::Dielectric::Dielectric (
            double refractionIndex,
            const Utils::Color & albedo )
```

Construct a new Dielectric object.

This function constructs a new Dielectric object with the given refraction index and albedo.

Parameters

refractionIndex	The refraction index of the dielectric.
albedo	The albedo of the dielectric.

Returns

A new Dielectric object.

7.15.2 Member Function Documentation

7.15.2.1 emitted()

```
Raytracer::Utils::Color Raytracer::Materials::Dielectric::emitted (
            double u,
```

```
            double v,
            const Utils::Point3 & point ) const  [override], [virtual]
```

Calculate the refracted ray.

This function calculates the refracted ray. The function returns the refracted ray.

Parameters

uv	The unit vector.
normal	The normal vector.
index	The refraction index.

Returns

The refracted ray.

Implements Raytracer::Interfaces::IMaterial.

7.15.2.2 reflectance()

```
double Raytracer::Materials::Dielectric::reflectance (
            double cosine,
            double index )  [static]
```

Calculate the reflectance of the dielectric material.

This function calculates the reflectance of the dielectric material. The function returns the reflectance of the dielectric material.

Parameters

cosine	The cosine of the angle.
index	The refraction index.

Returns

The reflectance of the dielectric material.

7.15.2.3 scatter()

```
bool Raytracer::Materials::Dielectric::scatter (
            const Core::Ray & ray,
            const Core::Payload & payload,
            Utils::Color & attenuation,
            Core::Ray & scattered ) const  [override], [virtual]
```

Scatter the ray with the dielectric material.

This function scatters the ray with the dielectric material. The function returns true if the ray is scattered. The function returns false if the ray is not scattered. The function updates the attenuation and scattered ray.

Parameters

ray	The ray to scatter.
payload	The payload of the ray.
attenuation	The attenuation of the ray.
scattered	The scattered ray.

Returns

true if the ray is scattered, false otherwise.

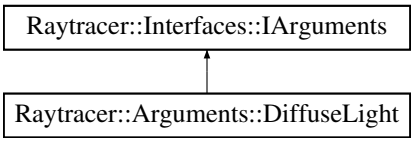
Implements Raytracer::Interfaces::IMaterial.

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

- /Users/riosj1/Code/raytracer/include/materials/Dielectric.hpp
- /Users/riosj1/Code/raytracer/sources/materials/Dielectric.cpp

7.16 Raytracer::Arguments::DiffuseLight Class Reference

Inheritance diagram for Raytracer::Arguments::DiffuseLight:



Public Member Functions

- **DiffuseLight** (Utils::Vec3 color)
- **DiffuseLight** (std::shared_ptr< Interfaces::ITexture > texture)
- **GET_SET** (Utils::Vec3, color)
- **GET_SET** (std::shared_ptr< Interfaces::ITexture >, texture)
- **ARG_KIND** (_kind)

Public Member Functions inherited from Raytracer::Interfaces::IArguments

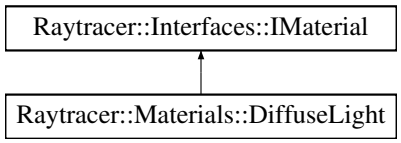
- virtual Arguments::ArgumentKind **kind** () const =0

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

- /Users/riosj1/Code/raytracer/include/arguments/Materials.hpp

7.17 Raytracer::Materials::DiffuseLight Class Reference

Inheritance diagram for Raytracer::Materials::DiffuseLight:



Public Member Functions

- DiffuseLight (std::shared_ptr< Interfaces::ITexture > texture)

  Construct a new DiffuseLight object.
- DiffuseLight (const Utils::Color &color)

  Construct a new DiffuseLight object.
- bool scatter (const Core::Ray &ray, const Core::Payload &payload, Utils::Color &attenuation, Core::Ray &scattered) const override

  Scatter the ray with the diffuse light material.
- Utils::Color emitted (double u, double v, const Utils::Point3 &point) const override

  Emitted light of the diffuse light material.

7.17.1 Constructor & Destructor Documentation

7.17.1.1 DiffuseLight() [1/2]

```
Raytracer::Materials::DiffuseLight::DiffuseLight (
            std::shared_ptr< Interfaces::ITexture > texture )
```

Construct a new DiffuseLight object.

This function constructs a new DiffuseLight object with the given texture. The diffuse light material emits light with the given texture.

Parameters

texture	The texture of the diffuse light.

Returns

A new DiffuseLight object.

7.17.1.2 DiffuseLight() [2/2]

```
Raytracer::Materials::DiffuseLight::DiffuseLight (
            const Utils::Color & color )
```

Construct a new DiffuseLight object.

This function constructs a new DiffuseLight object with the given color. The diffuse light material emits light with the given color.

Parameters

color	The color of the diffuse light.

Returns

A new DiffuseLight object.

7.17.2 Member Function Documentation

7.17.2.1 emitted()

```
Raytracer::Utils::Color Raytracer::Materials::DiffuseLight::emitted (
            double u,
            double v,
            const Utils::Point3 & point ) const  [override], [virtual]
```

Emitted light of the diffuse light material.

This function returns the emitted light of the diffuse light material. The function returns the emitted light of the material at the given point.

Parameters

u	The u coordinate of the texture.
v	The v coordinate of the texture.
point	The point to get the emitted light from.

Returns

The emitted light of the material.

Implements Raytracer::Interfaces::IMaterial.

7.17.2.2 scatter()

```
bool Raytracer::Materials::DiffuseLight::scatter (
            const Core::Ray & ray,
            const Core::Payload & payload,
            Utils::Color & attenuation,
            Core::Ray & scattered ) const  [override], [virtual]
```

Scatter the ray with the diffuse light material.

This function scatters the ray with the diffuse light material. The function returns true if the ray is scattered. The function returns false if the ray is not scattered. The function updates the attenuation and scattered ray.

Parameters

ray	The ray to scatter.
payload	The payload of the ray.
attenuation	The attenuation of the ray.
scattered	The scattered ray.

Returns

true if the ray is scattered, false otherwise.

Implements Raytracer::Interfaces::IMaterial.

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

- /Users/riosj1/Code/raytracer/include/materials/DiffuseLight.hpp
- /Users/riosj1/Code/raytracer/sources/materials/DiffuseLight.cpp

7.18 Raytracer::Config::Factory Class Reference

Static Public Member Functions

- template<typename I , typename E >
  requires isValidEnum<E, ConfigTextures>
  static std::shared_ptr< I > **get** (const std::string &name, std::shared_ptr< Interfaces::IArguments > args)
- template<typename I , typename E >
  requires isValidEnum<E, ConfigEffects>
  static std::shared_ptr< I > **get** (const std::string &name, std::shared_ptr< Interfaces::IArguments > args)
- template<typename I , typename E >
  requires isValidEnum<E, ConfigMaterials>
  static std::shared_ptr< I > **get** (const std::string &name, std::shared_ptr< Interfaces::IArguments > args)
- template<typename I , typename E >
  requires isValidEnum<E, ConfigShapes>
  static std::shared_ptr< I > **get** (const std::string &name, std::shared_ptr< Interfaces::IArguments > args)

Static Public Attributes

- static FactoryMap< Raytracer::Interfaces::ITexture, ConfigTextures > textures
  Factory map for textures.
- static FactoryMap< Raytracer::Interfaces::IHittable, ConfigEffects > effects
  Factory map for effects.
- static FactoryMap< Raytracer::Interfaces::IMaterial, ConfigMaterials > materials
  Factory map for materials.
- static FactoryMap< Raytracer::Interfaces::IHittable, ConfigShapes > shapes
  Factory map for shapes.

7.18.1 Member Data Documentation

7.18.1.1 effects

```
Raytracer::Config::FactoryMap< Raytracer::Interfaces::IHittable, Raytracer::Config::Config↩
Effects > Raytracer::Config::Factory::effects  [static]
```

Factory map for effects.

This map is used to create effects based on their name. The key is the name of the effect and the value is a lambda function that creates the effect.

7.18.1.2 materials

```
Raytracer::Config::FactoryMap< Raytracer::Interfaces::IMaterial, Raytracer::Config::Config↩
Materials > Raytracer::Config::Factory::materials  [static]
```

Factory map for materials.

This map is used to create materials based on their name. The key is the name of the material and the value is a lambda function that creates the material.

7.18.1.3 shapes

```
Raytracer::Config::FactoryMap< Raytracer::Interfaces::IHittable, Raytracer::Config::Config↩
Shapes > Raytracer::Config::Factory::shapes  [static]
```

Factory map for shapes.

This map is used to create shapes based on their name. The key is the name of the shape and the value is a lambda function that creates the shape.

7.18.1.4 textures

```
Raytracer::Config::FactoryMap< Raytracer::Interfaces::ITexture, Raytracer::Config::Config↩
Textures > Raytracer::Config::Factory::textures  [static]
```

Factory map for textures.

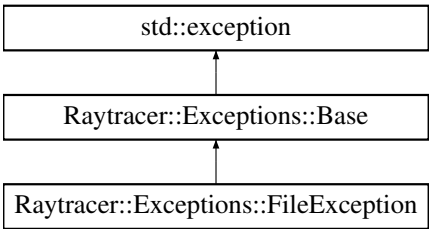
This map is used to create textures based on their name. The key is the name of the texture and the value is a lambda function that creates the texture.

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

- /Users/riosj1/Code/raytracer/include/config/Factory.hpp
- /Users/riosj1/Code/raytracer/sources/config/Factory.cpp

7.19 Raytracer::Exceptions::FileException Class Reference

Inheritance diagram for Raytracer::Exceptions::FileException:



Public Member Functions

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

Public Member Functions inherited from Raytracer::Exceptions::Base

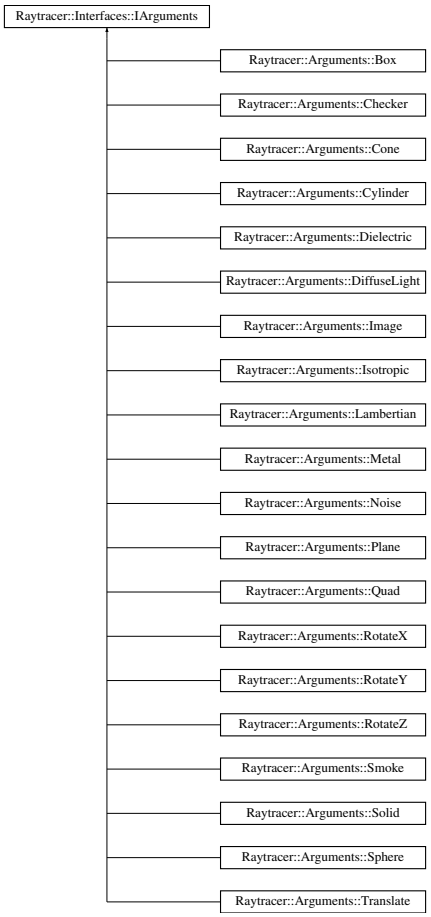
- **Base** (const std::string &message)
- virtual const char * **what** () const noexcept override

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

- /Users/riosj1/Code/raytracer/include/exceptions/File.hpp

7.20 Raytracer::Interfaces::IArguments Class Reference

Inheritance diagram for Raytracer::Interfaces::IArguments:



Public Member Functions

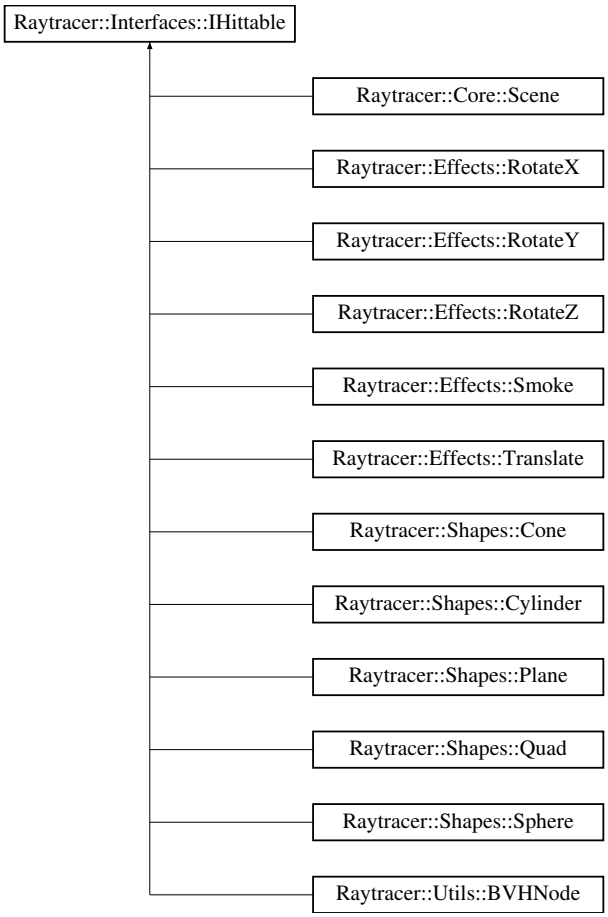
- virtual Arguments::ArgumentKind **kind** () const =0

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

- /Users/riosj1/Code/raytracer/include/interfaces/IArguments.hpp

7.21 Raytracer::Interfaces::IHittable Class Reference

Inheritance diagram for Raytracer::Interfaces::IHittable:



Public Member Functions

- virtual bool hit (const Core::Ray &ray, Utils::Interval interval, Core::Payload &payload) const =0
- virtual Utils::AxisAlignedBBox boundingBox () const =0

7.21.1 Member Function Documentation

7.21.1.1 boundingBox()

```
virtual Utils::AxisAlignedBBox Raytracer::Interfaces::IHittable::boundingBox ( ) const  [pure
virtual]
```

Implemented in Raytracer::Core::Scene, Raytracer::Effects::RotateX, Raytracer::Effects::RotateY, Raytracer::Effects::RotateZ, Raytracer::Effects::Smoke, Raytracer::Effects::Translate, Raytracer::Shapes::Cone, Raytracer::Shapes::Cylinder, Raytracer::Shapes::Plane, Raytracer::Shapes::Quad, Raytracer::Shapes::Sphere, and Raytracer::Utils::BVHNode.

7.21.1.2 hit()

```
virtual bool Raytracer::Interfaces::IHittable::hit (
            const Core::Ray & ray,
            Utils::Interval interval,
            Core::Payload & payload ) const  [pure virtual]
```

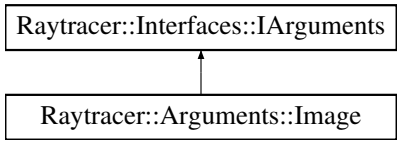
Implemented in Raytracer::Utils::BVHNode, Raytracer::Shapes::Sphere, Raytracer::Core::Scene, Raytracer::Effects::RotateX, Raytracer::Effects::RotateY, Raytracer::Effects::RotateZ, Raytracer::Effects::Smoke, Raytracer::Effects::Translate, Raytracer::Shapes::Cone, Raytracer::Shapes::Cylinder, Raytracer::Shapes::Plane, and Raytracer::Shapes::Quad.

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

- /Users/riosj1/Code/raytracer/include/interfaces/IHittable.hpp

7.22 Raytracer::Arguments::Image Class Reference

Inheritance diagram for Raytracer::Arguments::Image:



Public Member Functions

- **Image** (std::string filename)
- **GET_SET** (std::string, filename)
- **ARG_KIND** (ArgumentKind::ARG_IMAGE)

Public Member Functions inherited from Raytracer::Interfaces::IArguments

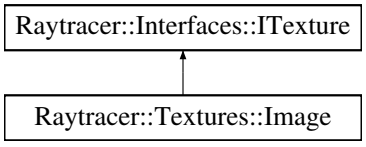
- virtual Arguments::ArgumentKind **kind** () const =0

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

- /Users/riosj1/Code/raytracer/include/arguments/Textures.hpp

7.23 Raytracer::Textures::Image Class Reference

Inheritance diagram for Raytracer::Textures::Image:



Public Member Functions

- Image (std::string filename)

  Construct a new Image object.
- Utils::Color value (double u, double v, const Utils::Point3 &point) const override

  Get the value of the image texture.

7.23.1 Constructor & Destructor Documentation

7.23.1.1 Image()

```
Raytracer::Textures::Image::Image (
            std::string filename )
```

Construct a new Image object.

This function constructs a new Image object with the given filename.

Parameters

filename	The filename of the image.

Returns

A new Image object.

7.23.2 Member Function Documentation

7.23.2.1 value()

```
Raytracer::Utils::Color Raytracer::Textures::Image::value (
            double u,
            double v,
            const Utils::Point3 & point ) const  [override], [virtual]
```

Get the value of the image texture.

This function returns the value of the image texture at the given UV coordinates and point.

Parameters

u	The U coordinate.
v	The V coordinate.
point	The point.

Returns

The value of the image texture.

Implements Raytracer::Interfaces::ITexture.

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

- /Users/riosj1/Code/raytracer/include/textures/Image.hpp
- /Users/riosj1/Code/raytracer/sources/textures/Image.cpp

7.24 Raytracer::Utils::ImageHelper Class Reference

Public Member Functions

- ImageHelper (const char *filename)

  Construct a new ImageHelper object.
- bool load (const std::string &filename)

  Load the image from the given filename.
- const unsigned char * pixelData (int x, int y) const

  Get the pixel data at the given coordinates.
- **GET_SET** (int, width)
- **GET_SET** (int, height)

7.24.1 Constructor & Destructor Documentation

7.24.1.1 ImageHelper()

```
Raytracer::Utils::ImageHelper::ImageHelper (
            const char * filename )
```

Construct a new ImageHelper object.

This function constructs a new ImageHelper object with the given filename. The ImageHelper object is used to load and read PPM images.

Parameters

filename	The filename of the image.

Returns

A new ImageHelper object.

7.24.2 Member Function Documentation

7.24.2.1 load()

```
bool Raytracer::Utils::ImageHelper::load (
            const std::string & filename )
```

Load the image from the given filename.

This function loads the image from the given filename. The image must be in PPM format (P6). The function returns true if the image was successfully loaded, and false otherwise.

Parameters

filename	The filename of the image.

Returns

True if the image was successfully loaded, and false otherwise.

7.24.2.2 pixelData()

```
const unsigned char * Raytracer::Utils::ImageHelper::pixelData (
            int x,
            int y ) const
```

Get the pixel data at the given coordinates.

This function returns the pixel data at the given coordinates. The function returns magenta if the coordinates are out of bounds.

Parameters

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

Returns

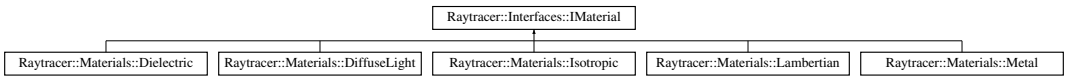
The pixel data at the given coordinates.

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

- /Users/riosj1/Code/raytracer/include/utils/ImageHelper.hpp
- /Users/riosj1/Code/raytracer/sources/utils/ImageHelper.cpp

7.25 Raytracer::Interfaces::IMaterial Class Reference

Inheritance diagram for Raytracer::Interfaces::IMaterial:



Public Member Functions

- virtual Utils::Color emitted (double u, double v, const Utils::Point3 &point) const =0
- virtual bool scatter (const Core::Ray &ray, const Core::Payload &payload, Utils::Color &attenuation, Core::Ray &scattered) const =0

7.25.1 Member Function Documentation

7.25.1.1 emitted()

```
virtual Utils::Color Raytracer::Interfaces::IMaterial::emitted (
            double u,
            double v,
            const Utils::Point3 & point ) const  [pure virtual]
```

Implemented in Raytracer::Materials::Dielectric, Raytracer::Materials::DiffuseLight, Raytracer::Materials::Isotropic, Raytracer::Materials::Lambertian, and Raytracer::Materials::Metal.

7.25.1.2 scatter()

```
virtual bool Raytracer::Interfaces::IMaterial::scatter (
            const Core::Ray & ray,
            const Core::Payload & payload,
            Utils::Color & attenuation,
            Core::Ray & scattered ) const  [pure virtual]
```

Implemented in Raytracer::Materials::Dielectric, Raytracer::Materials::DiffuseLight, Raytracer::Materials::Isotropic, Raytracer::Materials::Lambertian, and Raytracer::Materials::Metal.

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

- /Users/riosj1/Code/raytracer/include/interfaces/IMaterial.hpp

7.26 Raytracer::Utils::Interval Class Reference

Public Member Functions

- Interval (double min, double max)

  Construct a new Interval object.
- Interval (const Interval &a, const Interval &b)

  Construct a new Interval object.
- double size () const

  Get the minimum value of the interval.
- bool contains (double x) const

  Check if the interval contains the given value.
- bool surrounds (double x) const

  Check if the interval surrounds the given value.
- double clamp (double x) const

  Clamp the value to the interval.
- Interval expand (double x) const

  Expand the interval by the given value.

Static Public Attributes

- static const Interval Empty

  A constant empty interval.
- static const Interval Universe

  A constant universe interval.

7.26.1 Constructor & Destructor Documentation

7.26.1.1 Interval() [1/2]

```
Raytracer::Utils::Interval::Interval (
            double min,
            double max )
```

Construct a new Interval object.

This function constructs a new Interval object with the given minimum and maximum values.

Parameters

min	The minimum value.
max	The maximum value.

Returns

A new Interval object.

7.26.1.2 Interval() [2/2]

```
Raytracer::Utils::Interval::Interval (
            const Interval & a,
            const Interval & b )
```

Construct a new Interval object.

This function constructs a new Interval object by combining the given Intervals. The new Interval is the smallest Interval that contains both of the given Intervals.

Parameters

a	The first Interval.
b	The second Interval.

Returns

A new Interval object.

7.26.2 Member Function Documentation

7.26.2.1 clamp()

```
double Raytracer::Utils::Interval::clamp (
            double x ) const
```

Clamp the value to the interval.

This function clamps the value to the interval. The function returns the clamped value.

Parameters

x	The value to clamp.

Returns

The clamped value.

7.26.2.2 contains()

```
bool Raytracer::Utils::Interval::contains (
            double x ) const
```

Check if the interval contains the given value.

This function checks if the interval contains the given value. The function returns true if the interval contains the value, false otherwise.

Parameters

x	The value to check.

Returns

True if the interval contains the value, false otherwise.

7.26.2.3 expand()

```
Raytracer::Utils::Interval Raytracer::Utils::Interval::expand (
            double x ) const
```

Expand the interval by the given value.

This function expands the interval by the given value. The function returns the expanded interval.

Parameters

x	The value to expand the interval by.

Returns

The expanded interval.

7.26.2.4 size()

```
double Raytracer::Utils::Interval::size ( ) const
```

Get the minimum value of the interval.

This function returns the minimum value of the interval.

Returns

The minimum value of the interval.

7.26.2.5 surrounds()

```
bool Raytracer::Utils::Interval::surrounds (
            double x ) const
```

Check if the interval surrounds the given value.

This function checks if the interval surrounds the given value. The function returns true if the interval surrounds the value, false otherwise.

Parameters

x	The value to check.

Returns

True if the interval surrounds the value, false otherwise.

7.26.3 Member Data Documentation

7.26.3.1 Empty

```
const Raytracer::Utils::Interval Raytracer::Utils::Interval::Empty  [static]
```

Initial value:

```
=
    Interval(+std::numeric_limits<double>::infinity(),
        -std::numeric_limits<double>::infinity())
```

A constant empty interval.

This constant represents an empty interval.

7.26.3.2 Universe

```
const Raytracer::Utils::Interval Raytracer::Utils::Interval::Universe  [static]
```

Initial value:

```
=
    Interval(-std::numeric_limits<double>::infinity(),
        +std::numeric_limits<double>::infinity())
```

A constant universe interval.

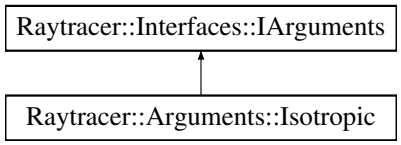
This constant represents the universe interval.

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

- /Users/riosj1/Code/raytracer/include/utils/Interval.hpp
- /Users/riosj1/Code/raytracer/sources/utils/Interval.cpp

7.27 Raytracer::Arguments::Isotropic Class Reference

Inheritance diagram for Raytracer::Arguments::Isotropic:



Public Member Functions

- **Isotropic** (Utils::Color color)
- **Isotropic** (std::shared_ptr< Interfaces::ITexture > texture)
- **GET_SET** (Utils::Color, color)
- **GET_SET** (std::shared_ptr< Interfaces::ITexture >, texture)
- **ARG_KIND** (_kind)

Public Member Functions inherited from Raytracer::Interfaces::IArguments

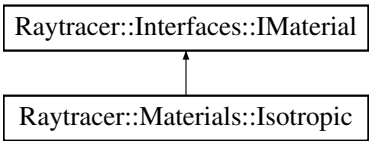
- virtual Arguments::ArgumentKind **kind** () const =0

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

- /Users/riosj1/Code/raytracer/include/arguments/Materials.hpp

7.28 Raytracer::Materials::Isotropic Class Reference

Inheritance diagram for Raytracer::Materials::Isotropic:



Public Member Functions

- Isotropic (std::shared_ptr< Interfaces::ITexture > texture)

  Construct a new Isotropic object.
- Isotropic (const Utils::Color &color)

  Construct a new Isotropic object.
- bool scatter (const Core::Ray &ray, const Core::Payload &payload, Utils::Color &attenuation, Core::Ray &scattered) const override

  Scatter the ray with the isotropic material.
- Utils::Color emitted (double u, double v, const Utils::Point3 &point) const override

  Emitted light of the isotropic material.

7.28.1 Constructor & Destructor Documentation

7.28.1.1 Isotropic() [1/2]

```
Raytracer::Materials::Isotropic::Isotropic (
            std::shared_ptr< Interfaces::ITexture > texture )
```

Construct a new Isotropic object.

This function constructs a new Isotropic object with the given texture.

Parameters

texture	The texture of the isotropic material.

Returns

A new Isotropic object.

7.28.1.2 Isotropic() [2/2]

```
Raytracer::Materials::Isotropic::Isotropic (
            const Utils::Color & color )
```

Construct a new Isotropic object.

This function constructs a new Isotropic object with the given color.

Parameters

color	The color of the isotropic material.

Returns

A new Isotropic object.

7.28.2 Member Function Documentation

7.28.2.1 emitted()

```
Raytracer::Utils::Color Raytracer::Materials::Isotropic::emitted (
            double u,
            double v,
            const Utils::Point3 & point ) const  [override], [virtual]
```

Emitted light of the isotropic material.

This function returns the emitted light of the isotropic material. The function returns the emitted light of the material at the given point.

Parameters

u	The u texture coordinate.
v	The v texture coordinate.
point	The point to get the emitted light from.

Returns

The emitted light of the isotropic material.

Implements Raytracer::Interfaces::IMaterial.

7.28.2.2 scatter()

```
bool Raytracer::Materials::Isotropic::scatter (
            const Core::Ray & ray,
            const Core::Payload & payload,
            Utils::Color & attenuation,
            Core::Ray & scattered ) const  [override], [virtual]
```

Scatter the ray with the isotropic material.

This function scatters the ray with the isotropic material. The function returns true if the ray is scattered. The function returns false if the ray is not scattered. The function updates the attenuation and scattered ray.

Parameters

ray	The ray to scatter.
payload	The payload of the ray.
attenuation	The attenuation of the ray.
scattered	The scattered ray.

Returns

true if the ray is scattered, false otherwise.

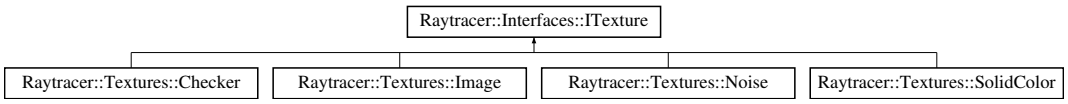
Implements Raytracer::Interfaces::IMaterial.

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

- /Users/riosj1/Code/raytracer/include/materials/Isotropic.hpp
- /Users/riosj1/Code/raytracer/sources/materials/Isotropic.cpp

7.29 Raytracer::Interfaces::ITexture Class Reference

Inheritance diagram for Raytracer::Interfaces::ITexture:



Public Member Functions

- virtual Utils::Color value (double u, double v, const Utils::Point3 &point) const =0

7.29.1 Member Function Documentation

7.29.1.1 value()

```
virtual Utils::Color Raytracer::Interfaces::ITexture::value (
            double u,
            double v,
            const Utils::Point3 & point ) const  [pure virtual]
```

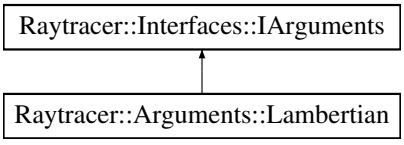
Implemented in Raytracer::Textures::Checker, Raytracer::Textures::Image, Raytracer::Textures::Noise, and Raytracer::Textures::SolidColor.

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

- /Users/riosj1/Code/raytracer/include/interfaces/ITexture.hpp

7.30 Raytracer::Arguments::Lambertian Class Reference

Inheritance diagram for Raytracer::Arguments::Lambertian:



Public Member Functions

- **Lambertian** (Utils::Vec3 color)
- **Lambertian** (double r, double g, double b)
- **Lambertian** (std::shared_ptr< Interfaces::ITexture > texture)
- **GET_SET** (Utils::Vec3, color)
- **GET_SET** (std::shared_ptr< Interfaces::ITexture >, texture)
- **ARG_KIND** (_kind)

Public Member Functions inherited from Raytracer::Interfaces::IArguments

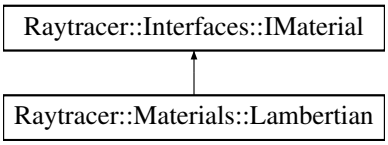
- virtual Arguments::ArgumentKind **kind** () const =0

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

- /Users/riosj1/Code/raytracer/include/arguments/Materials.hpp

7.31 Raytracer::Materials::Lambertian Class Reference

Inheritance diagram for Raytracer::Materials::Lambertian:



Public Member Functions

- Lambertian (const Utils::Color &albedo)

  Construct a new Lambertian object.
- Lambertian (std::shared_ptr< Interfaces::ITexture > texture)

  Construct a new Lambertian object.
- bool scatter (const Core::Ray &ray, const Core::Payload &payload, Utils::Color &attenuation, Core::Ray &scattered) const override

  Scatter the ray with the Lambertian material.
- Utils::Color emitted (double u, double v, const Utils::Point3 &point) const override

  Emitted light of the Lambertian material.

7.31.1 Constructor & Destructor Documentation

7.31.1.1 Lambertian() [1/2]

```
Raytracer::Materials::Lambertian::Lambertian (
            const Utils::Color & albedo )
```

Construct a new Lambertian object.

This function constructs a new Lambertian object with the given albedo. The Lambertian material scatters light with the given albedo.

Parameters

albedo	The albedo of the Lambertian material.

Returns

A new Lambertian object.

7.31.1.2 Lambertian() [2/2]

```
Raytracer::Materials::Lambertian::Lambertian (
            std::shared_ptr< Interfaces::ITexture > texture )
```

Construct a new Lambertian object.

This function constructs a new Lambertian object with the given texture. The Lambertian material scatters light with the given texture.

Parameters

texture	The texture of the Lambertian material.

Returns

A new Lambertian object.

7.31.2 Member Function Documentation

7.31.2.1 emitted()

```
Raytracer::Utils::Color Raytracer::Materials::Lambertian::emitted (
            double u,
            double v,
            const Utils::Point3 & point ) const  [override], [virtual]
```

Emitted light of the Lambertian material.

This function returns the emitted light of the Lambertian material. The function returns the emitted light of the material at the given point.

Parameters

u	The u coordinate of the texture.
v	The v coordinate of the texture.
point	The point to get the emitted light from.

Returns

The emitted light of the material.

Implements Raytracer::Interfaces::IMaterial.

7.31.2.2 scatter()

```
bool Raytracer::Materials::Lambertian::scatter (
            const Core::Ray & ray,
            const Core::Payload & payload,
            Utils::Color & attenuation,
            Core::Ray & scattered ) const  [override], [virtual]
```

Scatter the ray with the Lambertian material.

This function scatters the ray with the Lambertian material. The function returns true if the ray is scattered. The function returns false if the ray is not scattered. The function updates the attenuation and scattered ray.

Parameters

ray	The ray to scatter.
payload	The payload of the ray.
attenuation	The attenuation of the ray.
scattered	The scattered ray.

Returns

true if the ray is scattered, false otherwise.

Implements Raytracer::Interfaces::IMaterial.

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

- /Users/riosj1/Code/raytracer/include/materials/Lambertian.hpp
- /Users/riosj1/Code/raytracer/sources/materials/Lambertian.cpp

7.32 Raytracer::Config::Manager Class Reference

Public Member Functions

- Manager ()
  Construct a new Manager:: Manager object.
- bool parse (std::string path)
  Parse the configuration file.
- void bootstrap ()
  Bootstrap the configuration.
- void render (bool fast)
  Render the scene.
- **GET_SET** (Raytracer::Core::Scene, world)
- **GET_SET** (Raytracer::Core::Camera, camera)

7.32.1 Constructor & Destructor Documentation

7.32.1.1 Manager()

```
Raytracer::Config::Manager::Manager ( )
```

Construct a new Manager:: Manager object.

Initialize the argument map with the available arguments and their corresponding functions. Initialize the camera map with the available camera arguments and their corresponding functions.

7.32.2 Member Function Documentation

7.32.2.1 bootstrap()

```
void Raytracer::Config::Manager::bootstrap ( )
```

Bootstrap the configuration.

Bootstrap the configuration by adding the shapes to the world.

Returns

void

7.32.2.2 parse()

```
bool Raytracer::Config::Manager::parse (
            std::string path )
```

Parse the configuration file.

Parse the configuration file and load the scene, textures, materials, shapes and effects. If an error occurs while parsing the file, an exception is thrown.

Parameters

path	Path to the configuration file

Exceptions

Exceptions::ParseException	if an error occurs while parsing the file
Exceptions::CyclicException	if a cyclic dependency is detected
Exceptions::MissingException	if a path is not found
Exceptions::ParseException	if a path is not of the correct type
Exceptions::MissingException	if an argument is not found

Returns

void

7.32.2.3 render()

```
void Raytracer::Config::Manager::render (
            bool fast )
```

Render the scene.

Render the scene using the camera and the world. If the fast flag is set, the rendering is done in fast mode, which reduces the image width, the samples per pixel and the maximum depth.

Parameters

fast	Fast rendering mode

Returns

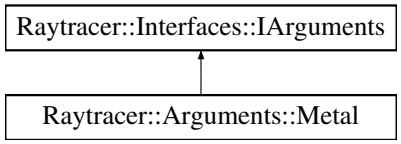
void

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

- /Users/riosj1/Code/raytracer/include/config/Manager.hpp
- /Users/riosj1/Code/raytracer/sources/config/Manager.cpp

7.33 Raytracer::Arguments::Metal Class Reference

Inheritance diagram for Raytracer::Arguments::Metal:



Public Member Functions

- **Metal** (Utils::Vec3 color, double fuzz)
- **GET_SET** (Utils::Vec3, color)
- **GET_SET** (double, fuzz)
- **ARG_KIND** (ArgumentKind::ARG_METAL)

Public Member Functions inherited from Raytracer::Interfaces::IArguments

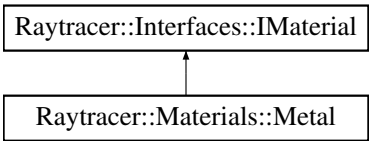
- virtual Arguments::ArgumentKind **kind** () const =0

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

- /Users/riosj1/Code/raytracer/include/arguments/Materials.hpp

7.34 Raytracer::Materials::Metal Class Reference

Inheritance diagram for Raytracer::Materials::Metal:



Public Member Functions

- Metal (const Utils::Color &albedo, double fuzz)

  Construct a new Metal object.
- bool scatter (const Core::Ray &ray, const Core::Payload &payload, Utils::Color &attenuation, Core::Ray &scattered) const override

  Scatter the ray with the Metal material.
- Utils::Color emitted (double u, double v, const Utils::Point3 &point) const override

  Emitted light of the Metal material.

7.34.1 Constructor & Destructor Documentation

7.34.1.1 Metal()

```
Raytracer::Materials::Metal::Metal (
            const Utils::Color & albedo,
            double fuzz )
```

Construct a new Metal object.

This function constructs a new Metal object with the given albedo and fuzz. The Metal material scatters light with the given albedo and fuzz.

Parameters

albedo	The albedo of the Metal material.
fuzz	The fuzz of the Metal material.

Returns

A new Metal object.

7.34.2 Member Function Documentation

7.34.2.1 emitted()

```
Raytracer::Utils::Color Raytracer::Materials::Metal::emitted (
            double u,
```

```
            double v,
            const Utils::Point3 & point ) const  [override], [virtual]
```

Emitted light of the Metal material.

This function returns the emitted light of the Metal material. The function returns the emitted light of the material at the given point.

Parameters

u	The u coordinate of the texture.
v	The v coordinate of the texture.
point	The point of intersection.

Returns

The emitted light of the material.

Implements Raytracer::Interfaces::IMaterial.

7.34.2.2 scatter()

```
bool Raytracer::Materials::Metal::scatter (
            const Core::Ray & ray,
            const Core::Payload & payload,
            Utils::Color & attenuation,
            Core::Ray & scattered ) const  [override], [virtual]
```

Scatter the ray with the Metal material.

This function scatters the ray with the Metal material. The function returns true if the ray is scattered. The function returns false if the ray is not scattered. The function updates the attenuation and scattered ray.

Parameters

ray	The ray to scatter.
payload	The payload of the ray.
attenuation	The attenuation of the ray.
scattered	The scattered ray.

Returns

true if the ray is scattered, false otherwise.

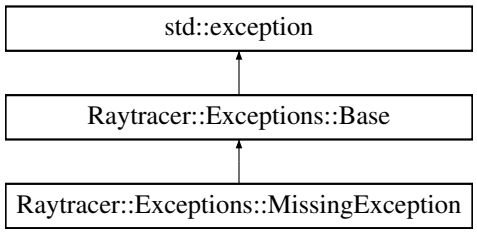
Implements Raytracer::Interfaces::IMaterial.

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

- /Users/riosj1/Code/raytracer/include/materials/Metal.hpp
- /Users/riosj1/Code/raytracer/sources/materials/Metal.cpp

7.35 Raytracer::Exceptions::MissingException Class Reference

Inheritance diagram for Raytracer::Exceptions::MissingException:



Public Member Functions

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

Public Member Functions inherited from Raytracer::Exceptions::Base

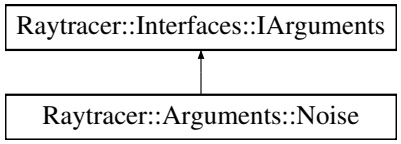
- **Base** (const std::string &message)
- virtual const char * **what** () const noexcept override

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

- /Users/riosj1/Code/raytracer/include/exceptions/Missing.hpp

7.36 Raytracer::Arguments::Noise Class Reference

Inheritance diagram for Raytracer::Arguments::Noise:



Public Member Functions

- **Noise** (double scale)
- **GET_SET** (double, scale)
- **ARG_KIND** (ArgumentKind::ARG_NOISE)

Public Member Functions inherited from Raytracer::Interfaces::IArguments

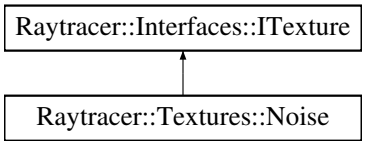
- virtual Arguments::ArgumentKind **kind** () const =0

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

- /Users/riosj1/Code/raytracer/include/arguments/Textures.hpp

7.37 Raytracer::Textures::Noise Class Reference

Inheritance diagram for Raytracer::Textures::Noise:



Public Member Functions

- Noise (double scale)

  Construct a new Noise object.
- Utils::Color value (double u, double v, const Utils::Point3 &point) const override

  Get the value of the noise texture.

7.37.1 Constructor & Destructor Documentation

7.37.1.1 Noise()

```
Raytracer::Textures::Noise::Noise (
            double scale )
```

Construct a new Noise object.

This function constructs a new Noise object with the given scale. The Noise texture is a texture that generates Perlin noise.

Parameters

scale	The scale of the noise texture.

Returns

A new Noise object.

7.37.2 Member Function Documentation

7.37.2.1 value()

```
Raytracer::Utils::Color Raytracer::Textures::Noise::value (
            double u,
            double v,
            const Utils::Point3 & point ) const  [override], [virtual]
```

Get the value of the noise texture.

This function returns the value of the noise texture at the given UV coordinates and point.

Parameters

u	The U coordinate.
v	The V coordinate.
point	The point.

Returns

The value of the noise texture.

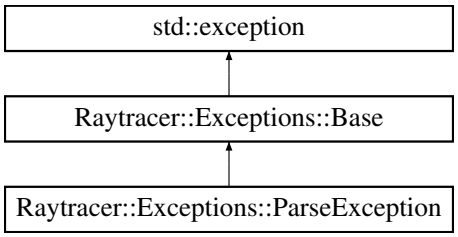
Implements Raytracer::Interfaces::ITexture.

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

- /Users/riosj1/Code/raytracer/include/textures/Noise.hpp
- /Users/riosj1/Code/raytracer/sources/textures/Noise.cpp

7.38 Raytracer::Exceptions::ParseException Class Reference

Inheritance diagram for Raytracer::Exceptions::ParseException:



Public Member Functions

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

Public Member Functions inherited from Raytracer::Exceptions::Base

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

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

- /Users/riosj1/Code/raytracer/include/exceptions/Parse.hpp

7.39 Raytracer::Core::Payload Class Reference

Public Member Functions

- void setFaceNormal (const Core::Ray &ray, const Utils::Vec3 &outwardNormal)

  Set the face normal based on the ray and the outward normal.

7.39.1 Member Function Documentation

7.39.1.1 setFaceNormal()

```
void Raytracer::Core::Payload::setFaceNormal (
            const Core::Ray & ray,
            const Utils::Vec3 & outwardNormal )
```

Set the face normal based on the ray and the outward normal.

This function sets the face normal based on the ray and the outward normal. If the ray is outside the object, the face normal is the outward normal. If the ray is inside the object, the face normal is the negative of the outward normal.

Parameters

ray	The ray to set the face normal from.
outwardNormal	The outward normal to set the face normal from.

Returns

void

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

- /Users/riosj1/Code/raytracer/include/core/Payload.hpp
- /Users/riosj1/Code/raytracer/sources/core/Payload.cpp

7.40 Raytracer::Utils::Perlin Class Reference

Public Member Functions

- Perlin ()

  Construct a new Perlin object.
- ~Perlin ()

  Destroy the Perlin object.
- double noise (const Utils::Point3 &point) const

  Get the noise value at the given point.
- double turbulence (const Utils::Point3 &point, int depth=7) const

  Get the turbulence value at the given point.

Static Public Member Functions

- static int * perlinGeneratePerm ()

  Generate the permutation table.
- static void permute (int *perm, int n)

  Permute the permutation table.
- static double perlinInterp (const Utils::Vec3 c[2][2][2], double u, double v, double w)

  Interpolate the noise value.

7.40.1 Constructor & Destructor Documentation

7.40.1.1 Perlin()

```
Raytracer::Utils::Perlin::Perlin ( )
```

Construct a new Perlin object.

This function constructs a new Perlin object. The Perlin object is a Perlin noise generator. The Perlin noise is a type of gradient noise.

Returns

A new Perlin object.

7.40.1.2 ~Perlin()

```
Raytracer::Utils::Perlin::~Perlin ( )
```

Destroy the Perlin object.

This function destroys the Perlin object.

7.40.2 Member Function Documentation

7.40.2.1 noise()

```
double Raytracer::Utils::Perlin::noise (
            const Utils::Point3 & point ) const
```

Get the noise value at the given point.

This function returns the noise value at the given point.

Parameters

point	The point.

Returns

The noise value.

7.40.2.2 perlinGeneratePerm()

```
int * Raytracer::Utils::Perlin::perlinGeneratePerm ( )  [static]
```

Generate the permutation table.

This function generates the permutation table.

Returns

The permutation table.

7.40.2.3 perlinInterp()

```
double Raytracer::Utils::Perlin::perlinInterp (
            const Utils::Vec3 c[2][2][2],
            double u,
            double v,
            double w )  [static]
```

Interpolate the noise value.

This function interpolates the noise value.

Parameters

c	The noise value.
u	The U coordinate.
v	The V coordinate.
w	The W coordinate.

Returns

The interpolated noise value.

7.40.2.4 permute()

```
void Raytracer::Utils::Perlin::permute (
            int * perm,
            int n )  [static]
```

Permute the permutation table.

This function permutes the permutation table.

Parameters

perm	The permutation table.
n	The size of the permutation table.

7.40.2.5 turbulence()

```
double Raytracer::Utils::Perlin::turbulence (
            const Utils::Point3 & point,
            int depth = 7 ) const
```

Get the turbulence value at the given point.

This function returns the turbulence value at the given point.

Parameters

point	The point.
depth	The depth.

Returns

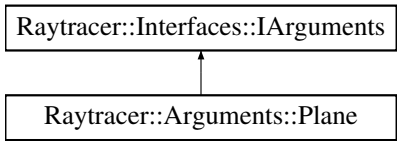
The turbulence value.

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

- /Users/riosj1/Code/raytracer/include/utils/Perlin.hpp
- /Users/riosj1/Code/raytracer/sources/utils/Perlin.cpp

7.41 Raytracer::Arguments::Plane Class Reference

Inheritance diagram for Raytracer::Arguments::Plane:



Public Member Functions

- **Plane** (Utils::Point3 &point, Utils::Vec3 &normal, std::shared_ptr< Interfaces::IMaterial > material)
- **GET_SET** (Utils::Point3, point)
- **GET_SET** (Utils::Vec3, normal)
- **GET_SET** (std::shared_ptr< Interfaces::IMaterial >, material)
- **ARG_KIND** (ArgumentKind::ARG_PLANE)

Public Member Functions inherited from Raytracer::Interfaces::IArguments

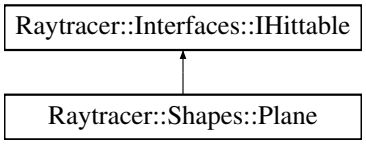
- virtual Arguments::ArgumentKind **kind** () const =0

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

- /Users/riosj1/Code/raytracer/include/arguments/Shapes.hpp

7.42 Raytracer::Shapes::Plane Class Reference

Inheritance diagram for Raytracer::Shapes::Plane:



Public Member Functions

- Plane (const Utils::Point3 &point, const Utils::Vec3 &normal, std::shared_ptr< Interfaces::IMaterial > material)

  Construct a new Plane object.
- bool hit (const Core::Ray &ray, Utils::Interval interval, Core::Payload &payload) const override

  Check if the ray hits the plane.
- Utils::AxisAlignedBBox boundingBox () const override

  Get the bounding box of the plane.

7.42.1 Constructor & Destructor Documentation

7.42.1.1 Plane()

```
Raytracer::Shapes::Plane::Plane (
            const Utils::Point3 & point,
            const Utils::Vec3 & normal,
            std::shared_ptr< Interfaces::IMaterial > material )
```

Construct a new Plane object.

This function constructs a new Plane object with the given point, normal, and material. The plane is centered at the given point with the given normal and material.

Parameters

point	The point of the plane.
normal	The normal of the plane.
material	The material of the plane.

Returns

A new Plane object.

7.42.2 Member Function Documentation

7.42.2.1 boundingBox()

```
Raytracer::Utils::AxisAlignedBBox Raytracer::Shapes::Plane::boundingBox ( ) const  [override],
[virtual]
```

Get the bounding box of the plane.

This function returns the bounding box of the plane.

Returns

The bounding box of the plane.

Implements Raytracer::Interfaces::IHittable.

7.42.2.2 hit()

```
bool Raytracer::Shapes::Plane::hit (
            const Core::Ray & ray,
            Utils::Interval interval,
            Core::Payload & payload ) const  [override], [virtual]
```

Check if the ray hits the plane.

This function checks if the ray hits the plane. The function returns true if the ray hits the plane. The function returns false if the ray does not hit the plane. The function updates the payload with the hit information.

Parameters

ray	The ray to check for hits.
interval	The interval to check for hits.
payload	The payload to update with the hit information.

Returns

true if the ray hits the plane, false otherwise.

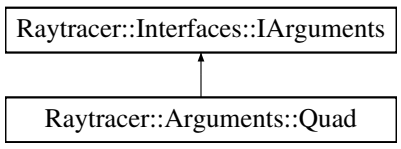
Implements Raytracer::Interfaces::IHittable.

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

- /Users/riosj1/Code/raytracer/include/shapes/Plane.hpp
- /Users/riosj1/Code/raytracer/sources/shapes/Plane.cpp

7.43 Raytracer::Arguments::Quad Class Reference

Inheritance diagram for Raytracer::Arguments::Quad:



Public Member Functions

- **Quad** (Utils::Point3 &Q, Utils::Point3 &u, Utils::Point3 &v, std::shared_ptr< Interfaces::IMaterial > material)
- **GET_SET** (Utils::Point3, Q)
- **GET_SET** (Utils::Point3, u)
- **GET_SET** (Utils::Point3, v)
- **GET_SET** (std::shared_ptr< Interfaces::IMaterial >, material)
- **ARG_KIND** (ArgumentKind::ARG_QUAD)

Public Member Functions inherited from Raytracer::Interfaces::IArguments

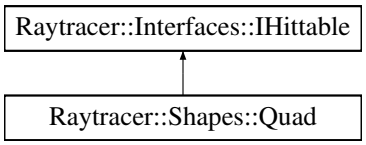
- virtual Arguments::ArgumentKind **kind** () const =0

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

- /Users/riosj1/Code/raytracer/include/arguments/Shapes.hpp

7.44 Raytracer::Shapes::Quad Class Reference

Inheritance diagram for Raytracer::Shapes::Quad:



Public Member Functions

- Quad (const Utils::Point3 &Q, const Utils::Vec3 &u, const Utils::Vec3 &v, std::shared_ptr< Interfaces::IMaterial > material)

  Construct a new Quad object.
- bool hit (const Core::Ray &ray, Utils::Interval interval, Core::Payload &payload) const override

  Check if the ray hits the quad.
- Utils::AxisAlignedBBox boundingBox () const override

  Get the bounding box of the quad.
- virtual void setBBox ()

  Set the bounding box of the quad.
- virtual bool isInterior (double a, double b, Core::Payload &payload) const

  Check if the point is inside the quad.

7.44.1 Constructor & Destructor Documentation

7.44.1.1 Quad()

```
Raytracer::Shapes::Quad::Quad (
            const Utils::Point3 & Q,
            const Utils::Vec3 & u,
            const Utils::Vec3 & v,
            std::shared_ptr< Interfaces::IMaterial > material )
```

Construct a new Quad object.

This function constructs a new Quad object with the given point, u, v, and material. The quad is centered at the given point with the given u, v, and material.

Parameters

Q	The point of the quad.
u	The u vector of the quad.
v	The v vector of the quad.
material	The material of the quad.

Returns

A new Quad object.

7.44.2 Member Function Documentation

7.44.2.1 boundingBox()

```
Raytracer::Utils::AxisAlignedBBox Raytracer::Shapes::Quad::boundingBox ( ) const  [override],
[virtual]
```

Get the bounding box of the quad.

This function returns the bounding box of the quad.

Returns

The bounding box of the quad.

Implements Raytracer::Interfaces::IHittable.

7.44.2.2 hit()

```
bool Raytracer::Shapes::Quad::hit (
            const Core::Ray & ray,
            Utils::Interval interval,
            Core::Payload & payload ) const  [override], [virtual]
```

Check if the ray hits the quad.

This function checks if the ray hits the quad. The function returns true if the ray hits the quad. The function returns false if the ray does not hit the quad. The function updates the payload with the hit information.

Parameters

ray	The ray to check for hits.
interval	The interval to check for hits.
payload	The payload to update with the hit information.

Returns

true if the ray hits the quad, false otherwise.

Implements Raytracer::Interfaces::IHittable.

7.44.2.3 isInterior()

```
bool Raytracer::Shapes::Quad::isInterior (
            double a,
            double b,
            Core::Payload & payload ) const  [virtual]
```

Check if the point is inside the quad.

This function checks if the point is inside the quad. The function returns true if the point is inside the quad. The function returns false if the point is not inside the quad. The function updates the payload with the u and v values of the point.

Parameters

a	The alpha value of the point.
b	The beta value of the point.
payload	The payload to update with the u and v values.

Returns

True if the point is inside the quad, false otherwise.

7.44.2.4 setBBox()

```
void Raytracer::Shapes::Quad::setBBox ( )  [virtual]
```

Set the bounding box of the quad.

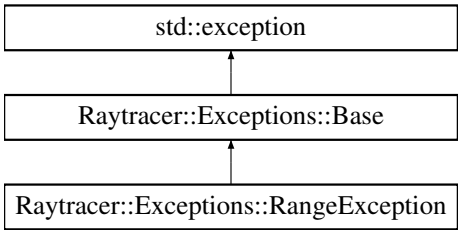
This function sets the bounding box of the quad.

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

- /Users/riosj1/Code/raytracer/include/shapes/Quad.hpp
- /Users/riosj1/Code/raytracer/sources/shapes/Quad.cpp

7.45 Raytracer::Exceptions::RangeException Class Reference

Inheritance diagram for Raytracer::Exceptions::RangeException:



Public Member Functions

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

Public Member Functions inherited from Raytracer::Exceptions::Base

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

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

- /Users/riosj1/Code/raytracer/include/exceptions/Range.hpp

7.46 Raytracer::Core::Ray Class Reference

Public Member Functions

- Ray (const Utils::Point3 &origin, const Utils::Vec3 &direction, double time=0.0)

  Construct a new Ray object.
- Utils::Point3 at (double t) const

  Get the origin of the ray.

7.46.1 Constructor & Destructor Documentation

7.46.1.1 Ray()

```
Raytracer::Core::Ray::Ray (
            const Utils::Point3 & origin,
            const Utils::Vec3 & direction,
            double time = 0.0 )
```

Construct a new Ray object.

This function constructs a new Ray object with the given origin, direction, and time.

Parameters

origin	The origin of the ray.
direction	The direction of the ray.
time	The time of the ray.

Returns

A new Ray object.

7.46.2 Member Function Documentation

7.46.2.1 at()

```
Raytracer::Utils::Point3 Raytracer::Core::Ray::at (
            double t ) const
```

Get the origin of the ray.

This function returns the origin of the ray.

Returns

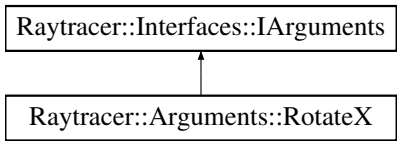
The origin of the ray.

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

- /Users/riosj1/Code/raytracer/include/core/Ray.hpp
- /Users/riosj1/Code/raytracer/sources/core/Ray.cpp

7.47 Raytracer::Arguments::RotateX Class Reference

Inheritance diagram for Raytracer::Arguments::RotateX:



Public Member Functions

- **RotateX** (std::shared_ptr< Interfaces::IHittable > object, double angle)
- **GET_SET** (double, angle)
- **GET_SET** (std::shared_ptr< Interfaces::IHittable >, object)
- **ARG_KIND** (ArgumentKind::ARG_ROTATE_X)

Public Member Functions inherited from Raytracer::Interfaces::IArguments

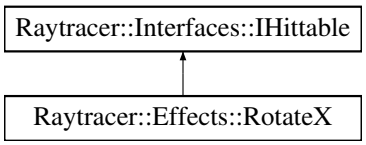
- virtual Arguments::ArgumentKind **kind** () const =0

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

- /Users/riosj1/Code/raytracer/include/arguments/Effects.hpp

7.48 Raytracer::Effects::RotateX Class Reference

Inheritance diagram for Raytracer::Effects::RotateX:



Public Member Functions

- RotateX (std::shared_ptr< Interfaces::IHittable > object, double angle)

  Construct a new RotateX object.
- bool hit (const Core::Ray &ray, Utils::Interval interval, Core::Payload &payload) const override

  Check if the ray hits the rotated object.
- Utils::AxisAlignedBBox boundingBox () const override

  Get the bounding box of the rotated object.

7.48.1 Constructor & Destructor Documentation

7.48.1.1 RotateX()

```
Raytracer::Effects::RotateX::RotateX (
            std::shared_ptr< Interfaces::IHittable > object,
            double angle )
```

Construct a new RotateX object.

This function constructs a new RotateX object with the given object and angle. The object is rotated around the x-axis by the given angle. The bounding box of the object is updated with the rotated bounding box.

Parameters

object	The object to rotate.
angle	The angle to rotate the object by.

Returns

A new RotateX object.

7.48.2 Member Function Documentation

7.48.2.1 boundingBox()

```
Raytracer::Utils::AxisAlignedBBox Raytracer::Effects::RotateX::boundingBox ( ) const  [override],
[virtual]
```

Get the bounding box of the rotated object.

This function returns the bounding box of the rotated object.

Returns

The bounding box of the rotated object.

Implements Raytracer::Interfaces::IHittable.

7.48.2.2 hit()

```
bool Raytracer::Effects::RotateX::hit (
            const Core::Ray & ray,
            Utils::Interval interval,
            Core::Payload & payload ) const  [override], [virtual]
```

Check if the ray hits the rotated object.

This function checks if the ray hits the rotated object. The function returns true if the ray hits the rotated object. The function returns false if the ray does not hit the rotated object. The function updates the payload with the hit information.

Parameters

ray	The ray to check for hits.
interval	The interval to check for hits.
payload	The payload to update with the hit information.

Returns

True if the ray hits the rotated object, false otherwise.

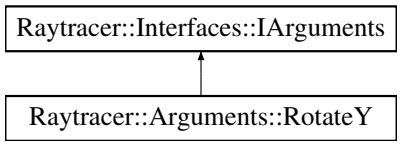
Implements Raytracer::Interfaces::IHittable.

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

- /Users/riosj1/Code/raytracer/include/effects/RotateX.hpp
- /Users/riosj1/Code/raytracer/sources/effects/RotateX.cpp

7.49 Raytracer::Arguments::RotateY Class Reference

Inheritance diagram for Raytracer::Arguments::RotateY:



Public Member Functions

- **RotateY** (std::shared_ptr< Interfaces::IHittable > object, double angle)
- **GET_SET** (double, angle)
- **GET_SET** (std::shared_ptr< Interfaces::IHittable >, object)
- **ARG_KIND** (ArgumentKind::ARG_ROTATE_Y)

Public Member Functions inherited from Raytracer::Interfaces::IArguments

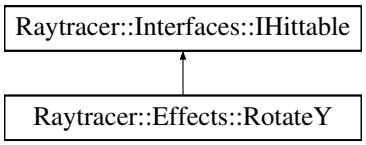
- virtual Arguments::ArgumentKind **kind** () const =0

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

- /Users/riosj1/Code/raytracer/include/arguments/Effects.hpp

7.50 Raytracer::Effects::RotateY Class Reference

Inheritance diagram for Raytracer::Effects::RotateY:



Public Member Functions

- RotateY (std::shared_ptr< Interfaces::IHittable > object, double angle)

  Construct a new RotateY object.
- bool hit (const Core::Ray &ray, Utils::Interval interval, Core::Payload &payload) const override

  Check if the ray hits the rotated object.
- Utils::AxisAlignedBBox boundingBox () const override

  Get the bounding box of the rotated object.

7.50.1 Constructor & Destructor Documentation

7.50.1.1 RotateY()

```
Raytracer::Effects::RotateY::RotateY (
            std::shared_ptr< Interfaces::IHittable > object,
            double angle )
```

Construct a new RotateY object.

This function constructs a new RotateY object with the given object and angle. The object is rotated around the y-axis by the given angle. The bounding box of the object is updated with the rotated bounding box.

Parameters

object	The object to rotate.
angle	The angle to rotate the object by.

Returns

A new RotateY object.

7.50.2 Member Function Documentation

7.50.2.1 boundingBox()

```
Raytracer::Utils::AxisAlignedBBox Raytracer::Effects::RotateY::boundingBox ( ) const  [override], [virtual]
```

Get the bounding box of the rotated object.

This function returns the bounding box of the rotated object.

Returns

The bounding box of the rotated object.

Implements Raytracer::Interfaces::IHittable.

7.50.2.2 hit()

```
bool Raytracer::Effects::RotateY::hit (
            const Core::Ray & ray,
            Utils::Interval interval,
            Core::Payload & payload ) const  [override], [virtual]
```

Check if the ray hits the rotated object.

This function checks if the ray hits the rotated object. The function returns true if the ray hits the rotated object. The function returns false if the ray does not hit the rotated object. The function updates the payload with the hit information.

Parameters

ray	The ray to check for hits.
interval	The interval to check for hits.
payload	The payload to update with the hit information.

Returns

True if the ray hits the rotated object, false otherwise.

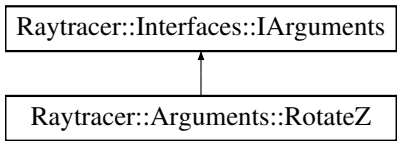
Implements Raytracer::Interfaces::IHittable.

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

- /Users/riosj1/Code/raytracer/include/effects/RotateY.hpp
- /Users/riosj1/Code/raytracer/sources/effects/RotateY.cpp

7.51 Raytracer::Arguments::RotateZ Class Reference

Inheritance diagram for Raytracer::Arguments::RotateZ:



Public Member Functions

- **RotateZ** (std::shared_ptr< Interfaces::IHittable > object, double angle)
- **GET_SET** (double, angle)
- **GET_SET** (std::shared_ptr< Interfaces::IHittable >, object)
- **ARG_KIND** (ArgumentKind::ARG_ROTATE_Z)

Public Member Functions inherited from Raytracer::Interfaces::IArguments

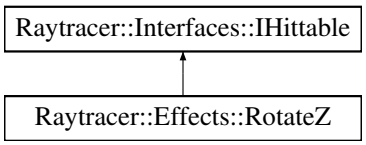
- virtual Arguments::ArgumentKind **kind** () const =0

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

- /Users/riosj1/Code/raytracer/include/arguments/Effects.hpp

7.52 Raytracer::Effects::RotateZ Class Reference

Inheritance diagram for Raytracer::Effects::RotateZ:



Public Member Functions

- RotateZ (std::shared_ptr< Interfaces::IHittable > object, double angle)

  Construct a new RotateZ object.
- bool hit (const Core::Ray &ray, Utils::Interval interval, Core::Payload &payload) const override

  Check if the ray hits the rotated object.
- Utils::AxisAlignedBBox boundingBox () const override

  Get the bounding box of the rotated object.

7.52.1 Constructor & Destructor Documentation

7.52.1.1 RotateZ()

```
Raytracer::Effects::RotateZ::RotateZ (
            std::shared_ptr< Interfaces::IHittable > object,
            double angle )
```

Construct a new RotateZ object.

This function constructs a new RotateZ object with the given object and angle. The object is rotated around the z-axis by the given angle. The bounding box of the object is updated with the rotated bounding box.

Parameters

object	The object to rotate.
angle	The angle to rotate the object by.

Returns

A new RotateZ object.

7.52.2 Member Function Documentation

7.52.2.1 boundingBox()

```
Raytracer::Utils::AxisAlignedBBox Raytracer::Effects::RotateZ::boundingBox ( ) const  [override],
[virtual]
```

Get the bounding box of the rotated object.

This function returns the bounding box of the rotated object.

Returns

The bounding box of the rotated object.

Implements Raytracer::Interfaces::IHittable.

7.52.2.2 hit()

```
bool Raytracer::Effects::RotateZ::hit (
            const Core::Ray & ray,
            Utils::Interval interval,
            Core::Payload & payload ) const  [override], [virtual]
```

Check if the ray hits the rotated object.

This function checks if the ray hits the rotated object. The function returns true if the ray hits the rotated object. The function returns false if the ray does not hit the rotated object. The function updates the payload with the hit information.

Parameters

ray	The ray to check for hits.
interval	The interval to check for hits.

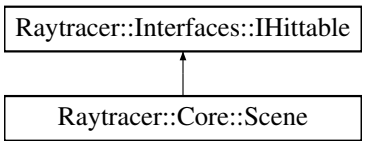
Implements Raytracer::Interfaces::IHittable.

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

- /Users/riosj1/Code/raytracer/include/effects/RotateZ.hpp
- /Users/riosj1/Code/raytracer/sources/effects/RotateZ.cpp

7.53 Raytracer::Core::Scene Class Reference

Inheritance diagram for Raytracer::Core::Scene:



Public Member Functions

- Scene (std::shared_ptr< Interfaces::IHittable > object)

  Construct a new Scene object.
- void clear ()

  Clear the scene.
- void add (std::shared_ptr< Interfaces::IHittable > object)

  Add an object to the scene.
- bool hit (const Core::Ray &ray, Utils::Interval interval, Core::Payload &payload) const override

  Check if the ray hits anything in the scene.
- Utils::AxisAlignedBBox boundingBox () const override

  Get the bounding box of the scene.

7.53.1 Constructor & Destructor Documentation

7.53.1.1 Scene()

```
Raytracer::Core::Scene::Scene (
            std::shared_ptr< Interfaces::IHittable > object )
```

Construct a new Scene object.

This function constructs a new Scene object with the given object. The object is added to the list of objects in the scene. The bounding box of the scene is updated with the bounding box of the object.

Parameters

object	The object to add to the scene.

Returns

A new Scene object.

7.53.2 Member Function Documentation

7.53.2.1 add()

```
void Raytracer::Core::Scene::add (
            std::shared_ptr< Interfaces::IHittable > object )
```

Add an object to the scene.

This function adds an object to the list of objects in the scene. The bounding box of the scene is updated with the bounding box of the object.

Parameters

object	The object to add to the scene.

Returns

void

7.53.2.2 boundingBox()

```
Raytracer::Utils::AxisAlignedBBox Raytracer::Core::Scene::boundingBox ( ) const  [override], [virtual]
```

Get the bounding box of the scene.

This function returns the bounding box of the scene.

Returns

The bounding box of the scene.

Implements Raytracer::Interfaces::IHittable.

7.53.2.3 clear()

```
void Raytracer::Core::Scene::clear ( )
```

Clear the scene.

This function clears the list of objects in the scene. The bounding box of the scene is reset to the default value. The scene is empty after this function is called.

Returns

void

7.53.2.4 hit()

```
bool Raytracer::Core::Scene::hit (
            const Core::Ray & ray,
            Utils::Interval interval,
            Core::Payload & payload ) const  [override], [virtual]
```

Check if the ray hits anything in the scene.

This function checks if the ray hits anything in the scene. The function returns true if the ray hits anything in the scene. The function returns false if the ray does not hit anything in the scene. The function updates the payload with the hit information.

Parameters

ray	The ray to check for hits.
interval	The interval to check for hits.
payload	The payload to update with the hit information.

Returns

true if the ray hits anything in the scene, false otherwise.

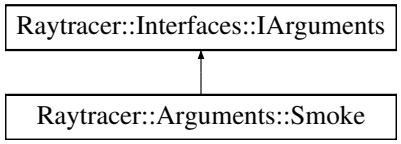
Implements Raytracer::Interfaces::IHittable.

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

- /Users/riosj1/Code/raytracer/include/core/Scene.hpp
- /Users/riosj1/Code/raytracer/sources/core/Scene.cpp

7.54 Raytracer::Arguments::Smoke Class Reference

Inheritance diagram for Raytracer::Arguments::Smoke:



Public Member Functions

- **Smoke** (std::shared_ptr< Interfaces::IHittable > object, double density, Utils::Color color)
- **Smoke** (std::shared_ptr< Interfaces::IHittable > object, double density, std::shared_ptr< Interfaces::ITexture > texture)
- **GET_SET** (double, density)
- **GET_SET** (Utils::Color, color)
- **GET_SET** (std::shared_ptr< Interfaces::ITexture >, texture)
- **GET_SET** (std::shared_ptr< Interfaces::IHittable >, object)
- **ARG_KIND** (_kind)

Public Member Functions inherited from Raytracer::Interfaces::IArguments

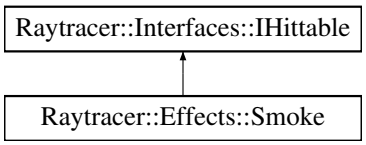
- virtual Arguments::ArgumentKind **kind** () const =0

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

- /Users/riosj1/Code/raytracer/include/arguments/Effects.hpp

7.55 Raytracer::Effects::Smoke Class Reference

Inheritance diagram for Raytracer::Effects::Smoke:



Public Member Functions

- Smoke (std::shared_ptr< Interfaces::IHittable > boundary, double density, std::shared_ptr< Interfaces::ITexture > texture)

  Construct a new Smoke object.
- Smoke (std::shared_ptr< Interfaces::IHittable > boundary, double density, const Utils::Color &albedo)

  Construct a new Smoke object.
- bool hit (const Core::Ray &ray, Utils::Interval interval, Core::Payload &payload) const override

  Check if the ray hits the smoke.
- Utils::AxisAlignedBBox boundingBox () const override

  Get the bounding box of the smoke.

7.55.1 Constructor & Destructor Documentation

7.55.1.1 Smoke() [1/2]

```
Raytracer::Effects::Smoke::Smoke (
            std::shared_ptr< Interfaces::IHittable > boundary,
            double density,
            std::shared_ptr< Interfaces::ITexture > texture )
```

Construct a new Smoke object.

This function constructs a new Smoke object with the given boundary, density, and texture. The smoke is created within the boundary with the given density and texture. The phase function of the smoke is set to isotropic with the given texture.

Parameters

boundary	The boundary to create the smoke within.
density	The density of the smoke.
texture	The texture of the smoke.

Returns

A new Smoke object.

7.55.1.2 Smoke() [2/2]

```
Raytracer::Effects::Smoke::Smoke (
            std::shared_ptr< Interfaces::IHittable > boundary,
            double density,
            const Utils::Color & albedo )
```

Construct a new Smoke object.

This function constructs a new Smoke object with the given boundary, density, and albedo. The smoke is created within the boundary with the given density and albedo. The phase function of the smoke is set to isotropic with the given albedo.

Parameters

boundary	The boundary to create the smoke within.
density	The density of the smoke.
albedo	The albedo of the smoke.

Returns

A new Smoke object.

7.55.2 Member Function Documentation

7.55.2.1 boundingBox()

```
Raytracer::Utils::AxisAlignedBBox Raytracer::Effects::Smoke::boundingBox ( ) const  [override],
[virtual]
```

Get the bounding box of the smoke.

This function returns the bounding box of the smoke.

Returns

The bounding box of the smoke.

Implements Raytracer::Interfaces::IHittable.

7.55.2.2 hit()

```
bool Raytracer::Effects::Smoke::hit (
            const Core::Ray & ray,
            Utils::Interval interval,
            Core::Payload & payload ) const  [override], [virtual]
```

Check if the ray hits the smoke.

This function checks if the ray hits the smoke. The function returns true if the ray hits the smoke. The function returns false if the ray does not hit the smoke. The function updates the payload with the hit information.

Parameters

ray	The ray to check for hits.
interval	The interval to check for hits.
payload	The payload to update with the hit information.

Returns

True if the ray hits the smoke, false otherwise.

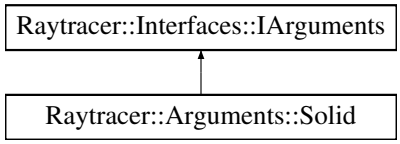
Implements Raytracer::Interfaces::IHittable.

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

- /Users/riosj1/Code/raytracer/include/effects/Smoke.hpp
- /Users/riosj1/Code/raytracer/sources/effects/Smoke.cpp

7.56 Raytracer::Arguments::Solid Class Reference

Inheritance diagram for Raytracer::Arguments::Solid:



Public Member Functions

- **Solid** (Utils::Vec3 color)
- **Solid** (double r, double g, double b)
- **GET_SET** (Utils::Vec3, color)
- **ARG_KIND** (_kind)

Public Member Functions inherited from Raytracer::Interfaces::IArguments

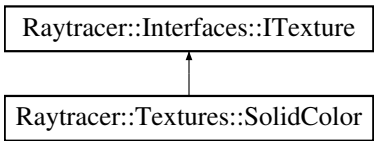
- virtual Arguments::ArgumentKind **kind** () const =0

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

- /Users/riosj1/Code/raytracer/include/arguments/Textures.hpp

7.57 Raytracer::Textures::SolidColor Class Reference

Inheritance diagram for Raytracer::Textures::SolidColor:



Public Member Functions

- SolidColor (const Utils::Color &albedo)

  Construct a new SolidColor object.
- SolidColor (double red, double green, double blue)

  Construct a new SolidColor object.
- Utils::Color value (double u, double v, const Utils::Point3 &point) const override

  Get the value of the solid color texture.

7.57.1 Constructor & Destructor Documentation

7.57.1.1 SolidColor() [1/2]

```
Raytracer::Textures::SolidColor::SolidColor (
            const Utils::Color & albedo )
```

Construct a new SolidColor object.

This function constructs a new SolidColor object with the given albedo. The SolidColor texture is a texture that generates a solid color.

Parameters

albedo	The albedo of the solid color texture.

Returns

A new SolidColor object.

7.57.1.2 SolidColor() [2/2]

```
Raytracer::Textures::SolidColor::SolidColor (
            double red,
            double green,
            double blue )
```

Construct a new SolidColor object.

This function constructs a new SolidColor object with the given red, green, and blue values. The SolidColor texture is a texture that generates a solid color.

Parameters

red	The red value of the solid color texture.
green	The green value of the solid color texture.
blue	The blue value of the solid color texture.

Returns

A new SolidColor object.

7.57.2 Member Function Documentation

7.57.2.1 value()

```
Raytracer::Utils::Color Raytracer::Textures::SolidColor::value (
            double u,
            double v,
            const Utils::Point3 & point ) const  [override], [virtual]
```

Get the value of the solid color texture.

This function returns the value of the solid color texture at the given UV coordinates and point.

Parameters

u	The U coordinate.
v	The V coordinate.
point	The point.

Returns

The value of the solid color texture.

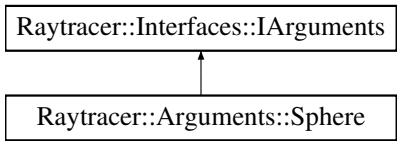
Implements Raytracer::Interfaces::ITexture.

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

- /Users/riosj1/Code/raytracer/include/textures/SolidColor.hpp
- /Users/riosj1/Code/raytracer/sources/textures/SolidColor.cpp

7.58 Raytracer::Arguments::Sphere Class Reference

Inheritance diagram for Raytracer::Arguments::Sphere:



Public Member Functions

- **Sphere** (Utils::Point3 center, double radius, std::shared_ptr< Interfaces::IMaterial > material)
- **Sphere** (Utils::Point3 centerOne, Utils::Point3 centerTwo, double radius, std::shared_ptr< Interfaces::IMaterial > material)
- **GET_SET** (Utils::Point3, center)
- **GET_SET** (Utils::Point3, centerTwo)
- **GET_SET** (double, radius)
- **GET_SET** (std::shared_ptr< Interfaces::IMaterial >, material)
- **ARG_KIND** (_kind)

Public Member Functions inherited from Raytracer::Interfaces::IArguments

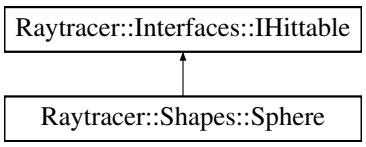
- virtual Arguments::ArgumentKind **kind** () const =0

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

- /Users/riosj1/Code/raytracer/include/arguments/Shapes.hpp

7.59 Raytracer::Shapes::Sphere Class Reference

Inheritance diagram for Raytracer::Shapes::Sphere:



Public Member Functions

- Sphere (const Utils::Point3 &center, double radius, std::shared_ptr< Interfaces::IMaterial > material)

  Construct a new Sphere object.
- Sphere (const Utils::Point3 &centerOne, const Utils::Point3 &centerTwo, double radius, std::shared_ptr< Interfaces::IMaterial > material)

  Construct a new Sphere object.
- bool hit (const Core::Ray &ray, Utils::Interval interval, Core::Payload &hit) const override

  Check if the ray hits the sphere.
- Utils::Point3 sphereCenter (double time) const

  Get the center of the sphere at the given time.
- Utils::AxisAlignedBBox boundingBox () const override

  Get the bounding box of the sphere.

Static Public Member Functions

- static void getSphereUV (const Utils::Point3 &point, double &u, double &v)

  Get the UV coordinates of the sphere.

7.59.1 Constructor & Destructor Documentation

7.59.1.1 Sphere() [1/2]

```
Raytracer::Shapes::Sphere::Sphere (
            const Utils::Point3 & center,
            double radius,
            std::shared_ptr< Interfaces::IMaterial > material )
```

Construct a new Sphere object.

This function constructs a new Sphere object with the given center, radius, and material. The sphere is centered at the given center with the given radius and material.

Parameters

center	The center of the sphere.
radius	The radius of the sphere.
material	The material of the sphere.

Returns

A new Sphere object.

7.59.1.2 Sphere() [2/2]

```
Raytracer::Shapes::Sphere::Sphere (
            const Utils::Point3 & one,
            const Utils::Point3 & two,
            double radius,
            std::shared_ptr< Interfaces::IMaterial > material )
```

Construct a new Sphere object.

This function constructs a new Sphere object with the given centers, radius, and material. The sphere is centered at the given centers with the given radius and material.

Parameters

one	The first center of the sphere.
two	The second center of the sphere.
radius	The radius of the sphere.
material	The material of the sphere.

Returns

A new Sphere object.

7.59.2 Member Function Documentation

7.59.2.1 boundingBox()

```
Raytracer::Utils::AxisAlignedBBox Raytracer::Shapes::Sphere::boundingBox ( ) const  [override],
[virtual]
```

Get the bounding box of the sphere.

This function returns the bounding box of the sphere.

Returns

The bounding box of the sphere.

Implements Raytracer::Interfaces::IHittable.

7.59.2.2 getSphereUV()

```
void Raytracer::Shapes::Sphere::getSphereUV (
            const Utils::Point3 & point,
            double & u,
            double & v )  [static]
```

Get the UV coordinates of the sphere.

This function returns the UV coordinates of the sphere.

Parameters

point	The point to get the UV coordinates.
u	The U coordinate of the UV coordinates.
v	The V coordinate of the UV coordinates.

7.59.2.3 hit()

```
bool Raytracer::Shapes::Sphere::hit (
            const Core::Ray & ray,
            Utils::Interval interval,
            Core::Payload & payload ) const  [override], [virtual]
```

Check if the ray hits the sphere.

This function checks if the ray hits the sphere. The function returns true if the ray hits the sphere. The function returns false if the ray does not hit the sphere. The function updates the payload with the hit information.

Parameters

ray	The ray to check for hits.
interval	The interval to check for hits.
payload	The payload to update with the hit information.

Returns

true if the ray hits the sphere, false otherwise.

Implements Raytracer::Interfaces::IHittable.

7.59.2.4 sphereCenter()

```
Raytracer::Utils::Point3 Raytracer::Shapes::Sphere::sphereCenter (
            double time ) const
```

Get the center of the sphere at the given time.

This function returns the center of the sphere at the given time.

Parameters

time	The time to get the center of the sphere.

Returns

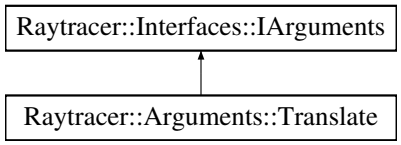
The center of the sphere at the given time.

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

- /Users/riosj1/Code/raytracer/include/shapes/Sphere.hpp
- /Users/riosj1/Code/raytracer/sources/shapes/Sphere.cpp

7.60 Raytracer::Arguments::Translate Class Reference

Inheritance diagram for Raytracer::Arguments::Translate:



Public Member Functions

- **Translate** (std::shared_ptr< Interfaces::IHittable > object, Utils::Vec3 offset)
- **GET_SET** (Utils::Vec3, offset)
- **GET_SET** (std::shared_ptr< Interfaces::IHittable >, object)
- **ARG_KIND** (ArgumentKind::ARG_TRANSLATE)

Public Member Functions inherited from Raytracer::Interfaces::IArguments

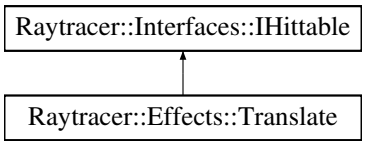
- virtual Arguments::ArgumentKind **kind** () const =0

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

- /Users/riosj1/Code/raytracer/include/arguments/Effects.hpp

7.61 Raytracer::Effects::Translate Class Reference

Inheritance diagram for Raytracer::Effects::Translate:



Public Member Functions

- Translate (std::shared_ptr< Interfaces::IHittable > object, const Utils::Vec3 &offset)

  Construct a new Translate object.
- bool hit (const Core::Ray &ray, Utils::Interval interval, Core::Payload &payload) const override

  Check if the ray hits the translated object.
- Utils::AxisAlignedBBox boundingBox () const override

  Get the bounding box of the translated object.

7.61.1 Constructor & Destructor Documentation

7.61.1.1 Translate()

```
Raytracer::Effects::Translate::Translate (
            std::shared_ptr< Interfaces::IHittable > object,
            const Utils::Vec3 & offset )
```

Construct a new Translate object.

This function constructs a new Translate object with the given object and offset. The object is translated by the given offset.

Parameters

object	The object to translate.
offset	The offset to translate the object by.

Returns

A new Translate object.

7.61.2 Member Function Documentation

7.61.2.1 boundingBox()

```
Raytracer::Utils::AxisAlignedBBox Raytracer::Effects::Translate::boundingBox ( ) const  [override],
[virtual]
```

Get the bounding box of the translated object.

This function returns the bounding box of the translated object.

Returns

The bounding box of the translated object.

Implements Raytracer::Interfaces::IHittable.

7.61.2.2 hit()

```
bool Raytracer::Effects::Translate::hit (
            const Core::Ray & ray,
            Utils::Interval interval,
            Core::Payload & payload ) const  [override], [virtual]
```

Check if the ray hits the translated object.

This function checks if the ray hits the translated object. The function returns true if the ray hits the translated object. The function returns false if the ray does not hit the translated object. The function updates the payload with the hit information.

Parameters

ray	The ray to check for hits.
interval	The interval to check for hits.
payload	The payload to update with the hit information.

Returns

true if the ray hits the translated object, false otherwise.

Implements Raytracer::Interfaces::IHittable.

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

- /Users/riosj1/Code/raytracer/include/effects/Translate.hpp
- /Users/riosj1/Code/raytracer/sources/effects/Translate.cpp

7.62 Raytracer::Utils::VecN< T, N > Class Template Reference

Public Member Functions

- **VecN** (T e0, T e1, T e2)
- T **x** () const
- T **y** () const
- T **z** () const
- VecN **operator-** () const
- T **operator[ ]** (std::size_t i) const
- T & **operator[ ]** (std::size_t i)
- VecN & **operator+=** (const VecN &v)
- VecN & **operator∗=** (double t)
- VecN & **operator/=** (double t)
- double **length** () const
- double **lengthSquared** () const
- bool **nearZero** () const
- VecN & **normalize** ()

Static Public Member Functions

- static VecN **random** ()
- static VecN **random** (double min, double max)

Public Attributes

- T **e** [N]

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

- /Users/riosj1/Code/raytracer/include/utils/VecN.hpp

Chapter 8

File Documentation

8.1 Effects.hpp

```
00001 #include "Common.hpp"
00002 #include "arguments/Kinds.hpp"
00003 #include "interfaces/IArguments.hpp"
00004 #include "interfaces/IHittable.hpp"
00005 #include "interfaces/ITexture.hpp"
00006 #include "utils/VecN.hpp"
00007
00008 #ifndef __ARG_EFFECTS_HPP__
00009     #define __ARG_EFFECTS_HPP__
00010
00011 namespace Raytracer::Arguments
00012 {
00013     class RotateX : public Interfaces::IArguments {
00014       private:
00015         double _angle;
00016         std::shared_ptr<Interfaces::IHittable> _object = nullptr;
00017
00018       public:
00019         RotateX(std::shared_ptr<Interfaces::IHittable> object, double angle)
00020             : _angle(angle), _object(object)
00021         {
00022         }
00023         GET_SET(double, angle);
00024         GET_SET(std::shared_ptr<Interfaces::IHittable>, object);
00025         ARG_KIND(ArgumentKind::ARG_ROTATE_X);
00026     };
00027
00028     class RotateY : public Interfaces::IArguments {
00029       private:
00030         double _angle;
00031         std::shared_ptr<Interfaces::IHittable> _object = nullptr;
00032
00033       public:
00034         RotateY(std::shared_ptr<Interfaces::IHittable> object, double angle)
00035             : _angle(angle), _object(object)
00036         {
00037         }
00038         GET_SET(double, angle);
00039         GET_SET(std::shared_ptr<Interfaces::IHittable>, object);
00040         ARG_KIND(ArgumentKind::ARG_ROTATE_Y);
00041     };
00042
00043     class RotateZ : public Interfaces::IArguments {
00044       private:
00045         double _angle;
00046         std::shared_ptr<Interfaces::IHittable> _object = nullptr;
00047
00048       public:
00049         RotateZ(std::shared_ptr<Interfaces::IHittable> object, double angle)
00050             : _angle(angle), _object(object)
00051         {
00052         }
00053         GET_SET(double, angle);
00054         GET_SET(std::shared_ptr<Interfaces::IHittable>, object);
00055         ARG_KIND(ArgumentKind::ARG_ROTATE_Z);
00056     };
00057
00058     class Smoke : public Interfaces::IArguments {
```

```
00059       private:
00060         ArgumentKind _kind;
00061         double _density;
00062         Utils::Color _color;
00063         std::shared_ptr<Interfaces::ITexture> _texture = nullptr;
00064         std::shared_ptr<Interfaces::IHittable> _object = nullptr;
00065
00066       public:
00067         Smoke(std::shared_ptr<Interfaces::IHittable> object, double density,
00068             Utils::Color color)
00069             : _density(density), _color(color), _object(object)
00070         {
00071             _kind = ArgumentKind::ARG_SMOKE_COLOR;
00072         }
00073         Smoke(std::shared_ptr<Interfaces::IHittable> object, double density,
00074             std::shared_ptr<Interfaces::ITexture> texture)
00075             : _density(density), _texture(texture), _object(object)
00076         {
00077             _kind = ArgumentKind::ARG_SMOKE_TEXTURE;
00078         }
00079         GET_SET(double, density);
00080         GET_SET(Utils::Color, color);
00081         GET_SET(std::shared_ptr<Interfaces::ITexture>, texture);
00082         GET_SET(std::shared_ptr<Interfaces::IHittable>, object);
00083         ARG_KIND(_kind);
00084     };
00085
00086     class Translate : public Interfaces::IArguments {
00087       private:
00088         Utils::Vec3 _offset;
00089         std::shared_ptr<Interfaces::IHittable> _object = nullptr;
00090
00091       public:
00092         Translate(
00093             std::shared_ptr<Interfaces::IHittable> object, Utils::Vec3 offset)
00094             : _offset(offset), _object(object)
00095         {
00096         }
00097         GET_SET(Utils::Vec3, offset);
00098         GET_SET(std::shared_ptr<Interfaces::IHittable>, object);
00099         ARG_KIND(ArgumentKind::ARG_TRANSLATE);
00100     };
00101 } // namespace Raytracer::Arguments
00102 #endif /* __ARG_EFFECTS_HPP__ */
```

8.2 Kinds.hpp

```
00001 #ifndef __ARG_KINDS_HPP__
00002 #define __ARG_KINDS_HPP__
00003
00004 namespace Raytracer::Arguments
00005 {
00006     enum class ArgumentKind {
00007         /* Textures */
00008         ARG_SOLID_COLOR,
00009         ARG_SOLID_RGB,
00010         ARG_NOISE,
00011         ARG_IMAGE,
00012         ARG_CHECKER_TEXTURE,
00013         ARG_CHECKER_COLOR,
00014         /* Effects */
00015         ARG_ROTATE_X,
00016         ARG_ROTATE_Y,
00017         ARG_ROTATE_Z,
00018         ARG_SMOKE_TEXTURE,
00019         ARG_SMOKE_COLOR,
00020         ARG_TRANSLATE,
00021         /* Materials */
00022         ARG_LAMBERTIAN_COLOR,
00023         ARG_LAMBERTIAN_TEXTURE,
00024         ARG_DIELECTRIC,
00025         ARG_DIELECTRIC_COLOR,
00026         ARG_DIFFUSE_LIGHT_COLOR,
00027         ARG_DIFFUSE_LIGHT_TEXTURE,
00028         ARG_ISOTROPIC_COLOR,
00029         ARG_ISOTROPIC_TEXTURE,
00030         ARG_METAL,
00031         /* Shapes */
00032         ARG_CONE,
00033         ARG_CYLINDER,
00034         ARG_PLANE,
00035         ARG_QUAD,
00036         ARG_SPHERE,
```

```
00037         ARG_SPHERE_MOVING,
00038         ARG_BOX,
00039     };
00040 }
00041
00042 #define ARG_KIND(k)                     \
00043     ArgumentKind kind() const override \
00044     {                                   \
00045         return k;                       \
00046     }
00047
00048 #endif /* __ARG_KINDS_HPP__ */
```

8.3 Materials.hpp

```
00001 #include <memory>
00002 #include "Common.hpp"
00003 #include "arguments/Kinds.hpp"
00004 #include "interfaces/IArguments.hpp"
00005 #include "interfaces/ITexture.hpp"
00006 #include "utils/VecN.hpp"
00007
00008 #ifndef __ARG_MATERIALS_HPP__
00009     #define __ARG_MATERIALS_HPP__
00010
00011 namespace Raytracer::Arguments
00012 {
00013     class Lambertian : public Interfaces::IArguments {
00014       private:
00015         ArgumentKind _kind;
00016         Utils::Vec3 _color;
00017         std::shared_ptr<Interfaces::ITexture> _texture = nullptr;
00018
00019       public:
00020         Lambertian(Utils::Vec3 color) : _color(color)
00021         {
00022             _kind = ArgumentKind::ARG_LAMBERTIAN_COLOR;
00023         }
00024         Lambertian(double r, double g, double b) : _color(r, g, b)
00025         {
00026             _kind = ArgumentKind::ARG_LAMBERTIAN_TEXTURE;
00027         }
00028         Lambertian(std::shared_ptr<Interfaces::ITexture> texture)
00029             : _texture(texture)
00030         {
00031             _kind = ArgumentKind::ARG_LAMBERTIAN_TEXTURE;
00032         }
00033         GET_SET(Utils::Vec3, color);
00034         GET_SET(std::shared_ptr<Interfaces::ITexture>, texture);
00035         ARG_KIND(_kind);
00036     };
00037
00038     class Dielectric : public Interfaces::IArguments {
00039       private:
00040         ArgumentKind _kind;
00041         double _refractionIndex;
00042         Utils::Color _color;
00043
00044       public:
00045         Dielectric(double refractionIndex) : _refractionIndex(refractionIndex)
00046         {
00047             _kind = ArgumentKind::ARG_DIELECTRIC;
00048         }
00049         Dielectric(double refractionIndex, Utils::Color color)
00050             : _refractionIndex(refractionIndex)
00051         {
00052             _kind = ArgumentKind::ARG_DIELECTRIC_COLOR;
00053         }
00054         GET_SET(double, refractionIndex);
00055         GET_SET(Utils::Color, color);
00056         ARG_KIND(_kind);
00057     };
00058
00059     class DiffuseLight : public Interfaces::IArguments {
00060       private:
00061         ArgumentKind _kind;
00062         Utils::Vec3 _color;
00063         std::shared_ptr<Interfaces::ITexture> _texture = nullptr;
00064
00065       public:
00066         DiffuseLight(Utils::Vec3 color) : _color(color)
00067         {
00068             _kind = ArgumentKind::ARG_DIFFUSE_LIGHT_COLOR;
```

```
00069         }
00070         DiffuseLight(std::shared_ptr<Interfaces::ITexture> texture)
00071             : _texture(texture)
00072         {
00073             _kind = ArgumentKind::ARG_DIFFUSE_LIGHT_TEXTURE;
00074         }
00075         GET_SET(Utils::Vec3, color);
00076         GET_SET(std::shared_ptr<Interfaces::ITexture>, texture);
00077         ARG_KIND(_kind);
00078     };
00079
00080     class Isotropic : public Interfaces::IArguments {
00081       private:
00082         ArgumentKind _kind;
00083         Utils::Color _color;
00084         std::shared_ptr<Interfaces::ITexture> _texture = nullptr;
00085
00086       public:
00087         Isotropic(Utils::Color color) : _color(color)
00088         {
00089             _kind = ArgumentKind::ARG_ISOTROPIC_COLOR;
00090         }
00091         Isotropic(std::shared_ptr<Interfaces::ITexture> texture)
00092             : _texture(texture)
00093         {
00094             _kind = ArgumentKind::ARG_ISOTROPIC_TEXTURE;
00095         }
00096         GET_SET(Utils::Color, color);
00097         GET_SET(std::shared_ptr<Interfaces::ITexture>, texture);
00098         ARG_KIND(_kind);
00099     };
00100
00101     class Metal : public Interfaces::IArguments {
00102       private:
00103         Utils::Vec3 _color;
00104         double _fuzz;
00105
00106       public:
00107         Metal(Utils::Vec3 color, double fuzz) : _color(color), _fuzz(fuzz)
00108         {
00109         }
00110         GET_SET(Utils::Vec3, color);
00111         GET_SET(double, fuzz);
00112         ARG_KIND(ArgumentKind::ARG_METAL);
00113     };
00114 } // namespace Raytracer::Arguments
00115
00116 #endif /* __ARG_MATERIALS_HPP__ */
```

8.4 Shapes.hpp

```
00001 #include <memory>
00002 #include "Common.hpp"
00003 #include "arguments/Kinds.hpp"
00004 #include "interfaces/IArguments.hpp"
00005 #include "interfaces/IMaterial.hpp"
00006 #include "interfaces/ITexture.hpp"
00007 #include "utils/VecN.hpp"
00008
00009 #ifndef __ARG_SHAPES_HPP__
00010     #define __ARG_SHAPES_HPP__
00011
00012 namespace Raytracer::Arguments
00013 {
00014     class Cone : public Interfaces::IArguments {
00015       private:
00016         double _radius;
00017         double _height;
00018         Utils::Point3 _center;
00019         std::shared_ptr<Interfaces::IMaterial> _material = nullptr;
00020
00021       public:
00022         Cone(Utils::Point3 &center, double radius, double height,
00023             std::shared_ptr<Interfaces::IMaterial> material)
00024             : _radius(radius), _height(height), _center(center),
00025               _material(material)
00026         {
00027         }
00028         GET_SET(Utils::Point3, center);
00029         GET_SET(double, radius);
00030         GET_SET(double, height);
00031         GET_SET(std::shared_ptr<Interfaces::IMaterial>, material);
00032         ARG_KIND(ArgumentKind::ARG_CONE);
```

```
00033     };
00034
00035     class Cylinder : public Interfaces::IArguments {
00036       private:
00037         double _radius;
00038         double _height;
00039         Utils::Point3 _center;
00040         std::shared_ptr<Interfaces::IMaterial> _material = nullptr;
00041
00042       public:
00043         Cylinder(Utils::Point3 &center, double radius, double height,
00044             std::shared_ptr<Interfaces::IMaterial> material)
00045             : _radius(radius), _height(height), _center(center),
00046               _material(material)
00047         {
00048         }
00049         GET_SET(Utils::Point3, center);
00050         GET_SET(double, radius);
00051         GET_SET(double, height);
00052         GET_SET(std::shared_ptr<Interfaces::IMaterial>, material);
00053         ARG_KIND(ArgumentKind::ARG_CYLINDER);
00054     };
00055
00056     class Plane : public Interfaces::IArguments {
00057       private:
00058         Utils::Point3 _point;
00059         Utils::Vec3 _normal;
00060         std::shared_ptr<Interfaces::IMaterial> _material = nullptr;
00061
00062       public:
00063         Plane(Utils::Point3 &point, Utils::Vec3 &normal,
00064             std::shared_ptr<Interfaces::IMaterial> material)
00065             : _point(point), _normal(normal), _material(material)
00066         {
00067         }
00068         GET_SET(Utils::Point3, point);
00069         GET_SET(Utils::Vec3, normal);
00070         GET_SET(std::shared_ptr<Interfaces::IMaterial>, material);
00071         ARG_KIND(ArgumentKind::ARG_PLANE);
00072     };
00073
00074     class Quad : public Interfaces::IArguments {
00075       private:
00076         Utils::Point3 _Q;
00077         Utils::Point3 _u;
00078         Utils::Point3 _v;
00079         std::shared_ptr<Interfaces::IMaterial> _material = nullptr;
00080
00081       public:
00082         Quad(Utils::Point3 &Q, Utils::Point3 &u, Utils::Point3 &v,
00083             std::shared_ptr<Interfaces::IMaterial> material)
00084             : _Q(Q), _u(u), _v(v), _material(material)
00085         {
00086         }
00087         GET_SET(Utils::Point3, Q);
00088         GET_SET(Utils::Point3, u);
00089         GET_SET(Utils::Point3, v);
00090         GET_SET(std::shared_ptr<Interfaces::IMaterial>, material);
00091         ARG_KIND(ArgumentKind::ARG_QUAD);
00092     };
00093
00094     class Sphere : public Interfaces::IArguments {
00095       private:
00096         ArgumentKind _kind;
00097         Utils::Point3 _center;
00098         Utils::Point3 _centerTwo;
00099         double _radius;
00100         std::shared_ptr<Interfaces::IMaterial> _material = nullptr;
00101
00102       public:
00103         Sphere(Utils::Point3 center, double radius,
00104             std::shared_ptr<Interfaces::IMaterial> material)
00105             : _center(center), _radius(radius), _material(material)
00106         {
00107             _kind = ArgumentKind::ARG_SPHERE;
00108         }
00109         Sphere(Utils::Point3 centerOne, Utils::Point3 centerTwo, double radius,
00110             std::shared_ptr<Interfaces::IMaterial> material)
00111             : _center(centerOne), _centerTwo(centerTwo), _radius(radius),
00112               _material(material)
00113         {
00114             _kind = ArgumentKind::ARG_SPHERE_MOVING;
00115         }
00116         GET_SET(Utils::Point3, center);
00117         GET_SET(Utils::Point3, centerTwo);
00118         GET_SET(double, radius);
00119         GET_SET(std::shared_ptr<Interfaces::IMaterial>, material);
```

```
00120         ARG_KIND(_kind);
00121     };
00122
00123     class Box : public Interfaces::IArguments {
00124       private:
00125         Utils::Point3 _pointOne;
00126         Utils::Point3 _pointTwo;
00127         std::shared_ptr<Interfaces::IMaterial> _material = nullptr;
00128
00129       public:
00130         Box(Utils::Point3 pointOne, Utils::Point3 pointTwo,
00131             std::shared_ptr<Interfaces::IMaterial> material)
00132             : _pointOne(pointOne), _pointTwo(pointTwo), _material(material)
00133         {
00134         }
00135         GET_SET(Utils::Point3, pointOne);
00136         GET_SET(Utils::Point3, pointTwo);
00137         GET_SET(std::shared_ptr<Interfaces::IMaterial>, material);
00138         ARG_KIND(ArgumentKind::ARG_BOX);
00139     };
00140 } // namespace Raytracer::Arguments
00141
00142 #endif /* __ARG_SHAPES_HPP__ */
```

8.5 Textures.hpp

```
00001 #include <memory>
00002 #include "Common.hpp"
00003 #include "arguments/Kinds.hpp"
00004 #include "interfaces/IArguments.hpp"
00005 #include "interfaces/ITexture.hpp"
00006 #include "utils/VecN.hpp"
00007
00008 #ifndef __ARG_TEXTURES_HPP__
00009     #define __ARG_TEXTURES_HPP__
00010
00011 namespace Raytracer::Arguments
00012 {
00013     class Solid : public Interfaces::IArguments {
00014       private:
00015         ArgumentKind _kind;
00016         Utils::Vec3 _color;
00017
00018       public:
00019         Solid(Utils::Vec3 color) : _color(color)
00020         {
00021             _kind = ArgumentKind::ARG_SOLID_COLOR;
00022         }
00023         Solid(double r, double g, double b) : _color(r, g, b)
00024         {
00025             _kind = ArgumentKind::ARG_SOLID_RGB;
00026         }
00027         GET_SET(Utils::Vec3, color);
00028         ARG_KIND(_kind);
00029     };
00030
00031     class Noise : public Interfaces::IArguments {
00032       private:
00033         double _scale;
00034
00035       public:
00036         Noise(double scale) : _scale(scale)
00037         {
00038         }
00039         GET_SET(double, scale);
00040         ARG_KIND(ArgumentKind::ARG_NOISE);
00041     };
00042
00043     class Image : public Interfaces::IArguments {
00044       private:
00045         std::string _filename;
00046
00047       public:
00048         Image(std::string filename) : _filename(filename)
00049         {
00050         }
00051         GET_SET(std::string, filename);
00052         ARG_KIND(ArgumentKind::ARG_IMAGE);
00053     };
00054
00055     class Checker : public Interfaces::IArguments {
00056       private:
00057         ArgumentKind _kind;
```

```
00058         double _scale;
00059         Utils::Vec3 _color1;
00060         Utils::Vec3 _color2;
00061         std::shared_ptr<Interfaces::ITexture> _texture1 = nullptr;
00062         std::shared_ptr<Interfaces::ITexture> _texture2 = nullptr;
00063
00064       public:
00065         Checker(double scale, Utils::Vec3 color1, Utils::Vec3 color2)
00066             : _scale(scale), _color1(color1), _color2(color2)
00067         {
00068             _kind = ArgumentKind::ARG_CHECKER_COLOR;
00069         }
00070         Checker(double scale, std::shared_ptr<Interfaces::ITexture> texture1,
00071             std::shared_ptr<Interfaces::ITexture> texture2)
00072             : _scale(scale), _texture1(texture1), _texture2(texture2)
00073         {
00074             _kind = ArgumentKind::ARG_CHECKER_TEXTURE;
00075         }
00076         GET_SET(double, scale);
00077         GET_SET(Utils::Vec3, color1);
00078         GET_SET(Utils::Vec3, color2);
00079         GET_SET(std::shared_ptr<Interfaces::ITexture>, texture1);
00080         GET_SET(std::shared_ptr<Interfaces::ITexture>, texture2);
00081         ARG_KIND(_kind);
00082     };
00083 } // namespace Raytracer::Arguments
00084
00085 #endif /* __ARG_TEXTURES_HPP__ */
```

8.6 Common.hpp

```
00001 #ifndef __COMMON_HPP__
00002 #define __COMMON_HPP__
00003
00004 #define GET_SET(type, name)  \
00005     const type &name() const \
00006     {                        \
00007         return _##name;      \
00008     }                        \
00009     type &name()             \
00010     {                        \
00011         return _##name;      \
00012     }                        \
00013     void name(type value)    \
00014     {                        \
00015         _##name = value;     \
00016     }
00017
00018 #endif /* __COMMON_HPP__ */
```

8.7 Factory.hpp

```
00001 #include <functional>
00002 #include <memory>
00003 #include <string>
00004 #include "interfaces/IArguments.hpp"
00005 #include "interfaces/IHittable.hpp"
00006 #include "interfaces/IMaterial.hpp"
00007 #include "interfaces/ITexture.hpp"
00008 #include <type_traits>
00009 #include <unordered_map>
00010
00011 #ifndef __CFG_FACTORY_HPP__
00012     #define __CFG_FACTORY_HPP__
00013
00014 namespace Raytracer::Config
00015 {
00016     enum class ConfigTextures {
00017         TEXTURE_SOLID,
00018         TEXTURE_NOISE,
00019         TEXTURE_IMAGE,
00020         TEXTURE_CHECKER,
00021     };
00022
00023     enum class ConfigEffects {
00024         EFFECT_ROTATE_X,
00025         EFFECT_ROTATE_Y,
00026         EFFECT_ROTATE_Z,
00027         EFFECT_SMOKE,
```

```
00028         EFFECT_TRANSLATE,
00029     };
00030
00031     enum class ConfigMaterials {
00032         MATERIAL_LAMBERTIAN,
00033         MATERIAL_DIELECTRIC,
00034         MATERIAL_DIFFUSE_LIGHT,
00035         MATERIAL_ISOTROPIC,
00036         MATERIAL_METAL,
00037     };
00038
00039     enum class ConfigShapes {
00040         SHAPE_CONE,
00041         SHAPE_CYLINDER,
00042         SHAPE_PLANE,
00043         SHAPE_QUAD,
00044         SHAPE_SPHERE,
00045         SHAPE_MOVING_SPHERE,
00046     };
00047
00048     template <typename I, typename E>
00049         requires std::is_enum_v<E>
00050     using FactoryFunction = std::function<std::shared_ptr<I>(
00051         std::shared_ptr<Interfaces::IArguments>)>;
00052
00053     template <typename I, typename E>
00054     using FactoryMap = std::unordered_map<std::string, FactoryFunction<I, E»;
00055
00056     template <typename T, typename E>
00057     concept isValidEnum = std::is_enum_v<T> && std::is_same_v<T, E>;
00058
00059     class Factory {
00060       private:
00061         Factory() = delete;
00062
00063         template <typename I, typename E>
00064         static std::shared_ptr<I> _get(const FactoryMap<I, E> &map,
00065             const std::string &name,
00066             std::shared_ptr<Interfaces::IArguments> args)
00067         {
00068             return map.at(name)(args);
00069         }
00070
00071       public:
00072         static FactoryMap<Raytracer::Interfaces::ITexture, ConfigTextures>
00073             textures;
00074         static FactoryMap<Raytracer::Interfaces::IHittable, ConfigEffects>
00075             effects;
00076         static FactoryMap<Raytracer::Interfaces::IMaterial, ConfigMaterials>
00077             materials;
00078         static FactoryMap<Raytracer::Interfaces::IHittable, ConfigShapes>
00079             shapes;
00080
00081         template <typename I, typename E>
00082             requires isValidEnum<E, ConfigTextures>
00083         static std::shared_ptr<I> get(const std::string &name,
00084             std::shared_ptr<Interfaces::IArguments> args)
00085         {
00086             return _get<I, E>(textures, name, args);
00087         }
00088
00089         template <typename I, typename E>
00090             requires isValidEnum<E, ConfigEffects>
00091         static std::shared_ptr<I> get(const std::string &name,
00092             std::shared_ptr<Interfaces::IArguments> args)
00093         {
00094             return _get<I, E>(effects, name, args);
00095         }
00096
00097         template <typename I, typename E>
00098             requires isValidEnum<E, ConfigMaterials>
00099         static std::shared_ptr<I> get(const std::string &name,
00100             std::shared_ptr<Interfaces::IArguments> args)
00101         {
00102             return _get<I, E>(materials, name, args);
00103         }
00104
00105         template <typename I, typename E>
00106             requires isValidEnum<E, ConfigShapes>
00107         static std::shared_ptr<I> get(const std::string &name,
00108             std::shared_ptr<Interfaces::IArguments> args)
00109         {
00110             return _get<I, E>(shapes, name, args);
00111         }
00112     };
00113 } // namespace Raytracer::Config
00114
```

```
00115 #endif /* __CFG_FACTORY_HPP__ */
```

8.8 Manager.hpp

```
00001 #include <functional>
00002 #include <libconfig.hh>
00003 #include <memory>
00004 #include <optional>
00005 #include <string>
00006 #include <variant>
00007 #include "Common.hpp"
00008 #include "core/Camera.hpp"
00009 #include "core/Scene.hpp"
00010 #include "interfaces/IArguments.hpp"
00011 #include "interfaces/IHittable.hpp"
00012 #include "interfaces/IMaterial.hpp"
00013 #include "interfaces/ITexture.hpp"
00014 #include "libconfig.h++"
00015 #include <type_traits>
00016 #include <unordered_map>
00017
00018 #ifndef __CFG_MANAGER_HPP__
00019     #define __CFG_MANAGER_HPP__
00020
00021 namespace Raytracer::Config
00022 {
00023     template <typename I>
00024     using ManagerMap = std::unordered_map<std::string, std::shared_ptr<I»;
00025
00026     using KeyTypes = std::tuple<double, int, int, int, Raytracer::Utils::Color,
00027         int, Raytracer::Utils::Point3, Raytracer::Utils::Point3,
00028         Raytracer::Utils::Point3, double>;
00029
00030     template <int I> using KeyType = std::tuple_element_t<I, KeyTypes>;
00031
00032     using CameraTypes = std::variant<int, double, Raytracer::Utils::Vec3>;
00033
00034     class Manager {
00035       private:
00036         Raytracer::Core::Scene _world;
00037         Raytracer::Core::Camera _camera;
00038         std::vector<std::string> _ids;
00039         ManagerMap<Interfaces::ITexture> _textures;
00040         ManagerMap<Interfaces::IHittable> _effects;
00041         ManagerMap<Interfaces::IMaterial> _materials;
00042         ManagerMap<Interfaces::IHittable> _shapes;
00043         std::unordered_map<std::string,
00044             std::function<std::shared_ptr<Interfaces::IArguments>(
00045                 libconfig::Setting &)»
00046             _argumentMap;
00047         std::unordered_map<std::string,
00048             std::function<void(Raytracer::Core::Camera &, CameraTypes &)»
00049             _cameraMap;
00050
00051       public:
00052         Manager();
00053         bool parse(std::string path);
00054         void bootstrap();
00055         void render(bool fast);
00056         GET_SET(Raytracer::Core::Scene, world);
00057         GET_SET(Raytracer::Core::Camera, camera);
00058
00059       private:
00060         template <typename I, typename E>
00061             requires std::is_enum_v<E>
00062         void genericParse(
00063             const libconfig::Setting &arguments, ManagerMap<I> &containerMap);
00064         static Utils::Color parseColor(const libconfig::Setting &color);
00065         template <typename I>
00066         std::shared_ptr<I> retrieve(const libconfig::Setting &arguments,
00067             ManagerMap<I> &containerMap, const std::string &name);
00068         std::shared_ptr<Raytracer::Interfaces::IArguments> create(
00069             const std::string &type, libconfig::Setting &args);
00070         void parseCamera(const libconfig::Setting &camera);
00071         void parseImports(const libconfig::Setting &imports);
00072         template <typename T>
00073             requires std::is_arithmetic_v<T>
00074         std::optional<T> parseOptional(
00075             const libconfig::Setting &setting, std::string &name);
00076         template <typename T>
00077             requires std::is_same_v<T, Raytracer::Utils::Vec3>
00078         std::optional<T> parseOptional(
00079             const libconfig::Setting &setting, std::string &name);
```

```
00080         template <std::size_t I>
00081         void extract(const libconfig::Setting &setting,
00082             std::array<std::string, 10> &keys);
00083         template <std::size_t... Is>
00084         void parseCameraHelper(const libconfig::Setting &camera,
00085             std::array<std::string, 10> &keys, std::index_sequence<Is...>);
00086     };
00087 } // namespace Raytracer::Config
00088
00089 #endif /* __CFG_MANAGER_HPP__ */
```

8.9 Camera.hpp

```
00001 #include <chrono>
00002 #include "Common.hpp"
00003 #include "core/Ray.hpp"
00004 #include "interfaces/IHittable.hpp"
00005 #include "utils/VecN.hpp"
00006
00007 #ifndef __CAMERA_HPP__
00008     #define __CAMERA_HPP__
00009
00010 namespace Raytracer::Core
00011 {
00012     class Camera {
00013       private:
00014         double _aspectRatio = 1.0;
00015         int _imageWidth = 100;
00016         int _samplesPerPixel = 10;
00017         int _maxDepth = 10;
00018         Utils::Color _backgroundColor = Utils::Color(0, 0, 0);
00019
00020         double _vFov = 90;
00021         Utils::Point3 _lookFrom = Utils::Point3(0, 0, 0);
00022         Utils::Point3 _lookAt = Utils::Point3(0, 0, -1);
00023         Utils::Vec3 _vUp = Utils::Vec3(0, 1, 0);
00024
00025         double _defocusAngle = 0;
00026         double _focusDistance = 10;
00027
00028         int _imageHeight;
00029         double _pixelSampleScale;
00030
00031         Utils::Point3 _center;
00032         Utils::Point3 _pixelZeroLoc;
00033         Utils::Vec3 _pixelDeltaU;
00034         Utils::Vec3 _pixelDeltaV;
00035         Utils::Vec3 _u, _v, _w;
00036         Utils::Vec3 _defocusDiskU;
00037         Utils::Vec3 _defocusDiskV;
00038
00039       public:
00040         Camera() = default;
00041         void setup();
00042         void render(const Interfaces::IHittable &world);
00043         Core::Ray getRay(double u, double v) const;
00044         Utils::Vec3 sampleSquare() const;
00045         Utils::Vec3 sampleDisk(double radius) const;
00046         Utils::Vec3 sampleDefocusDisk() const;
00047         Utils::Color rayColor(const Ray ray, int depth,
00048             const Interfaces::IHittable &world) const;
00049         void progress(
00050             const std::chrono::steady_clock::time_point &start, int j) const;
00051         GET_SET(double, aspectRatio)
00052         GET_SET(int, imageWidth)
00053         GET_SET(int, samplesPerPixel)
00054         GET_SET(int, maxDepth)
00055         GET_SET(Utils::Color, backgroundColor)
00056         GET_SET(double, vFov)
00057         GET_SET(Utils::Point3, lookFrom)
00058         GET_SET(Utils::Point3, lookAt)
00059         GET_SET(Utils::Vec3, vUp)
00060         GET_SET(double, defocusAngle)
00061         GET_SET(double, focusDistance)
00062         GET_SET(int, imageHeight)
00063         GET_SET(double, pixelSampleScale)
00064         GET_SET(Utils::Point3, center)
00065         GET_SET(Utils::Vec3, pixelZeroLoc)
00066         GET_SET(Utils::Vec3, pixelDeltaU)
00067         GET_SET(Utils::Vec3, pixelDeltaV)
00068         GET_SET(Utils::Vec3, u)
00069         GET_SET(Utils::Vec3, v)
00070         GET_SET(Utils::Vec3, w)
```

```
00071         GET_SET(Utils::Vec3, defocusDiskU)
00072         GET_SET(Utils::Vec3, defocusDiskV)
00073     };
00074 } // namespace Raytracer::Core
00075
00076 #endif /* __CAMERA_HPP__ */
```

8.10 Payload.hpp

```
00001 #include <memory>
00002 #include "core/Ray.hpp"
00003 #include "interfaces/IMaterial.hpp"
00004 #include "utils/VecN.hpp"
00005
00006 #ifndef __PAYLOAD_HPP__
00007     #define __PAYLOAD_HPP__
00008
00009 namespace Raytracer::Core
00010 {
00011     class Payload {
00012       private:
00013         Utils::Point3 _point;
00014         Utils::Vec3 _normal;
00015         std::shared_ptr<Interfaces::IMaterial> _material;
00016         double _t;
00017         double _u;
00018         double _v;
00019         bool _frontFace;
00020
00021       public:
00022         Payload() = default;
00023         void setFaceNormal(
00024             const Core::Ray &ray, const Utils::Vec3 &outwardNormal);
00025         GET_SET(Utils::Point3, point)
00026         GET_SET(Utils::Vec3, normal)
00027         GET_SET(std::shared_ptr<Interfaces::IMaterial>, material)
00028         GET_SET(double, t)
00029         GET_SET(double, u)
00030         GET_SET(double, v)
00031         GET_SET(bool, frontFace)
00032     };
00033 } // namespace Raytracer::Core
00034
00035 #endif /* __PAYLOAD_HPP__ */
```

8.11 Ray.hpp

```
00001 #include "Common.hpp"
00002 #include "utils/VecN.hpp"
00003
00004 #ifndef __RAY_HPP__
00005     #define __RAY_HPP__
00006
00007 namespace Raytracer::Core
00008 {
00009     class Ray {
00010       private:
00011         Utils::Point3 _origin;
00012         Utils::Vec3 _direction;
00013         double _time;
00014
00015       public:
00016         Ray() = default;
00017         Ray(const Utils::Point3 &origin, const Utils::Vec3 &direction,
00018             double time = 0.0);
00019         Utils::Point3 at(double t) const;
00020         GET_SET(Utils::Point3, origin)
00021         GET_SET(Utils::Vec3, direction)
00022         GET_SET(double, time)
00023     };
00024 } // namespace Raytracer::Core
00025
00026 #endif /* __RAY_HPP__ */
```

8.12 Scene.hpp

```
00001 #include <vector>
```

```
00002 #include "Common.hpp"
00003 #include "interfaces/IHittable.hpp"
00004
00005 #ifndef __SCENE_HPP__
00006     #define __SCENE_HPP__
00007
00008 namespace Raytracer::Core
00009 {
00010     class Scene : public Interfaces::IHittable {
00011       private:
00012         Utils::AxisAlignedBBox _bbox;
00013         std::vector<std::shared_ptr<Interfaces::IHittable» _objects;
00014
00015       public:
00016         Scene() = default;
00017         Scene(std::shared_ptr<Interfaces::IHittable> object);
00018         void clear();
00019         void add(std::shared_ptr<Interfaces::IHittable> object);
00020         bool hit(const Core::Ray &ray, Utils::Interval interval,
00021             Core::Payload &payload) const override;
00022         Utils::AxisAlignedBBox boundingBox() const override;
00023         GET_SET(std::vector<std::shared_ptr<Interfaces::IHittable», objects)
00024     };
00025 } // namespace Raytracer::Core
00026
00027 #endif /* __SCENE_HPP__ */
```

8.13 RotateX.hpp

```
00001 #include "interfaces/IHittable.hpp"
00002
00003 #ifndef __ROTATE_X_HPP__
00004     #define __ROTATE_X_HPP__
00005
00006 namespace Raytracer::Effects
00007 {
00008     class RotateX : public Interfaces::IHittable {
00009       private:
00010         std::shared_ptr<Interfaces::IHittable> _object;
00011         double _sinTheta;
00012         double _cosTheta;
00013         Utils::AxisAlignedBBox _bbox;
00014
00015       public:
00016         RotateX(std::shared_ptr<Interfaces::IHittable> object, double angle);
00017         bool hit(const Core::Ray &ray, Utils::Interval interval,
00018             Core::Payload &payload) const override;
00019         Utils::AxisAlignedBBox boundingBox() const override;
00020     };
00021
00022 } // namespace Raytracer::Effects
00023
00024 #endif /* __ROTATE_X_HPP__ */
```

8.14 RotateY.hpp

```
00001 #include "interfaces/IHittable.hpp"
00002
00003 #ifndef __ROTATE_Y_HPP__
00004     #define __ROTATE_Y_HPP__
00005
00006 namespace Raytracer::Effects
00007 {
00008     class RotateY : public Interfaces::IHittable {
00009       private:
00010         std::shared_ptr<Interfaces::IHittable> _object;
00011         double _sinTheta;
00012         double _cosTheta;
00013         Utils::AxisAlignedBBox _bbox;
00014
00015       public:
00016         RotateY(std::shared_ptr<Interfaces::IHittable> object, double angle);
00017         bool hit(const Core::Ray &ray, Utils::Interval interval,
00018             Core::Payload &payload) const override;
00019         Utils::AxisAlignedBBox boundingBox() const override;
00020     };
00021
00022 } // namespace Raytracer::Effects
00023
00024 #endif /* __ROTATE_Y_HPP__ */
```

8.15 RotateZ.hpp

```
00001 #include "interfaces/IHittable.hpp"
00002
00003 #ifndef __ROTATE_Z_HPP__
00004     #define __ROTATE_Z_HPP__
00005
00006 namespace Raytracer::Effects
00007 {
00008     class RotateZ : public Interfaces::IHittable {
00009       private:
00010         std::shared_ptr<Interfaces::IHittable> _object;
00011         double _sinTheta;
00012         double _cosTheta;
00013         Utils::AxisAlignedBBox _bbox;
00014
00015       public:
00016         RotateZ(std::shared_ptr<Interfaces::IHittable> object, double angle);
00017         bool hit(const Core::Ray &ray, Utils::Interval interval,
00018             Core::Payload &payload) const override;
00019         Utils::AxisAlignedBBox boundingBox() const override;
00020     };
00021
00022 } // namespace Raytracer::Effects
00023
00024 #endif /* __ROTATE_Z_HPP__ */
```

8.16 Smoke.hpp

```
00001 #include "interfaces/IHittable.hpp"
00002 #include "interfaces/ITexture.hpp"
00003
00004 #ifndef __SMOKE_HPP__
00005     #define __SMOKE_HPP__
00006
00007 namespace Raytracer::Effects
00008 {
00009     class Smoke : public Interfaces::IHittable {
00010       private:
00011         std::shared_ptr<Interfaces::IHittable> _boundary;
00012         std::shared_ptr<Interfaces::IMaterial> _phaseFunction;
00013         double _density;
00014
00015       public:
00016         Smoke(std::shared_ptr<Interfaces::IHittable> boundary, double density,
00017             std::shared_ptr<Interfaces::ITexture> texture);
00018         Smoke(std::shared_ptr<Interfaces::IHittable> boundary, double density,
00019             const Utils::Color &albedo);
00020         bool hit(const Core::Ray &ray, Utils::Interval interval,
00021             Core::Payload &payload) const override;
00022         Utils::AxisAlignedBBox boundingBox() const override;
00023     };
00024 } // namespace Raytracer::Effects
00025
00026 #endif /* __SMOKE_HPP__ */
```

8.17 Translate.hpp

```
00001 #include "interfaces/IHittable.hpp"
00002 #include "utils/AxisAlignedBBox.hpp"
00003 #include "utils/VecN.hpp"
00004
00005 #ifndef __TRANSLATE_HPP__
00006     #define __TRANSLATE_HPP__
00007
00008 namespace Raytracer::Effects
00009 {
00010     class Translate : public Interfaces::IHittable {
00011       private:
00012         std::shared_ptr<Interfaces::IHittable> _object;
00013         Utils::Vec3 _offset;
00014         Utils::AxisAlignedBBox _bbox;
00015
00016       public:
00017         Translate(std::shared_ptr<Interfaces::IHittable> object,
00018             const Utils::Vec3 &offset);
00019         bool hit(const Core::Ray &ray, Utils::Interval interval,
00020             Core::Payload &payload) const override;
00021         Utils::AxisAlignedBBox boundingBox() const override;
```

```
00022     };
00023 } // namespace Raytracer::Effects
00024
00025 #endif /* __TRANSLATE_HPP__ */
```

8.18 Argument.hpp

```
00001 #include <string>
00002 #include "exceptions/Base.hpp"
00003
00004 #ifndef __ARGUMENT_EXCEPTION_HPP__
00005     #define __ARGUMENT_EXCEPTION_HPP__
00006
00007 namespace Raytracer::Exceptions
00008 {
00009     class ArgumentException : public Base {
00010       public:
00011         ArgumentException(const std::string &message)
00012             : Base("Argument error: " + message)
00013         {
00014         }
00015         virtual ~ArgumentException() = default;
00016     };
00017 } // namespace Raytracer::Exceptions
00018
00019 #endif /* __ARGUMENT_EXCEPTION_HPP__ */
```

8.19 Base.hpp

```
00001 #include <exception>
00002 #include <string>
00003
00004 #ifndef __BASE_EXCEPTION_HPP__
00005     #define __BASE_EXCEPTION_HPP__
00006
00007 namespace Raytracer::Exceptions
00008 {
00009     class Base : public std::exception {
00010       public:
00011         Base(const std::string &message) : _message(message)
00012         {
00013         }
00014         virtual ~Base() = default;
00015
00016         virtual const char *what() const noexcept override
00017         {
00018             return _message.c_str();
00019         }
00020
00021       private:
00022         std::string _message;
00023     };
00024 } // namespace Raytracer::Exceptions
00025
00026 #endif /* __BASE_EXCEPTION_HPP__ */
```

8.20 Cyclic.hpp

```
00001 #include <string>
00002 #include "exceptions/Base.hpp"
00003
00004 #ifndef __CYCLIC_EXCEPTION_HPP__
00005     #define __CYCLIC_EXCEPTION_HPP__
00006
00007 namespace Raytracer::Exceptions
00008 {
00009     class CyclicException : public Base {
00010       public:
00011         CyclicException(const std::string &message)
00012             : Base("Cyclic error: " + message)
00013         {
00014         }
00015         virtual ~CyclicException() = default;
00016     };
00017 } // namespace Raytracer::Exceptions
00018
00019 #endif /* __CYCLIC_EXCEPTION_HPP__ */
```

8.21 File.hpp

```
00001 #include <string>
00002 #include "exceptions/Base.hpp"
00003
00004 #ifndef __FILE_EXCEPTION_HPP__
00005     #define __FILE_EXCEPTION_HPP__
00006
00007 namespace Raytracer::Exceptions
00008 {
00009     class FileException : public Base {
00010       public:
00011         FileException(const std::string &message)
00012             : Base("File error: " + message)
00013         {
00014         }
00015         virtual ~FileException() = default;
00016     };
00017 } // namespace Raytracer::Exceptions
00018
00019 #endif /* __FILE_EXCEPTION_HPP__ */
```

8.22 Missing.hpp

```
00001 #include <string>
00002 #include "exceptions/Base.hpp"
00003
00004 #ifndef __MISSING_EXCEPTION_HPP__
00005     #define __MISSING_EXCEPTION_HPP__
00006
00007 namespace Raytracer::Exceptions
00008 {
00009     class MissingException : public Base {
00010       public:
00011         MissingException(const std::string &message)
00012             : Base("Missing error: " + message)
00013         {
00014         }
00015         virtual ~MissingException() = default;
00016     };
00017 } // namespace Raytracer::Exceptions
00018
00019 #endif /* __MISSING_EXCEPTION_HPP__ */
```

8.23 Parse.hpp

```
00001 #include <string>
00002 #include "exceptions/Base.hpp"
00003
00004 #ifndef __PARSE_EXCEPTION_HPP__
00005     #define __PARSE_EXCEPTION_HPP__
00006
00007 namespace Raytracer::Exceptions
00008 {
00009     class ParseException : public Base {
00010       public:
00011         ParseException(const std::string &message)
00012             : Base("Parse error: " + message)
00013         {
00014         }
00015         virtual ~ParseException() = default;
00016     };
00017 } // namespace Raytracer::Exceptions
00018
00019 #endif /* __PARSE_EXCEPTION_HPP__ */
```

8.24 Range.hpp

```
00001 #include <string>
00002 #include "exceptions/Base.hpp"
00003
00004 #ifndef __RANGE_EXCEPTION_HPP__
00005     #define __RANGE_EXCEPTION_HPP__
00006
00007 namespace Raytracer::Exceptions
```

```
00008 {
00009     class RangeException : public Base {
00010       public:
00011         RangeException(const std::string &message)
00012             : Base("Range error: " + message)
00013         {
00014         }
00015         virtual ~RangeException() = default;
00016     };
00017 } // namespace Raytracer::Exceptions
00018
00019 #endif /* __RANGE_EXCEPTION_HPP__ */
```

8.25 IArguments.hpp

```
00001 #include "arguments/Kinds.hpp"
00002
00003 #ifndef __IARGUMENTS_HPP__
00004     #define __IARGUMENTS_HPP__
00005
00006 namespace Raytracer::Interfaces
00007 {
00008     class IArguments {
00009       public:
00010         virtual ~IArguments() = default;
00011         virtual Arguments::ArgumentKind kind() const = 0;
00012     };
00013 } // namespace Raytracer::Interfaces
00014
00015 #endif /* __IARGUMENTS_HPP__ */
```

8.26 IHittable.hpp

```
00001 #include "core/Payload.hpp"
00002 #include "core/Ray.hpp"
00003 #include "utils/AxisAlignedBBox.hpp"
00004 #include "utils/Interval.hpp"
00005
00006 #ifndef __IHITTABLE_HPP__
00007     #define __IHITTABLE_HPP__
00008
00009 namespace Raytracer::Interfaces
00010 {
00011     class IHittable {
00012       public:
00013         virtual ~IHittable() = default;
00014         virtual bool hit(const Core::Ray &ray, Utils::Interval interval,
00015             Core::Payload &payload) const = 0;
00016         virtual Utils::AxisAlignedBBox boundingBox() const = 0;
00017     };
00018 } // namespace Raytracer::Interfaces
00019
00020 #endif /* __IHITTABLE_HPP__ */
```

8.27 IMaterial.hpp

```
00001 #include "core/Ray.hpp"
00002 #include "utils/VecN.hpp"
00003
00004 #ifndef __IMATERIAL_HPP__
00005     #define __IMATERIAL_HPP__
00006
00007 namespace Raytracer::Core
00008 {
00009     class Payload;
00010 } // namespace Raytracer::Core
00011
00012 namespace Raytracer::Interfaces
00013 {
00014     class IMaterial {
00015       public:
00016         virtual ~IMaterial() = default;
00017         virtual Utils::Color emitted(
00018             double u, double v, const Utils::Point3 &point) const = 0;
00019         virtual bool scatter(const Core::Ray &ray,
```

```
00020             const Core::Payload &payload, Utils::Color &attenuation,
00021             Core::Ray &scattered) const = 0;
00022     };
00023 } // namespace Raytracer::Interfaces
00024
00025 #endif /* __IMATERIAL_HPP__ */
```

8.28 ITexture.hpp

```
00001 #include "utils/Color.hpp"
00002
00003 #ifndef __ITEXTURE_HPP__
00004     #define __ITEXTURE_HPP__
00005
00006 namespace Raytracer::Interfaces
00007 {
00008     class ITexture {
00009       public:
00010         virtual ~ITexture() = default;
00011         virtual Utils::Color value(
00012             double u, double v, const Utils::Point3 &point) const = 0;
00013     };
00014 } // namespace Raytracer::Interfaces
00015
00016 #endif /* __ITEXTURE_HPP__ */
```

8.29 Dielectric.hpp

```
00001 #include "interfaces/IMaterial.hpp"
00002
00003 #ifndef __DIELECTRIC_HPP__
00004     #define __DIELECTRIC_HPP__
00005
00006 namespace Raytracer::Materials
00007 {
00008     class Dielectric : public Interfaces::IMaterial {
00009       private:
00010         double _refractionIndex;
00011         Utils::Color _albedo = Utils::Color(1.0, 1.0, 1.0);
00012
00013       public:
00014         Dielectric(double refractionIndex);
00015         Dielectric(double refractionIndex, const Utils::Color &albedo);
00016         bool scatter(const Core::Ray &ray, const Core::Payload &payload,
00017             Utils::Color &attenuation, Core::Ray &scattered) const override;
00018         Utils::Color emitted(
00019             double u, double v, const Utils::Point3 &point) const override;
00020         static double reflectance(double cosine, double index);
00021     };
00022 } // namespace Raytracer::Materials
00023
00024 #endif /* __DIELECTRIC_HPP__ */
```

8.30 DiffuseLight.hpp

```
00001 #include <memory>
00002 #include "interfaces/IMaterial.hpp"
00003 #include "interfaces/ITexture.hpp"
00004
00005 #ifndef __DIFFUSELIGHT_HPP__
00006     #define __DIFFUSELIGHT_HPP__
00007
00008 namespace Raytracer::Materials
00009 {
00010     class DiffuseLight : public Interfaces::IMaterial {
00011       private:
00012         std::shared_ptr<Interfaces::ITexture> _texture;
00013
00014       public:
00015         DiffuseLight(std::shared_ptr<Interfaces::ITexture> texture);
00016         DiffuseLight(const Utils::Color &color);
00017         bool scatter(const Core::Ray &ray, const Core::Payload &payload,
00018             Utils::Color &attenuation, Core::Ray &scattered) const override;
00019         Utils::Color emitted(
00020             double u, double v, const Utils::Point3 &point) const override;
```

```
00021     };
00022 } // namespace Raytracer::Materials
00023
00024 #endif /* __DIFFUSELIGHT_HPP__ */
```

8.31 Isotropic.hpp

```
00001 #include <memory>
00002 #include "interfaces/IMaterial.hpp"
00003 #include "interfaces/ITexture.hpp"
00004
00005 #ifndef __ISOTROPIC_HPP__
00006     #define __ISOTROPIC_HPP__
00007
00008 namespace Raytracer::Materials
00009 {
00010     class Isotropic : public Interfaces::IMaterial {
00011       private:
00012         std::shared_ptr<Interfaces::ITexture> _texture;
00013
00014       public:
00015         Isotropic(std::shared_ptr<Interfaces::ITexture> texture);
00016         Isotropic(const Utils::Color &color);
00017         bool scatter(const Core::Ray &ray, const Core::Payload &payload,
00018             Utils::Color &attenuation, Core::Ray &scattered) const override;
00019         Utils::Color emitted(
00020             double u, double v, const Utils::Point3 &point) const override;
00021     };
00022 } // namespace Raytracer::Materials
00023
00024 #endif /* __ISOTROPIC_HPP__ */
```

8.32 Lambertian.hpp

```
00001 #include <memory>
00002 #include "interfaces/IMaterial.hpp"
00003 #include "interfaces/ITexture.hpp"
00004
00005 #ifndef __LAMBERTIAN_HPP__
00006     #define __LAMBERTIAN_HPP__
00007
00008 namespace Raytracer::Materials
00009 {
00010     class Lambertian : public Interfaces::IMaterial {
00011       private:
00012         std::shared_ptr<Interfaces::ITexture> _texture;
00013
00014       public:
00015         Lambertian(const Utils::Color &albedo);
00016         Lambertian(std::shared_ptr<Interfaces::ITexture> texture);
00017         bool scatter(const Core::Ray &ray, const Core::Payload &payload,
00018             Utils::Color &attenuation, Core::Ray &scattered) const override;
00019         Utils::Color emitted(
00020             double u, double v, const Utils::Point3 &point) const override;
00021     };
00022 } // namespace Raytracer::Materials
00023
00024 #endif /* __LAMBERTIAN_HPP__ */
```

8.33 Metal.hpp

```
00001 #include "interfaces/IMaterial.hpp"
00002
00003 #ifndef __METAL_HPP__
00004     #define __METAL_HPP__
00005
00006 namespace Raytracer::Materials
00007 {
00008     class Metal : public Interfaces::IMaterial {
00009       private:
00010         Utils::Color _albedo;
00011         double _fuzz;
00012
00013       public:
00014         Metal(const Utils::Color &albedo, double fuzz);
```

```
00015         bool scatter(const Core::Ray &ray, const Core::Payload &payload,
00016             Utils::Color &attenuation, Core::Ray &scattered) const override;
00017         Utils::Color emitted(
00018             double u, double v, const Utils::Point3 &point) const override;
00019     };
00020 } // namespace Raytracer::Materials
00021
00022 #endif /* __METAL_HPP__ */
```

8.34 Cone.hpp

```
00001 #include <memory>
00002 #include "interfaces/IHittable.hpp"
00003
00004 #ifndef __CONE_HPP__
00005     #define __CONE_HPP__
00006
00007 namespace Raytracer::Shapes
00008 {
00009     class Cone : public Interfaces::IHittable {
00010       private:
00011         Utils::Point3 _center;
00012         double _radius;
00013         double _height;
00014         std::shared_ptr<Interfaces::IMaterial> _material;
00015         Utils::AxisAlignedBBox _bbox;
00016
00017       public:
00018         Cone() = default;
00019         Cone(const Utils::Point3 &center, double radius, double height,
00020             std::shared_ptr<Interfaces::IMaterial> material);
00021         virtual bool hit(const Core::Ray &ray, Utils::Interval interval,
00022             Core::Payload &payload) const override;
00023         virtual Utils::AxisAlignedBBox boundingBox() const override;
00024     };
00025 } // namespace Raytracer::Shapes
00026
00027 #endif /* __CONE_HPP__ */
```

8.35 Cylinder.hpp

```
00001 #include <memory>
00002 #include "interfaces/IHittable.hpp"
00003 #include "interfaces/IMaterial.hpp"
00004
00005 #ifndef __CYLINDER_HPP__
00006     #define __CYLINDER_HPP__
00007
00008 namespace Raytracer::Shapes
00009 {
00010     class Cylinder : public Interfaces::IHittable {
00011       private:
00012         Utils::Point3 _center;
00013         double _radius;
00014         double _height;
00015         std::shared_ptr<Interfaces::IMaterial> _material;
00016         Utils::AxisAlignedBBox _bbox;
00017
00018       public:
00019         Cylinder() = default;
00020         Cylinder(const Utils::Point3 &center, double radius, double height,
00021             std::shared_ptr<Interfaces::IMaterial> material);
00022         virtual bool hit(const Core::Ray &ray, Utils::Interval interval,
00023             Core::Payload &payload) const override;
00024         virtual Utils::AxisAlignedBBox boundingBox() const override;
00025     };
00026 } // namespace Raytracer::Shapes
00027
00028 #endif /* __CYLINDER_HPP__ */
```

8.36 Plane.hpp

```
00001 #include <memory>
00002 #include "interfaces/IHittable.hpp"
00003 #include "utils/VecN.hpp"
```

```
00004
00005 #ifndef __PLANE_HPP__
00006     #define __PLANE_HPP__
00007
00008 namespace Raytracer::Shapes
00009 {
00010     class Plane : public Interfaces::IHittable {
00011       private:
00012         Utils::Point3 _point;
00013         Utils::Vec3 _normal;
00014         std::shared_ptr<Interfaces::IMaterial> _material;
00015         Utils::AxisAlignedBBox _bbox;
00016
00017       public:
00018         Plane(const Utils::Point3 &point, const Utils::Vec3 &normal,
00019             std::shared_ptr<Interfaces::IMaterial> material);
00020         bool hit(const Core::Ray &ray, Utils::Interval interval,
00021             Core::Payload &payload) const override;
00022         Utils::AxisAlignedBBox boundingBox() const override;
00023     };
00024 } // namespace Raytracer::Shapes
00025 #endif /* __PLANE_HPP__ */
```

8.37 Quad.hpp

```
00001 #include <memory>
00002 #include "core/Scene.hpp"
00003 #include "interfaces/IHittable.hpp"
00004
00005 #ifndef __QUAD_HPP__
00006     #define __QUAD_HPP__
00007
00008 namespace Raytracer::Shapes
00009 {
00010     class Quad : public Interfaces::IHittable {
00011       private:
00012         Utils::Point3 _Q;
00013         Utils::Vec3 _u;
00014         Utils::Vec3 _v;
00015         Utils::Vec3 _w;
00016         std::shared_ptr<Interfaces::IMaterial> _material;
00017         Utils::AxisAlignedBBox _bbox;
00018         Utils::Vec3 _normal;
00019         double _D;
00020
00021       public:
00022         Quad(const Utils::Point3 &Q, const Utils::Vec3 &u,
00023             const Utils::Vec3 &v,
00024             std::shared_ptr<Interfaces::IMaterial> material);
00025         bool hit(const Core::Ray &ray, Utils::Interval interval,
00026             Core::Payload &payload) const override;
00027         Utils::AxisAlignedBBox boundingBox() const override;
00028         virtual void setBBox();
00029         virtual bool isInterior(
00030             double a, double b, Core::Payload &payload) const;
00031     };
00032
00033     std::shared_ptr<Core::Scene> box(const Utils::Point3 &a,
00034         const Utils::Point3 &b,
00035         std::shared_ptr<Interfaces::IMaterial> material);
00036 } // namespace Raytracer::Shapes
00037
00038 #endif /* __QUAD_HPP__ */
```

8.38 Sphere.hpp

```
00001 #include <memory>
00002 #include "interfaces/IHittable.hpp"
00003 #include "utils/VecN.hpp"
00004
00005 #ifndef __SPHERE_HPP__
00006     #define __SPHERE_HPP__
00007
00008 namespace Raytracer::Shapes
00009 {
00010     class Sphere : public Interfaces::IHittable {
00011       private:
00012         Utils::Point3 _center;
00013         double _radius;
```

```
00014         std::shared_ptr<Interfaces::IMaterial> _material;
00015         bool _isMoving = false;
00016         Utils::Vec3 _centerVec;
00017         Utils::AxisAlignedBBox _bbox;
00018
00019       public:
00020         Sphere(const Utils::Point3 &center, double radius,
00021             std::shared_ptr<Interfaces::IMaterial> material);
00022         Sphere(const Utils::Point3 &centerOne, const Utils::Point3 &centerTwo,
00023             double radius, std::shared_ptr<Interfaces::IMaterial> material);
00024         bool hit(const Core::Ray &ray, Utils::Interval interval,
00025             Core::Payload &hit) const override;
00026         Utils::Point3 sphereCenter(double time) const;
00027         Utils::AxisAlignedBBox boundingBox() const override;
00028         static void getSphereUV(
00029             const Utils::Point3 &point, double &u, double &v);
00030     };
00031 } // namespace Raytracer::Shapes
00032
00033 #endif /* __SPHERE_HPP__ */
```

8.39 Checker.hpp

```
00001 #include <memory>
00002 #include "interfaces/ITexture.hpp"
00003
00004 #ifndef __CHECKER_HPP__
00005     #define __CHECKER_HPP__
00006
00007 namespace Raytracer::Textures
00008 {
00009     class Checker : public Interfaces::ITexture {
00010       private:
00011         std::shared_ptr<Interfaces::ITexture> _odd;
00012         std::shared_ptr<Interfaces::ITexture> _even;
00013         double _scale;
00014
00015       public:
00016         Checker(double scale, std::shared_ptr<Interfaces::ITexture> even,
00017             std::shared_ptr<Interfaces::ITexture> odd);
00018         Checker(double scale, const Utils::Color &a, const Utils::Color &b);
00019         Utils::Color value(
00020             double u, double v, const Utils::Point3 &point) const override;
00021     };
00022 } // namespace Raytracer::Textures
00023
00024 #endif /* __CHECKER_HPP__ */
```

8.40 Image.hpp

```
00001 #include "interfaces/ITexture.hpp"
00002 #include "utils/ImageHelper.hpp"
00003
00004 #ifndef __IMAGE_HPP__
00005     #define __IMAGE_HPP__
00006
00007 namespace Raytracer::Textures
00008 {
00009     class Image : public Interfaces::ITexture {
00010       private:
00011         Utils::ImageHelper _helper;
00012
00013       public:
00014         Image(std::string filename);
00015         Utils::Color value(
00016             double u, double v, const Utils::Point3 &point) const override;
00017     };
00018 } // namespace Raytracer::Textures
00019
00020 #endif /* __IMAGE_HPP__ */
```

8.41 Noise.hpp

```
00001 #include "interfaces/ITexture.hpp"
00002 #include "utils/Perlin.hpp"
```

```
00003
00004 #ifndef __NOISE_HPP__
00005     #define __NOISE_HPP__
00006
00007 namespace Raytracer::Textures
00008 {
00009     class Noise : public Interfaces::ITexture {
00010       private:
00011         double _scale;
00012         Utils::Perlin _perlin;
00013
00014       public:
00015         Noise(double scale);
00016         Utils::Color value(
00017             double u, double v, const Utils::Point3 &point) const override;
00018     };
00019 } // namespace Raytracer::Textures
00020
00021 #endif /* __NOISE_HPP__ */
```

8.42 SolidColor.hpp

```
00001 #include "interfaces/ITexture.hpp"
00002
00003 #ifndef __SOLIDCOLOR_HPP__
00004     #define __SOLIDCOLOR_HPP__
00005
00006 namespace Raytracer::Textures
00007 {
00008     class SolidColor : public Interfaces::ITexture {
00009       private:
00010         Utils::Color _albedo;
00011
00012       public:
00013         SolidColor(const Utils::Color &albedo);
00014         SolidColor(double red, double green, double blue);
00015         Utils::Color value(
00016             double u, double v, const Utils::Point3 &point) const override;
00017     };
00018 } // namespace Raytracer::Textures
00019
00020 #endif /* __SOLIDCOLOR_HPP__ */
```

8.43 AxisAlignedBBox.hpp

```
00001 #include "core/Ray.hpp"
00002 #include "utils/Interval.hpp"
00003 #include "utils/VecN.hpp"
00004
00005 #ifndef __AXIS_ALIGNED_BBOX_HPP__
00006     #define __AXIS_ALIGNED_BBOX_HPP__
00007
00008 namespace Raytracer::Utils
00009 {
00010     class AxisAlignedBBox {
00011       private:
00012         Interval _x;
00013         Interval _y;
00014         Interval _z;
00015
00016       public:
00017         AxisAlignedBBox() = default;
00018         AxisAlignedBBox(
00019             const Interval &x, const Interval &y, const Interval &z);
00020         AxisAlignedBBox(const Point3 &a, const Point3 &b);
00021         AxisAlignedBBox(const AxisAlignedBBox &a, const AxisAlignedBBox &b);
00022         const Interval &axisInterval(int n) const;
00023         bool hit(const Core::Ray &ray, Interval interval) const;
00024         int longestAxis() const;
00025         void padToMinimum();
00026         static const AxisAlignedBBox Empty;
00027         static const AxisAlignedBBox Universe;
00028         GET_SET(Interval, x)
00029         GET_SET(Interval, y)
00030         GET_SET(Interval, z)
00031     };
00032
00033     Utils::AxisAlignedBBox operator+(
00034         const Utils::AxisAlignedBBox &value, Utils::Vec3 offset);
```

```
00035     Utils::AxisAlignedBBox operator+(
00036         Utils::Vec3 offset, const Utils::AxisAlignedBBox &value);
00037 } // namespace Raytracer::Utils
00038
00039 #endif /* __AXIS_ALIGNED_BBOX_HPP__ */
```

8.44 BVHNode.hpp

```
00001 #include "core/Scene.hpp"
00002 #include "interfaces/IHittable.hpp"
00003
00004 #ifndef __BVH_NODE_HPP__
00005     #define __BVH_NODE_HPP__
00006
00007 namespace Raytracer::Utils
00008 {
00009     class BVHNode : public Interfaces::IHittable {
00010       private:
00011         std::shared_ptr<Interfaces::IHittable> _left;
00012         std::shared_ptr<Interfaces::IHittable> _right;
00013         AxisAlignedBBox _bbox;
00014
00015       public:
00016         BVHNode() = default;
00017         BVHNode(Core::Scene list);
00018         BVHNode(std::vector<std::shared_ptr<Interfaces::IHittable» &objects,
00019             size_t start, size_t end);
00020         bool hit(const Core::Ray &ray, Interval interval,
00021             Core::Payload &payload) const override;
00022         AxisAlignedBBox boundingBox() const override;
00023         static bool boxCompare(const std::shared_ptr<Interfaces::IHittable> &a,
00024             const std::shared_ptr<Interfaces::IHittable> &b, int axis);
00025         static bool boxXCompare(
00026             const std::shared_ptr<Interfaces::IHittable> &a,
00027             const std::shared_ptr<Interfaces::IHittable> &b);
00028         static bool boxYCompare(
00029             const std::shared_ptr<Interfaces::IHittable> &a,
00030             const std::shared_ptr<Interfaces::IHittable> &b);
00031         static bool boxZCompare(
00032             const std::shared_ptr<Interfaces::IHittable> &a,
00033             const std::shared_ptr<Interfaces::IHittable> &b);
00034         GET_SET(std::shared_ptr<Interfaces::IHittable>, left)
00035         GET_SET(std::shared_ptr<Interfaces::IHittable>, right)
00036     };
00037 } // namespace Raytracer::Utils
00038
00039 #endif /* __BVH_NODE_HPP__ */
```

8.45 Color.hpp

```
00001 #include "VecN.hpp"
00002
00003 #ifndef __COLOR_HPP__
00004     #define __COLOR_HPP__
00005
00006 namespace Raytracer::Utils
00007 {
00008     double linearToGamma(double linear);
00009     void writeColor(std::ostream &out, const Color &pixelColor);
00010 } // namespace Raytracer::Utils
00011
00012 #endif /* __COLOR_HPP__ */
```

8.46 ImageHelper.hpp

```
00001 #include <string>
00002 #include <vector>
00003 #include "Common.hpp"
00004
00005 #ifndef __IMAGE_HELPER_HPP__
00006     #define __IMAGE_HELPER_HPP__
00007
00008 namespace Raytracer::Utils
00009 {
00010     class ImageHelper {
```

```
00011       private:
00012         int _width = 0;
00013         int _height = 0;
00014         std::vector<unsigned char> data;
00015
00016       public:
00017         ImageHelper() = default;
00018         ImageHelper(const char *filename);
00019         bool load(const std::string &filename);
00020         const unsigned char *pixelData(int x, int y) const;
00021         GET_SET(int, width);
00022         GET_SET(int, height);
00023     };
00024 } // namespace Raytracer::Utils
00025
00026 #endif /* __IMAGE_HELPER_HPP__ */
```

8.47 Interval.hpp

```
00001 #include <limits>
00002 #include "Common.hpp"
00003
00004 #ifndef __INTERVAL_HPP__
00005     #define __INTERVAL_HPP__
00006
00007 namespace Raytracer::Utils
00008 {
00009     class Interval {
00010       private:
00011         double _min = +std::numeric_limits<double>::infinity();
00012         double _max = -std::numeric_limits<double>::infinity();
00013
00014       public:
00015         Interval() = default;
00016         Interval(double min, double max);
00017         Interval(const Interval &a, const Interval &b);
00018         double size() const;
00019         bool contains(double x) const;
00020         bool surrounds(double x) const;
00021         double clamp(double x) const;
00022         Interval expand(double x) const;
00023         static const Interval Empty;
00024         static const Interval Universe;
00025         GET_SET(double, min)
00026         GET_SET(double, max)
00027     };
00028
00029     Utils::Interval operator+(const Utils::Interval &value, double offset);
00030     Utils::Interval operator+(double offset, const Utils::Interval &value);
00031 } // namespace Raytracer::Utils
00032
00033 #endif /* __INTERVAL_HPP__ */
```

8.48 Perlin.hpp

```
00001 #include "utils/Color.hpp"
00002 #include "utils/VecN.hpp"
00003
00004 #ifndef __PERLIN_HPP__
00005     #define __PERLIN_HPP__
00006
00007 namespace Raytracer::Utils
00008 {
00009     class Perlin {
00010       private:
00011         static constexpr int pointCount = 256;
00012         Utils::Vec3 *_randVec;
00013         int *_permX;
00014         int *_permY;
00015         int *_permZ;
00016
00017       public:
00018         Perlin();
00019         ~Perlin();
00020         double noise(const Utils::Point3 &point) const;
00021         double turbulence(const Utils::Point3 &point, int depth = 7) const;
00022         static int *perlinGeneratePerm();
00023         static void permute(int *perm, int n);
00024         static double perlinInterp(
```

```
00025             const Utils::Vec3 c[2][2][2], double u, double v, double w);
00026     };
00027 } // namespace Raytracer::Utils
00028
00029 #endif /* __PERLIN_HPP__ */
```

8.49 VecN.hpp

```
00001 #include <cmath>
00002 #include <cstddef>
00003 #include <iostream>
00004 #include "exceptions/Range.hpp"
00005 #include <type_traits>
00006
00007 #ifndef __VEC_N_HPP__
00008     #define __VEC_N_HPP__
00009
00010 namespace Raytracer::Utils
00011 {
00012     inline double degreesToRadians(double degrees)
00013     {
00014         return degrees * M_PI / 180.0;
00015     }
00016
00017     inline double randomDouble()
00018     {
00019         return rand() / (RAND_MAX + 1.0);
00020     }
00021
00022     inline double randomDouble(double min, double max)
00023     {
00024         return min + (max - min) * randomDouble();
00025     }
00026
00027     inline int randomInt(int min, int max)
00028     {
00029         return static_cast<int>(randomDouble(min, max + 1));
00030     }
00031
00032     template <typename T>
00033     concept isNumerical = requires(T) { std::is_arithmetic_v<T>; };
00034
00035     template <typename T>
00036     concept isPositive = requires(T t) { t > 0; };
00037
00038     template <isNumerical T, std::size_t N>
00039         requires isPositive<T> && (N > 1)
00040     class VecN {
00041       public:
00042         T e[N];
00043
00044         VecN() : e{0, 0, 0}
00045         {
00046         }
00047
00048         VecN(T e0, T e1, T e2)
00049         {
00050             static_assert(
00051                 N == 3, "VecN(T e0, T e1, T e2) is only valid for N == 3");
00052             e[0] = e0;
00053             e[1] = e1;
00054             e[2] = e2;
00055         }
00056
00057         T x() const
00058         {
00059             return e[0];
00060         }
00061
00062         T y() const
00063         {
00064             static_assert(N == 3, "y() is only valid for >= VecN<2>");
00065             return e[1];
00066         }
00067
00068         T z() const
00069         {
00070             static_assert(N == 3, "z() is only valid for VecN<3>");
00071             return e[2];
00072         }
00073
00074         VecN operator-() const
00075         {
```

```
00076             VecN v;
00077             for (std::size_t i = 0; i < N; i++) {
00078                 v.e[i] = -e[i];
00079             }
00080
00081             return v;
00082         }
00083
00084         T operator[](std::size_t i) const
00085         {
00086             if (i >= N) {
00087                 throw Exceptions::RangeException("Index out of bounds");
00088             }
00089             return e[i];
00090         }
00091
00092         T &operator[](std::size_t i)
00093         {
00094             if (i >= N) {
00095                 throw Exceptions::RangeException("Index out of bounds");
00096             }
00097             return e[i];
00098         }
00099
00100        VecN &operator+=(const VecN &v)
00101        {
00102            for (std::size_t i = 0; i < N; i++) {
00103                e[i] += v.e[i];
00104            }
00105
00106            return *this;
00107        }
00108
00109        VecN &operator*=(double t)
00110        {
00111            for (std::size_t i = 0; i < N; i++) {
00112                e[i] *= t;
00113            }
00114
00115            return *this;
00116        }
00117
00118        VecN &operator/=(double t)
00119        {
00120            return *this *= 1 / t;
00121        }
00122
00123        double length() const
00124        {
00125            return std::sqrt(lengthSquared());
00126        }
00127
00128        double lengthSquared() const
00129        {
00130            double sum = 0;
00131            for (std::size_t i = 0; i < N; i++) {
00132                sum += e[i] * e[i];
00133            }
00134
00135            return sum;
00136        }
00137
00138        bool nearZero() const
00139        {
00140            static constexpr double s = 1e-8;
00141            for (std::size_t i = 0; i < N; i++) {
00142                if (std::fabs(e[i]) > s) {
00143                    return false;
00144                }
00145            }
00146
00147            return true;
00148        }
00149
00150        static VecN random()
00151        {
00152            VecN result;
00153            for (std::size_t i = 0; i < N; i++) {
00154                result[i] = randomDouble();
00155            }
00156
00157            return result;
00158        }
00159
00160        static VecN random(double min, double max)
00161        {
00162            VecN result;
```

```
00163             for (std::size_t i = 0; i < N; i++) {
00164                 result[i] = randomDouble(min, max);
00165             }
00166
00167             return result;
00168         }
00169
00170         VecN &normalize()
00171         {
00172             return *this /= length();
00173         }
00174     };
00175
00176     using Vec3 = VecN<double, 3>;
00177     using Point3 = Vec3;
00178     using Color = Vec3;
00179
00180     template <typename T, std::size_t N>
00181     std::ostream &operator<<(std::ostream &out, const VecN<T, N> &v)
00182     {
00183         for (std::size_t i = 0; i < N; i++) {
00184             out << v[i];
00185             if (i != N - 1) {
00186                 out << " ";
00187             }
00188         }
00189         return out;
00190     }
00191
00192     template <typename T, std::size_t N>
00193     VecN<T, N> operator+(const VecN<T, N> &u, const VecN<T, N> &v)
00194     {
00195         VecN<T, N> result;
00196         for (std::size_t i = 0; i < N; i++) {
00197             result.e[i] = u.e[i] + v.e[i];
00198         }
00199         return result;
00200     }
00201
00202     template <typename T, std::size_t N>
00203     VecN<T, N> operator-(const VecN<T, N> &u, const VecN<T, N> &v)
00204     {
00205         VecN<T, N> result;
00206         for (std::size_t i = 0; i < N; i++) {
00207             result.e[i] = u.e[i] - v.e[i];
00208         }
00209         return result;
00210     }
00211
00212     template <typename T, std::size_t N>
00213     VecN<T, N> operator*(const VecN<T, N> &u, const VecN<T, N> &v)
00214     {
00215         VecN<T, N> result;
00216         for (std::size_t i = 0; i < N; i++) {
00217             result.e[i] = u.e[i] * v.e[i];
00218         }
00219         return result;
00220     }
00221
00222     template <typename T, std::size_t N>
00223     VecN<T, N> operator*(double t, const VecN<T, N> &v)
00224     {
00225         VecN<T, N> result;
00226         for (std::size_t i = 0; i < N; i++) {
00227             result.e[i] = t * v.e[i];
00228         }
00229         return result;
00230     }
00231
00232     template <typename T, std::size_t N>
00233     VecN<T, N> operator*(const VecN<T, N> &v, double t)
00234     {
00235         return t * v;
00236     }
00237
00238     template <typename T, std::size_t N>
00239     VecN<T, N> operator/(const VecN<T, N> &v, double t)
00240     {
00241         return (1 / t) * v;
00242     }
00243
00244     template <typename T, std::size_t N>
00245     T dot(const VecN<T, N> &u, const VecN<T, N> &v)
00246     {
00247         T sum = 0;
00248         for (std::size_t i = 0; i < N; i++) {
00249             sum += u.e[i] * v.e[i];
```

```
00250         }
00251
00252         return sum;
00253     }
00254
00255     template <typename T, std::size_t N>
00256     VecN<T, N> cross(const VecN<T, N> &u, const VecN<T, N> &v)
00257     {
00258         static_assert(N == 3, "cross() is only valid for VecN<3>");
00259
00260         VecN<T, N> result;
00261         result[0] = u.e[1] * v.e[2] - u.e[2] * v.e[1];
00262         result[1] = u.e[2] * v.e[0] - u.e[0] * v.e[2];
00263         result[2] = u.e[0] * v.e[1] - u.e[1] * v.e[0];
00264
00265         return result;
00266     }
00267
00268     template <typename T, std::size_t N>
00269     VecN<T, N> unitVector(const VecN<T, N> &v)
00270     {
00271         return v / v.length();
00272     }
00273
00274     template <typename T, std::size_t N> VecN<T, N> randomInUnitSphere()
00275     {
00276         while (true) {
00277             VecN<T, N> vec = VecN<T, N>::random(-1, 1);
00278
00279             if (vec.lengthSquared() >= 1) {
00280                 continue;
00281             }
00282
00283             return vec;
00284         }
00285     }
00286
00287     template <typename T, std::size_t N> VecN<T, N> randomUnitVector()
00288     {
00289         return unitVector(randomInUnitSphere<T, N>());
00290     }
00291
00292     template <typename T, std::size_t N>
00293     VecN<T, N> randomInHemisphere(const VecN<T, N> &normal)
00294     {
00295         VecN<T, N> inUnitSphere = randomInUnitSphere<T, N>();
00296
00297         if (dot(inUnitSphere, normal) > 0.0) {
00298             return inUnitSphere;
00299         } else {
00300             return -inUnitSphere;
00301         }
00302     }
00303
00304     template <typename T, std::size_t N>
00305     VecN<T, N> reflect(const VecN<T, N> &v, const VecN<T, N> &n)
00306     {
00307         return v - 2 * dot(v, n) * n;
00308     }
00309
00310     template <typename T, std::size_t N>
00311     VecN<T, N> refract(
00312         const VecN<T, N> &uv, const VecN<T, N> &n, double etaiOverEtat)
00313     {
00314         double cosTheta = std::fmin(dot(-uv, n), 1.0);
00315         VecN<T, N> rayOutPerp = etaiOverEtat * (uv + cosTheta * n);
00316         VecN<T, N> rayOutParallel =
00317             -std::sqrt(std::fabs(1.0 - rayOutPerp.lengthSquared())) * n;
00318         return rayOutPerp + rayOutParallel;
00319     }
00320
00321     template <typename T, std::size_t N> VecN<T, N> randomInUnitDisk()
00322     {
00323         while (true) {
00324             VecN<T, N> p =
00325                 VecN<T, N>(randomDouble(-1, 1), randomDouble(-1, 1), 0);
00326             if (p.lengthSquared() >= 1) {
00327                 continue;
00328             }
00329
00330             return p;
00331         }
00332     }
00333 } // namespace Raytracer::Utils
00334
00335 #endif /* __VEC_N_HPP__ */
```

Index