Path Tracer

Generated by Doxygen 1.9.8

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Class Documentation

4.1 AABB Struct Reference

Struct representing an Axis Aligned Bounding Box.

`#include <bvh.hpp>`

Public Attributes

- Vector **min**
- Vector **max**

4.1.1 Detailed Description

Struct representing an Axis Aligned Bounding Box.

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

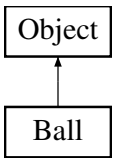
- objects/bvh.hpp

4.2 Ball Class Reference

Representation of a mathematical ball object in the scene.

`#include <ball.hpp>`

Inheritance diagram for Ball:



Public Member Functions

- **Ball** (Vector position, float radius, std::shared_ptr< Material > material)
- void collision (Ray &ray, Hit &rayHit, float &smallestDistance)

  Calculate whether a given ray collides with the ball.
- float **getRadius** () const
- void printInfo (std::ostream &out) const

  Print ball info to the desired output stream.

Public Member Functions inherited from Object

- **Object** (Vector position, std::shared_ptr< Material > material)
- Vector **getPosition** () const
- std::shared_ptr< Material > **getMaterial** () const

4.2.1 Detailed Description

Representation of a mathematical ball object in the scene.

4.2.2 Member Function Documentation

4.2.2.1 collision()

```
void Ball::collision (
            Ray & ray,
            Hit & rayHit,
            float & smallestDistance )  [inline], [virtual]
```

Calculate whether a given ray collides with the ball.

If the ray collides with the ball and the collision is closer than the current smallest distance, the "rayHit" data structure will be updated according to the collision.

Parameters

ray	ray whose collision will be checked
rayHit	address of a Hit data structure
smallestDistance	

Implements Object.

4.2.2.2 printInfo()

```
void Ball::printInfo (
            std::ostream & out ) const  [inline], [virtual]
```

Print ball info to the desired output stream.

Parameters

out	output stream

Returns

std::ostream& the output stream

Implements Object.

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

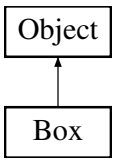
- objects/ball.hpp

4.3 Box Class Reference

Representation of a box (rectangular cuboid) object in the scene.

`#include <box.hpp>`

Inheritance diagram for Box:



Public Member Functions

- **Box** (Vector position, float width, float height, float depth, std::shared_ptr< Material > material)
- void collision (Ray &ray, Hit &rayHit, float &smallestDistance)

  Calculate whether a given ray collides with the box.
- float **getWidth** () const
- float **getHeight** () const
- float **getDepth** () const
- void rotate (float angle, Vector axis)

  Rotates the box around a given axis.
- void printInfo (std::ostream &out) const

  Print box info to the desired output stream.

Public Member Functions inherited from Object

- **Object** (Vector position, std::shared_ptr< Material > material)
- Vector **getPosition** () const
- std::shared_ptr< Material > **getMaterial** () const

4.3.1 Detailed Description

Representation of a box (rectangular cuboid) object in the scene.

The position represent the geometric center of the box.

4.3.2 Member Function Documentation

4.3.2.1 collision()

```
void Box::collision (
            Ray & ray,
            Hit & rayHit,
            float & smallestDistance )  [inline], [virtual]
```

Calculate whether a given ray collides with the box.

If the ray collides with the box and the collision is closer than the current smallest distance, the "rayHit" data structure will be updated according to the collision.

Parameters

ray	ray whose collision will be checked
rayHit	address of a Hit data structure
smallestDistance	

Implements Object.

4.3.2.2 printInfo()

```
void Box::printInfo (
            std::ostream & out ) const  [inline], [virtual]
```

Print box info to the desired output stream.

Parameters

out	output stream

Returns

std::ostream& the output stream

Implements Object.

4.3.2.3 rotate()

```
void Box::rotate (
            float angle,
            Vector axis )  [inline]
```

Rotates the box around a given axis.

The center of rotation is the box's center of mass.

Parameters

angle	rotation in radians
axis	axis of rotation (has to be normalized)

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

- objects/box.hpp

4.4 Button Class Reference

Implements a class for creating buttons for a SFML window.

`#include <button.hpp>`

Public Member Functions

- Button (std::string button_text, sf::Color textColor, int text_size, sf::Vector2f button_size, sf::Color button_↩ color)

  Constructor for the button.
- ∼**Button** ()=default

  Destroy the Button object.
- void setPos (sf::Vector2f pos)

  Sets the position of the button in the window.
- void setColor (sf::Color color)

  Sets the color of the button.
- void setFont (sf::Font &font)

  Sets the font of the text of the button.
- void draw (sf::RenderWindow &window)

  Renders the button on the specified SFML window.
- bool onButton (sf::RenderWindow &window)

  Checks if the user's mouse is on the button.

4.4.1 Detailed Description

Implements a class for creating buttons for a SFML window.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 Button()

```
Button::Button (
            std::string button_text,
            sf::Color textColor,
            int text_size,
            sf::Vector2f button_size,
            sf::Color button_color )  [inline]
```

Constructor for the button.

Parameters

button_text	The text inside the button
textColor	The color of the text inside the button
text_size	The size of the text inside the button
button_size	Size of the button
button_color	Color of the button

4.4.3 Member Function Documentation

4.4.3.1 draw()

```
void Button::draw (
            sf::RenderWindow & window )  [inline]
```

Renders the button on the specified SFML window.

Parameters

window	Window to be rendered on

4.4.3.2 onButton()

```
bool Button::onButton (
            sf::RenderWindow & window )  [inline]
```

Checks if the user's mouse is on the button.

Parameters

window	reference to the window the textbox is on

Returns

true value if mouse is on the button

false value if mouse is not on the button

4.4.3.3 setColor()

```
void Button::setColor (
            sf::Color color )  [inline]
```

Sets the color of the button.

Parameters

color	The color to be set to

4.4.3.4 setFont()

```
void Button::setFont (
            sf::Font & font )  [inline]
```

Sets the font of the text of the button.

Parameters

font	The font to be set to

4.4.3.5 setPos()

```
void Button::setPos (
            sf::Vector2f pos )  [inline]
```

Sets the position of the button in the window.

Parameters

pos	Position of the button

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

- interface/button.hpp

4.5 BVH Class Reference

Bounding volume hierarchy class for optimising TriangleMesh collisions. Creates a data structure that divides the triangles of the mesh in to groups of Axis Aligned Bounding Boxes (AABB). If a ray misses a box then none of the triangles in the box need to be checked for a collision.

```
#include <bvh.hpp>
```

Public Member Functions

- **BVH** ()

  Default constructor for BVH.
- BVH (std::vector< Triangle > tris)

  Construct a new BVH object.
- void UpdateNodeBounds (std::vector< Node > &bvhNodes, int nodeIdx)

  Updates the bounds of the AABB based on the triangles it contains.
- void Subdivide (std::vector< Node > &bvhNodes, int nodeIdx)

  Divides the traingles of the current node into new AABBs recursively.
- void BVHCollision (Ray &ray, Hit &rayHit, float &smallestDistance, const int nodeIdx)

  Calculate whether a given ray collides with the TriangleMesh contained in the BVH recursively.
- bool AABBCollision (AABB box, Ray ray, float smallestDistance)

Calculate whether a given ray collides with the AABB.

- int getRootNodeIdx () const

  Get the RootNodeIdx of the BVH.

- std::vector< Triangle > getTriangles () const

  Get the triangles vector.

- std::vector< Node > getNodes () const

  Get the nodes vector.

4.5.1 Detailed Description

Bounding volume hierarchy class for optimising TriangleMesh collisions. Creates a data structure that divides the triangles of the mesh in to groups of Axis Aligned Bounding Boxes (AABB). If a ray misses a box then none of the triangles in the box need to be checked for a collision.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 BVH()

```
BVH::BVH (
            std::vector< Triangle > tris )  [inline]
```

Construct a new BVH object.

Parameters

tris	vector containing all the triangles in the mesh

4.5.3 Member Function Documentation

4.5.3.1 AABBCollision()

```
bool BVH::AABBCollision (
            AABB box,
            Ray ray,
            float smallestDistance )  [inline]
```

Calculate whether a given ray collides with the AABB.

Parameters

box	the AABB for which collision is tested
ray	ray whose collision will be checked
smallestDistance	current smallest distance

Returns

true If the ray collides with the box

false If the ray does not collide with the boc

4.5.3.2 BVHCollision()

```
void BVH::BVHCollision (
            Ray & ray,
            Hit & rayHit,
            float & smallestDistance,
            const int nodeIdx )  [inline]
```

Calculate whether a given ray collides with the TriangleMesh contained in the BVH recursively.

Parameters

ray	ray whose collision will be checked
rayHit	address of a Hit data structure
smallestDistance	current smallest distance
nodeIdx	current node index

4.5.3.3 getNodes()

```
std::vector< Node > BVH::getNodes ( ) const  [inline]
```

Get the nodes vector.

Returns

std::vector<Node>

4.5.3.4 getRootNodeIdx()

```
int BVH::getRootNodeIdx ( ) const  [inline]
```

Get the RootNodeIdx of the BVH.

Returns

int

4.5.3.5 getTriangles()

```
std::vector< Triangle > BVH::getTriangles ( ) const  [inline]
```

Get the triangles vector.

Returns

std::vector<Triangle>

4.5.3.6 Subdivide()

```
void BVH::Subdivide (
            std::vector< Node > & bvhNodes,
            int nodeIdx )  [inline]
```

Divides the traingles of the current node into new AABBs recursively.

Parameters

bvhNodes	vector containing the nodes
nodeIdx	current node index

4.5.3.7 UpdateNodeBounds()

```
void BVH::UpdateNodeBounds (
            std::vector< Node > & bvhNodes,
            int nodeIdx )  [inline]
```

Updates the bounds of the AABB based on the triangles it contains.

Parameters

bvhNodes	vector containing the nodes
nodeIdx	the current node index

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

- objects/bvh.hpp

4.6 Camera Struct Reference

Struct representing the camera.

```
#include <types.hpp>
```

Public Attributes

- Point **position**
- Vector **lookingAt**
- Vector **direction**
- Vector **up**
- Vector **left**
- float **fov**
- float **focus_distance**
- float **DoF**

4.6.1 Detailed Description

Struct representing the camera.

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

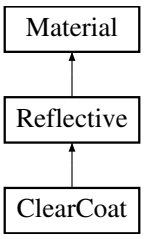
- utils/types.hpp

4.7 ClearCoat Class Reference

Representation of a clear coat material, which can be used for materials that are partially reflective and partially diffusive. Reflective color can be defined independently from diffusive color.

`#include <material.hpp>`

Inheritance diagram for ClearCoat:



Public Member Functions

- Color getClearCoatColor ()

  Gets the clear coat color for the material.
- float getClearCoat ()

  Gets clear coat value for the clear coat material.
- ClearCoat (Color color, std::string name, float specularity, float clearcoat, Color clearcoat_color)

  Constructor for ClearCoat class.
- void updateRay (Ray &ray, Hit &hit)

  Updates the ray according to properties of mirror material.

Public Member Functions inherited from Reflective

- Reflective (Color color, std::string name, float specularity)

  Constructor for Reflective class material.
- float getSpecularity ()

  Gets the specularity of the reflective material.
- void updateRay (Ray &ray, Hit &hit)

  Updates the ray according to properties of reflective material.

Public Member Functions inherited from Material

- Color getColor ()

  Gets the color vector for the material.
- std::string getName ()

  Gets the name of the material.
- Material (Color color, std::string name)

  Constructor for material class.
- Vector reflectionDir (Ray &ray, Hit &hit)

  Computes the direction of perfectly reflected ray.
- Vector diffuseDir (Hit &hit)

  Computes the direction of the diffuse ray.
- void updateColor (Ray &ray)

  Updates the color of the ray after interacting with the material.

Additional Inherited Members

Public Attributes inherited from Material

- RandomGenerator **rnd_**

4.7.1 Detailed Description

Representation of a clear coat material, which can be used for materials that are partially reflective and partially diffusive. Reflective color can be defined independently from diffusive color.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 ClearCoat()

```
ClearCoat::ClearCoat (
            Color color,
            std::string name,
            float specularity,
            float clearcoat,
            Color clearcoat_color )  [inline]
```

Constructor for ClearCoat class.

Parameters

color	Material color as RGB value
name	Name of the material
specularity	Specularity of the material
clearcoat	How large proportion of the rays behaves reflectively
clearcoat_color	Color of the reflected rays

4.7.3 Member Function Documentation

4.7.3.1 getClearCoat()

```
float ClearCoat::getClearCoat ( )  [inline]
```

Gets clear coat value for the clear coat material.

Returns

float representing probability of the clear coat bounce

4.7.3.2 getClearCoatColor()

```
Color ClearCoat::getClearCoatColor ( )  [inline]
```

Gets the clear coat color for the material.

Returns

Color that represents the RGB value for the clear coat bounces

4.7.3.3 updateRay()

```
void ClearCoat::updateRay (
            Ray & ray,
            Hit & hit )  [inline], [virtual]
```

Updates the ray according to properties of mirror material.

Parameters

ray	Ray that did hit the material

Implements Material.

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

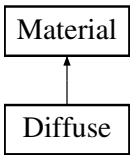
- scenery/material.hpp

4.8 Diffuse Class Reference

Representation of a diffusive material.

```
#include <material.hpp>
```

Inheritance diagram for Diffuse:



Public Member Functions

- Diffuse (Color color, std::string name, float emission_strength, Color emission_color)

  Constructor for Diffuse class.
- Color getEmColor ()

  Gets the emission color for the material.
- float getEmStrength ()

  Gets the emission strength for the material.
- bool isEmitting ()

  Tells if the material is emitting i.e. has emission strength larger than zero.
- void updateRay (Ray &ray, Hit &hit)

  Updates the ray according to properties of diffuse material.

Public Member Functions inherited from Material

- Color getColor ()

  Gets the color vector for the material.
- std::string getName ()

  Gets the name of the material.
- Material (Color color, std::string name)

  Constructor for material class.
- Vector reflectionDir (Ray &ray, Hit &hit)

  Computes the direction of perfectly reflected ray.
- Vector diffuseDir (Hit &hit)

  Computes the direction of the diffuse ray.
- void updateColor (Ray &ray)

  Updates the color of the ray after interacting with the material.

Additional Inherited Members

Public Attributes inherited from Material

- RandomGenerator **rnd_**

4.8.1 Detailed Description

Representation of a diffusive material.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 Diffuse()

```
Diffuse::Diffuse (
            Color color,
            std::string name,
            float emission_strength,
            Color emission_color )  [inline]
```

Constructor for Diffuse class.

Parameters

color	Material color as RGB value
name	Name of the material
emission_strength	Emission strength of the material
emission_color	Emission color of the material

4.8.3 Member Function Documentation

4.8.3.1 getEmColor()

```
Color Diffuse::getEmColor ( )  [inline]
```

Gets the emission color for the material.

Returns

Color vector representing the RGB value for the emission color

4.8.3.2 getEmStrength()

```
float Diffuse::getEmStrength ( )  [inline]
```

Gets the emission strength for the material.

Returns

float representing the emission strength for the material

4.8.3.3 isEmitting()

```
bool Diffuse::isEmitting ( )  [inline]
```

Tells if the material is emitting i.e. has emission strength larger than zero.

Returns

true if material is emitting and zero if it is not emitting

4.8.3.4 updateRay()

```
void Diffuse::updateRay (
            Ray & ray,
            Hit & hit )  [inline], [virtual]
```

Updates the ray according to properties of diffuse material.

Parameters

ray	Ray that did hit the material

Implements Material.

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

- scenery/material.hpp

4.9 Environment Class Reference

An object for representing a scene environment.

`#include <environment.hpp>`

Public Member Functions

- **Environment** ()

  Construct a new Environment with black void.
- void setSky (Color skyColor=Color(0.2, 0.5, 1.0), Color horizonColor=Color(0.7, 0.8, 0.8), Color groundColor=Color(0.1, 0.1, 0.1))

  Set a sky to the environment.
- Light getLight (Ray &ray)

  Calculate environment light collected by a runaway ray.
- Color **getHorizonColor** ()
- Color **getGroundColor** ()
- Color **getSkyColor** ()

4.9.1 Detailed Description

An object for representing a scene environment.

Currently only solid color or sky environments are supported.

4.9.2 Member Function Documentation

4.9.2.1 getLight()

```
Light Environment::getLight (
            Ray & ray )  [inline]
```

Calculate environment light collected by a runaway ray.

The ray direction determines what the ray should "see" in the environment.

Parameters

ray	a ray whose path doesn't intersect with any objects

Returns

Light collected from the environment by the ray

4.9.2.2 setSky()

```
void Environment::setSky (
            Color skyColor = Color(0.2, 0.5, 1.0),
            Color horizonColor = Color(0.7, 0.8, 0.8),
            Color groundColor = Color(0.1, 0.1, 0.1) )  [inline]
```

Set a sky to the environment.

Parameters are optional, otherwise a default sky is created.

Parameters

skyColor	
horizonColor	
groundColor	

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

- scenery/environment.hpp

4.10 FileLoader Class Reference

Implements a class for reading yaml scene files.

```
#include <fileloader.hpp>
```

Public Member Functions

- FileLoader (std::string filepath)

  Construct a new FileLoader object.
- std::shared_ptr< Scene > loadSceneFile ()

  Loads a scene from the file specified in filepath and returns a shared pointer to this scene.
- ~**FileLoader** ()

  Destructor for FileLoader object.
- std::string **getFilepath** ()

4.10.1 Detailed Description

Implements a class for reading yaml scene files.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 FileLoader()

```
FileLoader::FileLoader (
            std::string filepath )  [inline]
```

Construct a new FileLoader object.

Parameters

filepath	Filepath to the yaml file that contains properties of the scene

4.10.3 Member Function Documentation

4.10.3.1 loadSceneFile()

```
std::shared_ptr< Scene > FileLoader::loadSceneFile ( )  [inline]
```

Loads a scene from the file specified in filepath and returns a shared pointer to this scene.

Returns

A shared pointer to scene object

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

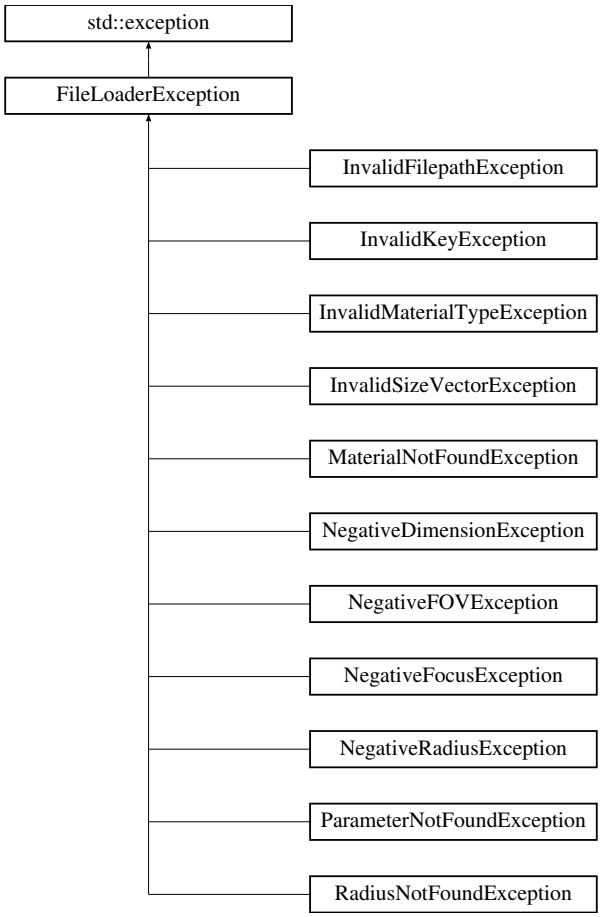
- fileloader/fileloader.hpp

4.11 FileLoaderException Class Reference

An abstract base class for FileLoader exceptions. Exceptions are designed for invalid user inputs in the YAML files.

```
#include <fileloader_ex.hpp>
```

Inheritance diagram for FileLoaderException:



Public Member Functions

- virtual const char * what () const noexcept=0

  Pure virtual function overload for exception messages.

4.11.1 Detailed Description

An abstract base class for FileLoader exceptions. Exceptions are designed for invalid user inputs in the YAML files.

4.11.2 Member Function Documentation

4.11.2.1 what()

```
virtual const char * FileLoaderException::what ( ) const  [pure virtual], [noexcept]
```

Pure virtual function overload for exception messages.

Implemented in InvalidFilepathException, NegativeRadiusException, InvalidSizeVectorException, NegativeFOVException, NegativeFocusException, InvalidKeyException, MaterialNotFoundException, RadiusNotFoundException, ParameterNotFoundExceptio NegativeDimensionException, and InvalidMaterialTypeException.

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

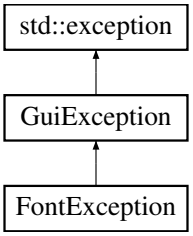
- fileloader/fileloader_ex.hpp

4.12 FontException Class Reference

Exception for not finding the given font.

`#include <gui_ex.hpp>`

Inheritance diagram for FontException:



Public Member Functions

- **FontException** (std::string fontPath)
- virtual const char * what () const noexcept

4.12.1 Detailed Description

Exception for not finding the given font.

4.12.2 Member Function Documentation

4.12.2.1 what()

`virtual const char * FontException::what ( ) const [inline], [virtual], [noexcept]`

Implements GuiException.

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

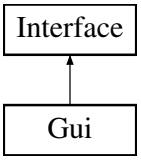
- interface/gui_ex.hpp

4.13 Gui Class Reference

Implements the Graphical user interface class.

`#include <gui.hpp>`

Inheritance diagram for Gui:



Public Member Functions

- std::shared_ptr< Scene > titleScreen ()

  Opens a titlescreen where the user can input a filepath. If the filepath contains a valid yaml file, a shared pointer to a scene constructed with this file is returned. An invalid filepath will result in an error being shown to the user.
- void openSettings (std::shared_ptr< Scene > loadedScene)

  Opens a settings menu where the user can preview and influence a loaded scene.
- void **openRender** (int resX, int resY, std::shared_ptr< Scene > loadedScene, int sampleSize, float dof, int bounces)

Public Member Functions inherited from Interface

- void createImg (std::vector< std::vector< Color > > pixels)

  Create a sf::Image from the matrix of RGB values.
- bool saveImage (const std::string &filename)

  Save the image with given filename.
- sf::Image **getImage** ()

4.13.1 Detailed Description

Implements the Graphical user interface class.

4.13.2 Member Function Documentation

4.13.2.1 openSettings()

```
void Gui::openSettings (
            std::shared_ptr< Scene > loadedScene )  [inline]
```

Opens a settings menu where the user can preview and influence a loaded scene.

The setting menu contains a "preview" button and textboxes for parameter. There are two types of parameters that the user can change. First are the parameters that influence only the final render the user can initiate with pressing enter. The other parameters, in addition to changing the final render, affect the preview image.

Parameters

loadedScene	the scene to be influenced

4.13.2.2 titleScreen()

```
std::shared_ptr< Scene > Gui::titleScreen ( )  [inline]
```

Opens a titlescreen where the user can input a filepath. If the filepath contains a valid yaml file, a shared pointer to a scene constructed with this file is returned. An invalid filepath will result in an error being shown to the user.

Returns

std::shared_ptr<Scene> The shared pointer to the scene created with the inputted yaml file

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

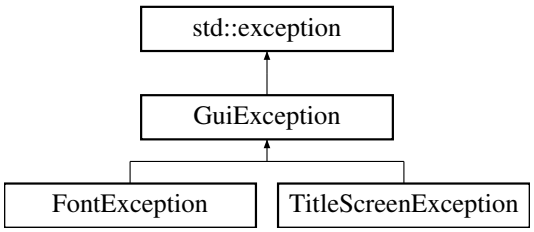
- interface/gui.hpp

4.14 GuiException Class Reference

Abstact class for a GUI exception.

`#include <gui_ex.hpp>`

Inheritance diagram for GuiException:



Public Member Functions

- virtual const char * **what** () const noexcept=0

4.14.1 Detailed Description

Abstact class for a GUI exception.

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

- interface/gui_ex.hpp

4.15 Hit Struct Reference

Struct containing information about a ray hitting an object.

`#include <types.hpp>`

Public Attributes

- bool **did_hit** = false
- std::shared_ptr< Material > **material**
- Vector **normal**
- Point **point**
- float **distance**

4.15.1 Detailed Description

Struct containing information about a ray hitting an object.

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

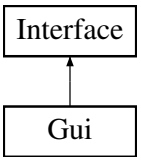
- utils/types.hpp

4.16 Interface Class Reference

Class that creates an image from a raw RGB matrix and saves the image.

`#include <interface.hpp>`

Inheritance diagram for Interface:



Public Member Functions

- void createImg (std::vector< std::vector< Color > > pixels)
  Create a sf::Image from the matrix of RGB values.
- bool saveImage (const std::string &filename)
  Save the image with given filename.
- sf::Image **getImage** ()

4.16.1 Detailed Description

Class that creates an image from a raw RGB matrix and saves the image.

4.16.2 Member Function Documentation

4.16.2.1 createImg()

```
void Interface::createImg (
            std::vector< std::vector< Color > > pixels )  [inline]
```

Create a sf::Image from the matrix of RGB values.

Parameters

pixels	matrix of RGB values

4.16.2.2 saveImage()

```
bool Interface::saveImage (
            const std::string & filename )  [inline]
```

Save the image with given filename.

Parameters

filename	Image filename

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

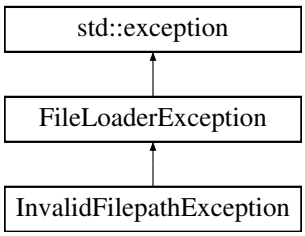
- interface/interface.hpp

4.17 InvalidFilepathException Class Reference

Representation of invalid filepath exception.

`#include <fileloader_ex.hpp>`

Inheritance diagram for InvalidFilepathException:



Public Member Functions

- InvalidFilepathException (std::string filepath)
  Constructor for InvalidFilePathException.
- virtual const char * what () const noexcept
  Function that creates an exception message for invalid filepath.

4.17.1 Detailed Description

Representation of invalid filepath exception.

4.17.2 Constructor & Destructor Documentation

4.17.2.1 InvalidFilepathException()

```
InvalidFilepathException::InvalidFilepathException (
            std::string filepath )  [inline]
```

Constructor for InvalidFilePathException.

Parameters

filepath	Filepath of the given YAML file

4.17.3 Member Function Documentation

4.17.3.1 what()

```
virtual const char * InvalidFilepathException::what ( ) const  [inline], [virtual], [noexcept]
```

Function that creates an exception message for invalid filepath.

Returns

Pointer to the first char of the exception message.

Implements FileLoaderException.

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

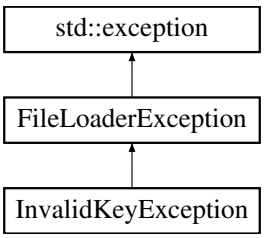
- fileloader/fileloader_ex.hpp

4.18 InvalidKeyException Class Reference

Representation of invalid key exception.

```
#include <fileloader_ex.hpp>
```

Inheritance diagram for InvalidKeyException:



Public Member Functions

- InvalidKeyException (std::string filepath, std::string key)

  Constructor for InvalidKeyException.
- virtual const char * what () const noexcept

  Function that creates an exception message for invalid key value in the YAML file.

4.18.1 Detailed Description

Representation of invalid key exception.

4.18.2 Constructor & Destructor Documentation

4.18.2.1 InvalidKeyException()

```
InvalidKeyException::InvalidKeyException (
            std::string filepath,
            std::string key )  [inline]
```

Constructor for InvalidKeyException.

Parameters

filepath	Filepath of the YAML file
key	String of the key that was not found from the YAML file

4.18.3 Member Function Documentation

4.18.3.1 what()

```
virtual const char * InvalidKeyException::what ( ) const  [inline], [virtual], [noexcept]
```

Function that creates an exception message for invalid key value in the YAML file.

Returns

Pointer to the first char of the exception message

Implements FileLoaderException.

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

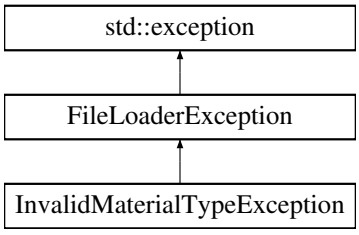
- fileloader/fileloader_ex.hpp

4.19 InvalidMaterialTypeException Class Reference

Representation of invalid material type exception.

`#include <fileloader_ex.hpp>`

Inheritance diagram for InvalidMaterialTypeException:



Public Member Functions

- InvalidMaterialTypeException (std::string filepath, int line)

  Constructor for InvalidMaterialTypeException.
- virtual const char * what () const noexcept

  Function that creates an exception message for invalid material type in the YAML file.

4.19.1 Detailed Description

Representation of invalid material type exception.

4.19.2 Constructor & Destructor Documentation

4.19.2.1 InvalidMaterialTypeException()

```
InvalidMaterialTypeException::InvalidMaterialTypeException (
            std::string filepath,
            int line )  [inline]
```

Constructor for InvalidMaterialTypeException.

Parameters

filepath	Filepath of the YAML file
line	Line number where material is defined

4.19.3 Member Function Documentation

4.19.3.1 what()

```
virtual const char * InvalidMaterialTypeException::what ( ) const  [inline], [virtual], [noexcept]
```

Function that creates an exception message for invalid material type in the YAML file.

Returns

Pointer to the first char of the exception message

Implements FileLoaderException.

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

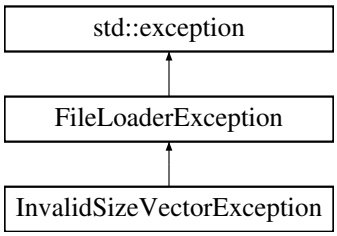
- fileloader/fileloader_ex.hpp

4.20 InvalidSizeVectorException Class Reference

Representation of invalid size vector exception.

`#include <fileloader_ex.hpp>`

Inheritance diagram for InvalidSizeVectorException:



Public Member Functions

- InvalidSizeVectorException (std::string filepath, size_t size, int line)

  Constructor for InvalidSizeVectorException.
- virtual const char * what () const noexcept

  Function that creates an exception message for invalid size vector.

4.20.1 Detailed Description

Representation of invalid size vector exception.

4.20.2 Constructor & Destructor Documentation

4.20.2.1 InvalidSizeVectorException()

```
InvalidSizeVectorException::InvalidSizeVectorException (
            std::string filepath,
            size_t size,
            int line )  [inline]
```

Constructor for InvalidSizeVectorException.

Parameters

filepath	Filepath of the YAML file
size	Size of the invalid size vector
line	Line number where the invalid size vector is defined

4.20.3 Member Function Documentation

4.20.3.1 what()

```
virtual const char * InvalidSizeVectorException::what ( ) const  [inline], [virtual], [noexcept]
```

Function that creates an exception message for invalid size vector.

Returns

Pointer to the first char of the exception message

Implements FileLoaderException.

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

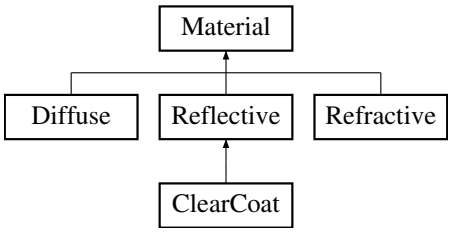
- fileloader/fileloader_ex.hpp

4.21 Material Class Reference

An abstract base class for any type of material object can have.

```
#include <material.hpp>
```

Inheritance diagram for Material:



Public Member Functions

- Color getColor ()
  Gets the color vector for the material.
- std::string getName ()
  Gets the name of the material.
- Material (Color color, std::string name)
  Constructor for material class.
- Vector reflectionDir (Ray &ray, Hit &hit)
  Computes the direction of perfectly reflected ray.
- Vector diffuseDir (Hit &hit)
  Computes the direction of the diffuse ray.
- void updateColor (Ray &ray)
  Updates the color of the ray after interacting with the material.
- virtual void updateRay (Ray &ray, Hit &hit)=0
  Pure virtual function that updates the ray according to the properties of the material.

Public Attributes

- RandomGenerator **rnd_**

4.21.1 Detailed Description

An abstract base class for any type of material object can have.

4.21.2 Constructor & Destructor Documentation

4.21.2.1 Material()

```
Material::Material (
            Color color,
            std::string name )  [inline]
```

Constructor for material class.

Parameters

color	Material color as RGB value
name	Name of the material

4.21.3 Member Function Documentation

4.21.3.1 diffuseDir()

```
Vector Material::diffuseDir (
            Hit & hit )  [inline]
```

Computes the direction of the diffuse ray.

randomDirection() and ray.normal are both normalized so this gives a random vector whose probability distribution is weighted towards the ray.normal.

Parameters

ray	Ray that did hit the material

Returns

vector towards the direction of the diffused ray

4.21.3.2 getColor()

```
Color Material::getColor ( )  [inline]
```

Gets the color vector for the material.

Returns

Color vector representing the RGB value for the material

4.21.3.3 getName()

```
std::string Material::getName ( )  [inline]
```

Gets the name of the material.

Returns

std::string representing the name of the material

4.21.3.4 reflectionDir()

```
Vector Material::reflectionDir (
            Ray & ray,
            Hit & hit )  [inline]
```

Computes the direction of perfectly reflected ray.

Parameters

ray	Ray that did hit the material

Returns

vector towards the direction of the reflected ray

4.21.3.5 updateColor()

```
void Material::updateColor (
            Ray & ray )  [inline]
```

Updates the color of the ray after interacting with the material.

Component wise product with the original color and the color of the interaction material.

Parameters

ray	Ray that did hit the material

4.21.3.6 updateRay()

```
virtual void Material::updateRay (
```

```
            Ray & ray,
            Hit & hit )  [pure virtual]
```

Pure virtual function that updates the ray according to the properties of the material.

Parameters

ray	Ray that did hit the material

Implemented in Diffuse, Reflective, ClearCoat, and Refractive.

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

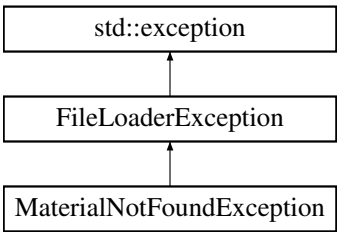
- scenery/material.hpp

4.22 MaterialNotFoundException Class Reference

Representation of material not found exception.

`#include <fileloader_ex.hpp>`

Inheritance diagram for MaterialNotFoundException:



Public Member Functions

- MaterialNotFoundException (std::string filepath, int line)
  Constructor for MaterialNotFoundException.
- virtual const char * what () const noexcept
  Function that creates an exception message for objects that are missing the material definition in the YAML file.

4.22.1 Detailed Description

Representation of material not found exception.

4.22.2 Constructor & Destructor Documentation

4.22.2.1 MaterialNotFoundException()

```
MaterialNotFoundException::MaterialNotFoundException (
            std::string filepath,
            int line )  [inline]
```

Constructor for MaterialNotFoundException.

Parameters

filepath	Filepath of the YAML file
line	Line number of the object that do not have material defined

4.22.3 Member Function Documentation

4.22.3.1 what()

```
virtual const char * MaterialNotFoundException::what ( ) const  [inline], [virtual], [noexcept]
```

Function that creates an exception message for objects that are missing the material definition in the YAML file.

Returns

Pointer to the first char of the exception message

Implements FileLoaderException.

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

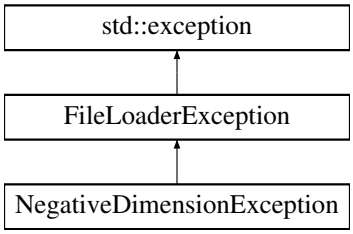
- fileloader/fileloader_ex.hpp

4.23 NegativeDimensionException Class Reference

Representation of negative dimension exception.

```
#include <fileloader_ex.hpp>
```

Inheritance diagram for NegativeDimensionException:



Public Member Functions

- NegativeDimensionException (std::string filepath, float value, int line)

  Constructor for NegativeDimensionException.
- virtual const char * what () const noexcept

  Function that creates an exception message for negative dimension value.

4.23.1 Detailed Description

Representation of negative dimension exception.

4.23.2 Constructor & Destructor Documentation

4.23.2.1 NegativeDimensionException()

```
NegativeDimensionException::NegativeDimensionException (
            std::string filepath,
            float value,
            int line )  [inline]
```

Constructor for NegativeDimensionException.

Parameters

filepath	Filepath of the YAML file
value	Value of the negative dimension
line	Line number where the negative dimension is defined

4.23.3 Member Function Documentation

4.23.3.1 what()

```
virtual const char * NegativeDimensionException::what ( ) const  [inline], [virtual], [noexcept]
```

Function that creates an exception message for negative dimension value.

Returns

Pointer to the first char of the exception message

Implements FileLoaderException.

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

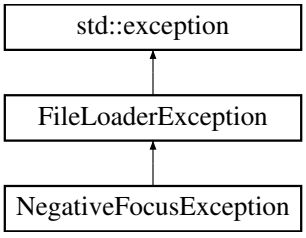
- fileloader/fileloader_ex.hpp

4.24 NegativeFocusException Class Reference

Representation of negative focus exception.

`#include <fileloader_ex.hpp>`

Inheritance diagram for NegativeFocusException:



Public Member Functions

- NegativeFocusException (std::string filepath, float focus, int line)

  Constructor for NegativeFocusException.
- virtual const char * what () const noexcept

  Function that creates an exception message for negative focus distance.

4.24.1 Detailed Description

Representation of negative focus exception.

4.24.2 Constructor & Destructor Documentation

4.24.2.1 NegativeFocusException()

```
NegativeFocusException::NegativeFocusException (
            std::string filepath,
            float focus,
            int line )  [inline]
```

Constructor for NegativeFocusException.

Parameters

filepath	Filepath of the YAML file
focus	Value of the negative focus distance
line	Line number where the negative dimension is defined

4.24.3 Member Function Documentation

4.24.3.1 what()

```
virtual const char * NegativeFocusException::what ( ) const  [inline], [virtual], [noexcept]
```

Function that creates an exception message for negative focus distance.

Returns

Pointer to the first char of the exception message

Implements FileLoaderException.

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

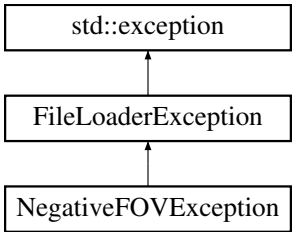
- fileloader/fileloader_ex.hpp

4.25 NegativeFOVException Class Reference

Representation of negative field of view exception.

`#include <fileloader_ex.hpp>`

Inheritance diagram for NegativeFOVException:



Public Member Functions

- NegativeFOVException (std::string filepath, float fow, int line)

  Constructor for NegativeFOVException.
- virtual const char * what () const noexcept

  Function that creates an exception message for negative field of view.

4.25.1 Detailed Description

Representation of negative field of view exception.

4.25.2 Constructor & Destructor Documentation

4.25.2.1 NegativeFOVException()

```
NegativeFOVException::NegativeFOVException (
            std::string filepath,
            float fow,
            int line )  [inline]
```

Constructor for NegativeFOVException.

Parameters

filepath	Filepath of the YAML file
fow	Value of the negative field of view
line	Line number where the negative fow is defined

4.25.3 Member Function Documentation

4.25.3.1 what()

```
virtual const char * NegativeFOVException::what ( ) const  [inline], [virtual], [noexcept]
```

Function that creates an exception message for negative field of view.

Returns

Pointer to the first char of the exception message

Implements FileLoaderException.

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

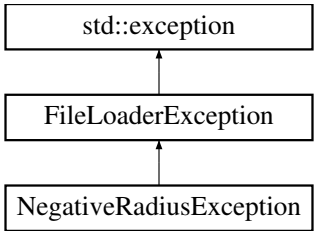
- fileloader/fileloader_ex.hpp

4.26 NegativeRadiusException Class Reference

Representation of negative radius exception.

`#include <fileloader_ex.hpp>`

Inheritance diagram for NegativeRadiusException:



Public Member Functions

- NegativeRadiusException (std::string filepath, float radius, int line)
  Constructor for NegativeRadiusException.
- virtual const char * what () const noexcept
  Function that creates an exception message for negative radius.

4.26.1 Detailed Description

Representation of negative radius exception.

4.26.2 Constructor & Destructor Documentation

4.26.2.1 NegativeRadiusException()

```
NegativeRadiusException::NegativeRadiusException (
            std::string filepath,
            float radius,
            int line )  [inline]
```

Constructor for NegativeRadiusException.

Parameters

filepath	Filepath of the YAML file
radius	Value of the negative radius
line	Line number where the negative radius is defined

4.26.3 Member Function Documentation

4.26.3.1 what()

```
virtual const char * NegativeRadiusException::what ( ) const  [inline], [virtual], [noexcept]
```

Function that creates an exception message for negative radius.

Returns

Pointer to the first char of the exception message.

Implements FileLoaderException.

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

- fileloader/fileloader_ex.hpp

4.27 Node Struct Reference

Struct representing a node in the bounding volume hiererchy.

```
#include <bvh.hpp>
```

Public Member Functions

- bool **isLeaf** ()

Public Attributes

- AABB **box**
- int **leftChild**
- int **firstTrildx**
- int **triCount**

4.27.1 Detailed Description

Struct representing a node in the bounding volume hiererchy.

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

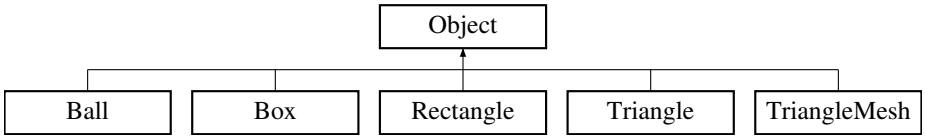
- objects/bvh.hpp

4.28 Object Class Reference

An abstract base class for any type of visible object in a scene.

`#include <object.hpp>`

Inheritance diagram for Object:



Public Member Functions

- **Object** (Vector position, std::shared_ptr< Material > material)
- Vector **getPosition** () const
- std::shared_ptr< Material > **getMaterial** () const
- virtual void collision (Ray &ray, Hit &rayHit, float &smallestDistance)=0

  Calculate whether a given ray collides with the object.
- virtual void printInfo (std::ostream &out) const =0

  Print object info to the desired output stream.

4.28.1 Detailed Description

An abstract base class for any type of visible object in a scene.

4.28.2 Member Function Documentation

4.28.2.1 collision()

```
virtual void Object::collision (
            Ray & ray,
            Hit & rayHit,
            float & smallestDistance )  [pure virtual]
```

Calculate whether a given ray collides with the object.

If the ray collides with the object and the collision is closer than the current smallest distance, the "rayHit" data structure will be updated according to the collision.

Parameters

ray	ray whose collision will be checked
rayHit	address of a Hit data structure
smallestDistance	

Implemented in Ball, Box, Rectangle, Triangle, and TriangleMesh.

4.28.2.2 printInfo()

```
virtual void Object::printInfo (
            std::ostream & out ) const  [pure virtual]
```

Print object info to the desired output stream.

Parameters

out	output stream
object	object to be printed

Returns

std::ostream& the output stream

Implemented in Ball, Box, Rectangle, Triangle, and TriangleMesh.

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

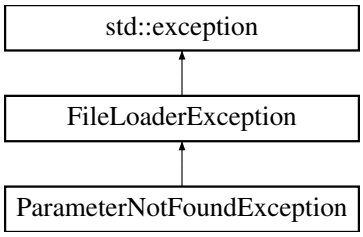
- objects/object.hpp

4.29 ParameterNotFoundException Class Reference

Representation of parameter not found exception.

```
#include <fileloader_ex.hpp>
```

Inheritance diagram for ParameterNotFoundException:



Public Member Functions

- ParameterNotFoundException (std::string filepath, int line)

  Constructor for ParameterNotFoundException.
- virtual const char * what () const noexcept

  Function that creates an exception message for missing parameters.

4.29.1 Detailed Description

Representation of parameter not found exception.

4.29.2 Constructor & Destructor Documentation

4.29.2.1 ParameterNotFoundException()

```
ParameterNotFoundException::ParameterNotFoundException (
            std::string filepath,
            int line )  [inline]
```

Constructor for ParameterNotFoundException.

Parameters

filepath	Filepath of the YAML file
line	Line number where the parameter is missing

4.29.3 Member Function Documentation

4.29.3.1 what()

```
virtual const char * ParameterNotFoundException::what ( ) const  [inline], [virtual], [noexcept]
```

Function that creates an exception message for missing parameters.

Returns

Pointer to the first char of the exception message

Implements FileLoaderException.

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

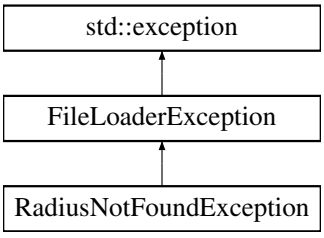
- fileloader/fileloader_ex.hpp

4.30 RadiusNotFoundException Class Reference

Representation of radius not found exception.

```
#include <fileloader_ex.hpp>
```

Inheritance diagram for RadiusNotFoundException:



Public Member Functions

- RadiusNotFoundException (std::string filepath, int line)

  Constructor for RadiusNotFoundException.
- virtual const char * what () const noexcept

  Function that creates an exception message for ball object that do not have radius defined in the YAML file.

4.30.1 Detailed Description

Representation of radius not found exception.

4.30.2 Constructor & Destructor Documentation

4.30.2.1 RadiusNotFoundException()

```
RadiusNotFoundException::RadiusNotFoundException (
            std::string filepath,
            int line )  [inline]
```

Constructor for RadiusNotFoundException.

Parameters

filepath	Filepath of the YAML file
line	Line number of the ball that do not have radius defined

4.30.3 Member Function Documentation

4.30.3.1 what()

```
virtual const char * RadiusNotFoundException::what ( ) const  [inline], [virtual], [noexcept]
```

Function that creates an exception message for ball object that do not have radius defined in the YAML file.

Returns

Pointer to the first char of the exception message

Implements FileLoaderException.

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

- fileloader/fileloader_ex.hpp

4.31 RandomGenerator Class Reference

An object for creating random numbers and directions.

`#include <randomgenerator.hpp>`

Public Member Functions

- Vector randomDirection ()

  Creates a random direction, i.e., a random point on the unit sphere.
- float randomZeroToOne ()

  Returns a random real number between 0 and 1.
- Vector2 randomInCircle ()

  Creates a random point inside the unit disk.

4.31.1 Detailed Description

An object for creating random numbers and directions.

4.31.2 Member Function Documentation

4.31.2.1 randomDirection()

```
Vector RandomGenerator::randomDirection ( )  [inline]
```

Creates a random direction, i.e., a random point on the unit sphere.

Returns

3-dimensional vector pointing to a random direction

4.31.2.2 randomInCircle()

```
Vector2 RandomGenerator::randomInCircle ( )  [inline]
```

Creates a random point inside the unit disk.

Returns

2-dimensional vector inside of the unit disk.

4.31.2.3 randomZeroToOne()

```
float RandomGenerator::randomZeroToOne ( )  [inline]
```

Returns a random real number between 0 and 1.

Returns

float

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

- utils/randomgenerator.hpp

4.32 Ray Struct Reference

Struct representing a ray.

```
#include <types.hpp>
```

Public Attributes

- Point **origin**
- Vector **direction**
- bool **inside_material** = false
- Color **color** = Color(1.0, 1.0, 1.0)
- Light **light** = Color(0.0, 0.0, 0.0)

4.32.1 Detailed Description

Struct representing a ray.

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

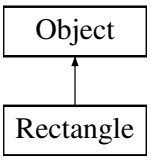
- utils/types.hpp

4.33 Rectangle Class Reference

Representation of a box (rectangular cuboid) object in the scene.

```
#include <rectangle.hpp>
```

Inheritance diagram for Rectangle:



Public Member Functions

- **Rectangle** (Vector position, float width, float height, std::shared_ptr< Material > material)
- void collision (Ray &ray, Hit &rayHit, float &smallestDistance)

  Calculate whether a given ray collides with the rectangle.
- float **getWidth** () const
- float **getHeight** () const
- void rotate (float angle, Vector axis)

  Rotates the box around a given axis.
- void printInfo (std::ostream &out) const

  Print box info to the desired output stream.

Public Member Functions inherited from Object

- **Object** (Vector position, std::shared_ptr< Material > material)
- Vector **getPosition** () const
- std::shared_ptr< Material > **getMaterial** () const

4.33.1 Detailed Description

Representation of a box (rectangular cuboid) object in the scene.

The position represent the geometric center of the box.

4.33.2 Member Function Documentation

4.33.2.1 collision()

```
void Rectangle::collision (
            Ray & ray,
            Hit & rayHit,
            float & smallestDistance )  [inline], [virtual]
```

Calculate whether a given ray collides with the rectangle.

If the ray collides with the rectangle and the collision is closer than the current smallest distance, the "rayHit" data structure will be updated according to the collision.

Parameters

ray	ray whose collision will be checked
rayHit	address of a Hit data structure
smallestDistance	

Implements Object.

4.33.2.2 printInfo()

```
void Rectangle::printInfo (
```

```
            std::ostream & out ) const  [inline], [virtual]
```

Print box info to the desired output stream.

Parameters

out	output stream

Returns

std::ostream& the output stream

Implements Object.

4.33.2.3 rotate()

```
void Rectangle::rotate (
            float angle,
            Vector axis )  [inline]
```

Rotates the box around a given axis.

The center of rotation is the box's center of mass.

Parameters

angle	rotation in radians
axis	axis of rotation (has to be normalized)

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

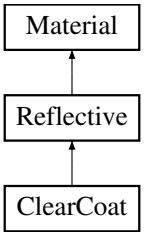
- objects/rectangle.hpp

4.34 Reflective Class Reference

Representation of a reflective material that can be used for mirror or metal like materials.

```
#include <material.hpp>
```

Inheritance diagram for Reflective:



Public Member Functions

- Reflective (Color color, std::string name, float specularity)

  Constructor for Reflective class material.
- float getSpecularity ()

  Gets the specularity of the reflective material.
- void updateRay (Ray &ray, Hit &hit)

  Updates the ray according to properties of reflective material.

Public Member Functions inherited from Material

- Color getColor ()

  Gets the color vector for the material.
- std::string getName ()

  Gets the name of the material.
- Material (Color color, std::string name)

  Constructor for material class.
- Vector reflectionDir (Ray &ray, Hit &hit)

  Computes the direction of perfectly reflected ray.
- Vector diffuseDir (Hit &hit)

  Computes the direction of the diffuse ray.
- void updateColor (Ray &ray)

  Updates the color of the ray after interacting with the material.

Additional Inherited Members

Public Attributes inherited from Material

- RandomGenerator **rnd_**

4.34.1 Detailed Description

Representation of a reflective material that can be used for mirror or metal like materials.

4.34.2 Constructor & Destructor Documentation

4.34.2.1 Reflective()

```
Reflective::Reflective (
            Color color,
            std::string name,
            float specularity )  [inline]
```

Constructor for Reflective class material.

Parameters

color	Material color as RGB value
name	Name of the material
specularity	Specularity of the material

4.34.3 Member Function Documentation

4.34.3.1 getSpecularity()

```
float Reflective::getSpecularity ( )  [inline]
```

Gets the specularity of the reflective material.

Returns

specularity of the object

4.34.3.2 updateRay()

```
void Reflective::updateRay (
            Ray & ray,
            Hit & hit )  [inline], [virtual]
```

Updates the ray according to properties of reflective material.

Parameters

ray	Ray that did hit the material

Implements Material.

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

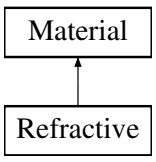
- scenery/material.hpp

4.35 Refractive Class Reference

Representation of a refractive material, which can be used for glass or diamond like materials.

`#include <material.hpp>`

Inheritance diagram for Refractive:



Public Member Functions

- Refractive (Color color, std::string name, float refraction_ratio)

  Constructor for Refractive class.
- void updateRay (Ray &ray, Hit &hit)

  Updates the ray according to properties of Refractive material.

Public Member Functions inherited from Material

- Color getColor ()

  Gets the color vector for the material.
- std::string getName ()

  Gets the name of the material.
- Material (Color color, std::string name)

  Constructor for material class.
- Vector reflectionDir (Ray &ray, Hit &hit)

  Computes the direction of perfectly reflected ray.
- Vector diffuseDir (Hit &hit)

  Computes the direction of the diffuse ray.
- void updateColor (Ray &ray)

  Updates the color of the ray after interacting with the material.

Additional Inherited Members

Public Attributes inherited from Material

- RandomGenerator **rnd_**

4.35.1 Detailed Description

Representation of a refractive material, which can be used for glass or diamond like materials.

4.35.2 Constructor & Destructor Documentation

4.35.2.1 Refractive()

```
Refractive::Refractive (
            Color color,
            std::string name,
            float refraction_ratio )  [inline]
```

Constructor for Refractive class.

Parameters

color	Material color as RGB value
name	Name of the material
refraction_ratio	Refraction ratio from outside to inside the material

4.35.3 Member Function Documentation

4.35.3.1 updateRay()

```
void Refractive::updateRay (
            Ray & ray,
            Hit & hit )  [inline], [virtual]
```

Updates the ray according to properties of Refractive material.

Parameters

ray	Ray that did hit the material
hit	Information about the hit point

Implements Material.

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

- scenery/material.hpp

4.36 Renderer Class Reference

Implements the ray tracing algorithm.

`#include <renderer.hpp>`

Public Member Functions

- Renderer (int res_x, int res_y, std::shared_ptr< Scene > sceneToRender)
  Construct a new Renderer object and initialize all necessary values.
- void setMaxBounces (int bounces)
  Function to set maximum bounces for rays.
- auto parallelRender (int samples)
  Rendering function that uses all available CPU cores.
- void setDof (float dof)
  Set the depth of field for the camera in the renderer.

4.36.1 Detailed Description

Implements the ray tracing algorithm.

4.36.2 Constructor & Destructor Documentation

4.36.2.1 Renderer()

```
Renderer::Renderer (
            int res_x,
            int res_y,
            std::shared_ptr< Scene > sceneToRender )  [inline]
```

Construct a new Renderer object and initialize all necessary values.

Parameters

res_x	horizontal resolution of the rendering area
res_y	vertical resolution of the rendering area
sceneToRender	Scene object to be renderer

4.36.3 Member Function Documentation

4.36.3.1 parallelRender()

```
auto Renderer::parallelRender (
            int samples )  [inline]
```

Rendering function that uses all available CPU cores.

Parameters

samples	Amount of samples that will be taken for each pixel

4.36.3.2 setDof()

```
void Renderer::setDof (
            float dof )  [inline]
```

Set the depth of field for the camera in the renderer.

Parameters

dof	Value for depth of field

4.36.3.3 setMaxBounces()

```
void Renderer::setMaxBounces (
            int bounces )  [inline]
```

Function to set maximum bounces for rays.

Parameters

bounces	Number of maximum bounces ray can take

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

- rendering/renderer.hpp

4.37 Scene Class Reference

Representation of a 3D-scene.

`#include <scene.hpp>`

Public Member Functions

- **Scene** ()=default

  Construct an empty scene.
- Scene (Camera camera, std::list< std::shared_ptr< Object > > objects)

  Construct a new Scene object with a camera and some objects.
- ~**Scene** ()

  Destructor for Scene object. Deletes all the object in the scene.
- Scene & **operator=** (const Scene &that)=default
- **Scene** (const Scene &that)=default
- Camera **getCamera** () const
- Environment & **getEnvironment** ()
- std::list< std::shared_ptr< Object > > **getObjects** () const
- void **setFov** (float fov)
- void **setFocusDist** (float dof)

Friends

- std::ostream & operator<< (std::ostream &out, const Scene &scene)

  Print scene info to the desired output stream.

4.37.1 Detailed Description

Representation of a 3D-scene.

4.37.2 Constructor & Destructor Documentation

4.37.2.1 Scene()

```
Scene::Scene (
            Camera camera,
            std::list< std::shared_ptr< Object > > objects )  [inline]
```

Construct a new Scene object with a camera and some objects.

Parameters

camera	
objects	

4.37.3 Friends And Related Symbol Documentation

4.37.3.1 operator<<

```
std::ostream & operator<< (
            std::ostream & out,
            const Scene & scene )  [friend]
```

Print scene info to the desired output stream.

Parameters

out	output stream
scene	scene to be printed

Returns

std::ostream& the output stream

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

- scenery/scene.hpp

4.38 Textbox Class Reference

Implements a class for creating textboxes that take inputs into SFML-window.

`#include <textbox.hpp>`

Public Member Functions

- Textbox (int size, sf::Color color, bool isSelected, int limit, std::string preText)
  Construct a new Textbox object.
- ~**Textbox** ()=default
  Destroy the Textbox object.
- void setFont (sf::Font &font)
  Set the font of the texbox text.
- void **setSelected** ()
  Set the object to be selected.
- void **setUnselected** ()
  Set the object to be unselected.
- void setPos (sf::Vector2f pos)
  Sets the position of the box in the SFML window.
- void setColor (sf::Color textColor)
  Set the color of the text inside the box.
- void draw (sf::RenderWindow &window)
  Renders a box on a specified window.
- bool onButton (sf::RenderWindow &window)
  Checks if the users mouse is on the textbox.
- std::string getInput ()
  Get the text typed into the textbox.
- void typedOn (sf::Event input)
  Takes an event, that is supposed to be a keypress, and acts accrodingly: checks if the box is selected and if it is the input is checked. If the input is semi-logical (unicode < 128) and the character limit is not exceeded, the character is added to the box. Backspace key deletes the previous inputted character instead of adding it to the box.

4.38.1 Detailed Description

Implements a class for creating textboxes that take inputs into SFML-window.

4.38.2 Constructor & Destructor Documentation

4.38.2.1 Textbox()

```
Textbox::Textbox (
            int size,
            sf::Color color,
            bool isSelected,
            int limit,
            std::string preText )  [inline]
```

Construct a new Textbox object.

Parameters

size	Size of the text
color	Color of the text
isSelected	A boolean value telling if the box is selected
limit	Limit to the the user can input into the box
preText	A non modifiable text in the box before the input space

4.38.3 Member Function Documentation

4.38.3.1 draw()

```
void Textbox::draw (
            sf::RenderWindow & window )  [inline]
```

Renders a box on a specified window.

Parameters

window	a reference to the window

4.38.3.2 getInput()

```
std::string Textbox::getInput ( )  [inline]
```

Get the text typed into the textbox.

Returns

std::string of the input text

4.38.3.3 onButton()

```
bool Textbox::onButton (
            sf::RenderWindow & window )  [inline]
```

Checks if the users mouse is on the textbox.

Parameters

window	reference to the window the textbox is on

Returns

true value if mouse is on the box

false value if mouse is not on the box

4.38.3.4 setColor()

```
void Textbox::setColor (
            sf::Color textColor )  [inline]
```

Set the color of the text inside the box.

Parameters

textColor	Color that the text is to be set to

4.38.3.5 setFont()

```
void Textbox::setFont (
            sf::Font & font )  [inline]
```

Set the font of the texbox text.

Parameters

font	a loaded font for the text

4.38.3.6 setPos()

```
void Textbox::setPos (
            sf::Vector2f pos )  [inline]
```

Sets the position of the box in the SFML window.

Parameters

pos	Position of the box

4.38.3.7 typedOn()

```
void Textbox::typedOn (
            sf::Event input )  [inline]
```

Takes an event, that is supposed to be a keypress, and acts accrodingly: checks if the box is selected and if it is the input is checked. If the input is semi-logical (unicode $<$ 128) and the character limit is not exceeded, the character is added to the box. Backspace key deletes the previous inputted character instead of adding it to the box.

Parameters

input	a sf::Event of what the user is doing eg. clicking a button or moving mouse

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

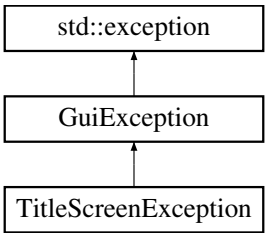
- interface/textbox.hpp

4.39 TitleScreenException Class Reference

Exception for unexpected behaviour in the title screen.

`#include <gui_ex.hpp>`

Inheritance diagram for TitleScreenException:



Public Member Functions

- virtual const char $*$ what () const noexcept

4.39.1 Detailed Description

Exception for unexpected behaviour in the title screen.

4.39.2 Member Function Documentation

4.39.2.1 what()

```
virtual const char * TitleScreenException::what ( ) const  [inline], [virtual], [noexcept]
```

Implements GuiException.

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

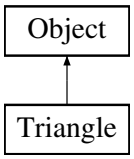
- interface/gui_ex.hpp

4.40 Triangle Class Reference

Representation of a mathematical triangle object in the scene.

`#include <triangle.hpp>`

Inheritance diagram for Triangle:



Public Member Functions

- Triangle (Vector v0, Vector v1, Vector v2, std::shared_ptr< Material > m)
  Construct a new Triangle object.
- void collision (Ray &ray, Hit &rayHit, float &smallestDistance)
  Calculate whether a given ray collides with the triangle using the Möller-Trumbore algorithm.
- void printInfo (std::ostream &out) const
  Print triangle info to the desired output stream.
- std::vector< Vector > getVertexPos () const
  Get the vertex positions of the triangle.
- std::vector< Vector > getPlaneVec () const
  Get the plane vectors of the triangle.
- Vector getNormal () const
  Get the normal of the triangle.
- Vector getCentroid () const
  Get the centroid of the triangle.

Public Member Functions inherited from Object

- **Object** (Vector position, std::shared_ptr< Material > material)
- Vector **getPosition** () const
- std::shared_ptr< Material > **getMaterial** () const

4.40.1 Detailed Description

Representation of a mathematical triangle object in the scene.

4.40.2 Constructor & Destructor Documentation

4.40.2.1 Triangle()

```
Triangle::Triangle (
            Vector v0,
            Vector v1,
            Vector v2,
            std::shared_ptr< Material > m )  [inline]
```

Construct a new Triangle object.

Parameters

v0	Vertex 1
v1	Vertex 2
v2	Vertex 3
m	Material of the triangle

4.40.3 Member Function Documentation

4.40.3.1 collision()

```
void Triangle::collision (
            Ray & ray,
            Hit & rayHit,
            float & smallestDistance )  [inline], [virtual]
```

Calculate whether a given ray collides with the triangle using the Möller-Trumbore algorithm.

If the ray collides with the triangle and the collision is closer than the current smallest distance, the "rayHit" data structure will be updated according to the collision.

Parameters

ray	ray whose collision will be checked
rayHit	address of a Hit data structure
smallestDistance	current smallest distance

Implements Object.

4.40.3.2 getCentroid()

```
Vector Triangle::getCentroid ( ) const  [inline]
```

Get the centroid of the triangle.

Returns

The centroid vector of the triangle

4.40.3.3 getNormal()

```
Vector Triangle::getNormal ( ) const  [inline]
```

Get the normal of the triangle.

Returns

The normal vector of the triangle

4.40.3.4 getPlaneVec()

```
std::vector< Vector > Triangle::getPlaneVec ( ) const  [inline]
```

Get the plane vectors of the triangle.

Returns

std::vector<Vector> containing the plane vectors

4.40.3.5 getVertexPos()

```
std::vector< Vector > Triangle::getVertexPos ( ) const  [inline]
```

Get the vertex positions of the triangle.

Returns

std::vector<Vector> containing the vertice vectors

4.40.3.6 printInfo()

```
void Triangle::printInfo (
            std::ostream & out ) const  [inline], [virtual]
```

Print triangle info to the desired output stream.

Parameters

out	output stream

Implements Object.

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

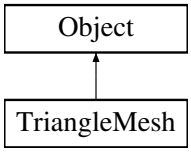
- objects/triangle.hpp

4.41 TriangleMesh Class Reference

Trianglemesh object consiting of Triangles, loaded from .obj file.

`#include <trianglemesh.hpp>`

Inheritance diagram for TriangleMesh:



Public Member Functions

- TriangleMesh (std::string obj_filepath, Vector scenePos, std::shared_ptr< Material > m, Vector rotation, double scale)

  Construct a new Triangle Mesh object.
- void collision (Ray &ray, Hit &rayHit, float &smallestDistance)

  Calclute whether a given ray collides with the TriangleMesh.
- void printInfo (std::ostream &out) const

  Print TriangleMesh info to the desired output stream.
- BVH getBVH () const

  Get the BVH of the trianglemesh object.
- std::string getName () const

  Get the name of the trianglemesh object.

Public Member Functions inherited from Object

- **Object** (Vector position, std::shared_ptr< Material > material)
- Vector **getPosition** () const
- std::shared_ptr< Material > **getMaterial** () const

4.41.1 Detailed Description

Trianglemesh object consiting of Triangles, loaded from .obj file.

4.41.2 Constructor & Destructor Documentation

4.41.2.1 TriangleMesh()

```
TriangleMesh::TriangleMesh (
            std::string obj_filepath,
            Vector scenePos,
            std::shared_ptr< Material > m,
            Vector rotation,
            double scale )  [inline]
```

Construct a new Triangle Mesh object.

Parameters

obj_filepath	filepath string
scenePos	position of the object in the scene
m	material of the object
scale	scaling of the object size
angle	counterclockwise rotation of the object in radians

4.41.3 Member Function Documentation

4.41.3.1 collision()

```
void TriangleMesh::collision (
            Ray & ray,
            Hit & rayHit,
            float & smallestDistance )  [inline], [virtual]
```

Calclute whether a given ray collides with the TriangleMesh.

Parameters

ray	ray whose collision will be checked
rayHit	address of a Hit data structure
smallestDistance	current smallest distance

Implements Object.

4.41.3.2 getBVH()

```
BVH TriangleMesh::getBVH ( ) const  [inline]
```

Get the BVH of the trianglemesh object.

Returns

BVH

4.41.3.3 getName()

```
std::string TriangleMesh::getName ( ) const  [inline]
```

Get the name of the trianglemesh object.

Returns

std::string

4.41.3.4 printInfo()

```
void TriangleMesh::printInfo (
            std::ostream & out ) const  [inline], [virtual]
```

Print TriangleMesh info to the desired output stream.

Parameters

out	output stream

Implements Object.

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

- objects/trianglemesh.hpp

Chapter 5

File Documentation

5.1 fileloader.hpp

```
00001 #pragma once
00002
00003 #include <yaml-cpp/yaml.h>
00004 #include <string>
00005 #include <exception>
00006 #include <memory>
00007 #include <sys/stat.h>
00008
00009 #include "trianglemesh.hpp"
00010 #include "box.hpp"
00011 #include "rectangle.hpp"
00012 #include "ball.hpp"
00013 #include "scene.hpp"
00014 #include "types.hpp"
00015 #include "fileloader_ex.hpp"
00016 #include "material.hpp"
00017
00023 class FileLoader {
00024     public:
00025
00032         FileLoader(std::string filepath) {
00033             // Check the existence of the given file
00034             struct stat buf;
00035             if (stat(filepath.c_str(), &buf) == 0)
00036             {
00037                 filepath_ = filepath;
00038             }
00039             else
00040             {
00041                 throw InvalidFilepathException(filepath);
00042             }
00043         }
00044
00050         std::shared_ptr<Scene> loadSceneFile() {
00051             Camera camera = LoadCamera();
00052             std::list<std::shared_ptr<Object» objects = LoadObjects();
00053             scene_ = std::make_shared<Scene>(camera, objects);
00054             LoadEnvironment(scene_);
00055             return scene_;
00056         }
00057
00061         ~FileLoader() {}
00062
00063         // Getter for filepath
00064         std::string getFilepath() { return filepath_; }
00065
00066
00067     private:
00068
00075         void LoadEnvironment(std::shared_ptr<Scene> scene) {
00076             if(YAML::LoadFile(filepath_)["Environment"]) {
00077                 YAML::Node environment_node = loadParams("Environment");
00078                 Eigen::Vector3d skyColor = LoadVector(environment_node, "SkyColor");
00079                 Eigen::Vector3d horizonColor = LoadVector(environment_node, "HorizonColor");
00080                 Eigen::Vector3d groundColor = LoadVector(environment_node, "GroundColor");
00081                 (*scene).getEnvironment().setSky(skyColor, horizonColor, groundColor);
00082             }else {
00083                 (*scene).getEnvironment().setSky();
```

```
00084             }
00085         }
00086
00093         Eigen::Vector3d LoadVector(YAML::Node node, std::string key) {
00094             Eigen::Vector3d vector;
00095             YAML::Node coords = node[key];
00096             // Throw exception if incorrect size or if not defined
00097             if (!coords.IsDefined()) {
00098                 throw InvalidKeyException(filepath_, key);
00099             }
00100             size_t size = coords.size();
00101             if (size != 3) {
00102                 throw InvalidSizeVectorException(filepath_, size, coords.Mark().line);
00103             }
00104             int i = 0;
00105             for (YAML::const_iterator it=coords.begin(); it!=coords.end(); ++it) {
00106                 vector[i] = it->as<float>();
00107                 i++;
00108             }
00109             return vector;
00110         }
00111
00119         std::shared_ptr<Material> LoadMaterial(YAML::Node node) {
00120             //Material material
00121             YAML::Node material_node = node["Material"];
00122             if (!material_node.IsDefined()) {
00123                 throw MaterialNotFoundException(filepath_, node.Mark().line);
00124             }
00125
00126             // Initialize default values
00127             Color color = Color(1.0, 1.0, 1.0);
00128             Color emission_color = Color(1.0, 1.0, 1.0);
00129             float emission_strength = 0.0;
00130             std::string name = "UNDEFINED";
00131             float specularity = 1.0;
00132             float clearcoat = 0.5;
00133             float refraction_ratio = 1.0;
00134             Color clearCoat_color = Color(1.0, 1.0, 1.0);
00135
00136             // Update the parameters based on .yaml file if the parameters are defined
00137             if(material_node["Color"]) {
00138                 color = LoadVector(material_node, "Color");
00139             }
00140             if(material_node["EmissionColor"]) {
00141                 emission_color = LoadVector(material_node, "EmissionColor");
00142             }
00143             if(material_node["EmissionStrength"]) {
00144                 emission_strength = material_node["EmissionStrength"].as<float>();
00145             }
00146             if(material_node["Specularity"]) {
00147                 specularity = material_node["Specularity"].as<float>();
00148             }
00149             if(material_node["Name"]) {
00150                 name = material_node["Name"].as<std::string>();
00151             }
00152             if(material_node["ClearCoat"]) {
00153                 clearcoat = material_node["ClearCoat"].as<float>();
00154             }
00155             if(material_node["ClearCoatColor"]) {
00156                 clearCoat_color = LoadVector(material_node, "ClearCoatColor");
00157             }
00158             if(material_node["RefractionRatio"]) {
00159                 refraction_ratio = material_node["RefractionRatio"].as<float>();
00160             }
00161
00162             // Return shared pointer to the correct material type or throw exception
00163             std::string type = material_node["Type"].as<std::string>();
00164             if (type == "Diffuse") {
00165                 return std::make_shared<Diffuse>(color, name, emission_strength, emission_color);
00166             }
00167             else if (type == "Reflective") {
00168                 return std::make_shared<Reflective>(color, name, specularity);
00169             }
00170             else if (type == "ClearCoat") {
00171                 return std::make_shared<ClearCoat>(color, name, specularity, clearcoat,
    clearCoat_color);
00172             }
00173             else if (type == "Refractive") {
00174                 return std::make_shared<Refractive>(color, name, refraction_ratio);
00175             }
00176             else {
00177                 throw InvalidMaterialTypeException(filepath_, material_node.Mark().line);
00178             }
00179         }
00180
00187         std::shared_ptr<Ball> LoadBall(YAML::Node ball) {
00188             YAML::Node radius_node = ball["Radius"];
```

```
00189             if (!radius_node.IsDefined()) {
00190                 throw RadiusNotFoundException(filepath_, ball.Mark().line);
00191             }
00192             float radius = radius_node.as<float>();
00193             if (radius < 0)
00194             {
00195                 throw NegativeRadiusException(filepath_, radius, radius_node.Mark().line);
00196             }
00197             else
00198             {
00199                 std::shared_ptr<Ball> ball_ptr = std::make_shared<Ball>(LoadVector(ball, "Position"),
      radius, LoadMaterial(ball));
00200                 return ball_ptr;
00201             }
00202         }
00203
00210         std::shared_ptr<Box> LoadBox(YAML::Node box) {
00211             YAML::Node width_node = box["Width"];
00212             YAML::Node height_node = box["Height"];
00213             YAML::Node depth_node = box["Depth"];
00214
00215             if (!width_node.IsDefined() || !height_node.IsDefined() || !depth_node.IsDefined()) {
00216                 throw ParameterNotFoundException(filepath_, box.Mark().line);
00217             }
00218
00219             float width = width_node.as<float>();
00220             float height = height_node.as<float>();
00221             float depth = depth_node.as<float>();
00222
00223             if (width < 0)
00224             {
00225                 throw NegativeDimensionException(filepath_, width, width_node.Mark().line);
00226             }
00227
00228             if (height < 0)
00229             {
00230                 throw NegativeDimensionException(filepath_, height, height_node.Mark().line);
00231             }
00232
00233             if (depth < 0)
00234             {
00235                 throw NegativeDimensionException(filepath_, depth, depth_node.Mark().line);
00236             }
00237
00238             std::shared_ptr<Box> box_ptr = std::make_shared<Box>(LoadVector(box, "Position"), width,
      height, depth, LoadMaterial(box));
00239
00240             Vector rotation;
00241             try
00242             {
00243                 rotation = LoadVector(box, "Rotation");
00244             }
00245             catch(const InvalidKeyException& e)
00246             {
00247                 std::cout « e.what() « std::endl;
00248                 std::cout « "Standard rotation (0, 0, 0) will be used." « std::endl;
00249                 rotation = Vector(0, 0, 0);
00250             }
00251             rotation = (rotation / 180.0) * M_PI;
00252
00253             box_ptr->rotate(rotation[0], Vector::UnitX());
00254             box_ptr->rotate(rotation[1], Vector::UnitY());
00255             box_ptr->rotate(rotation[2], Vector::UnitZ());
00256
00257             return box_ptr;
00258         }
00259
00266         std::shared_ptr<TriangleMesh> LoadTriangleMesh(YAML::Node tmesh) {
00267             YAML::Node filepath_node = tmesh["Filepath"];
00268             YAML::Node scale_node = tmesh["Scale"];
00269
00270             if (!filepath_node.IsDefined() || !scale_node.IsDefined()) {
00271                 throw ParameterNotFoundException(filepath_, tmesh.Mark().line);
00272             }
00273
00274             std::string tmesh_filepath = filepath_node.as<std::string>();
00275
00276             struct stat buf;
00277             if (stat(tmesh_filepath.c_str(), &buf) != 0)
00278             {
00279                 throw InvalidFilepathException(tmesh_filepath);
00280             }
00281
00282             double scale = scale_node.as<double>();
00283
00284             if (scale < 0)
00285             {
```

```
00286                 throw NegativeDimensionException(filepath_, scale, scale_node.Mark().line);
00287             }
00288 
00289             Vector rotation;
00290             try
00291             {
00292                 rotation = LoadVector(tmesh, "Rotation");
00293             }
00294             catch(const InvalidKeyException& e)
00295             {
00296                 std::cout << e.what() << std::endl;
00297                 std::cout << "Standard rotation (0, 0, 0) will be used." << std::endl;
00298                 rotation = Vector(0, 0, 0);
00299             }
00300             rotation = (rotation / 180.0) * M_PI;
00301 
00302             std::shared_ptr<TriangleMesh> tmesh_ptr = std::make_shared<TriangleMesh>(tmesh_filepath,
      LoadVector(tmesh, "Position"), LoadMaterial(tmesh), rotation, scale);
00303 
00304             return tmesh_ptr;
00305         }
00306 
00313         std::shared_ptr<Rectangle> LoadRectangle(YAML::Node rect) {
00314             YAML::Node width_node = rect["Width"];
00315             YAML::Node height_node = rect["Height"];
00316 
00317             if (!width_node.IsDefined() || !height_node.IsDefined()) {
00318                 throw ParameterNotFoundException(filepath_, rect.Mark().line);
00319             }
00320 
00321             float width = width_node.as<float>();
00322             float height = height_node.as<float>();
00323 
00324             if (width < 0)
00325             {
00326                 throw NegativeDimensionException(filepath_, width, width_node.Mark().line);
00327             }
00328 
00329             if (height < 0)
00330             {
00331                 throw NegativeDimensionException(filepath_, height, height_node.Mark().line);
00332             }
00333 
00334             std::shared_ptr<Rectangle> rect_ptr = std::make_shared<Rectangle>(LoadVector(rect,
      "Position"), width, height, LoadMaterial(rect));
00335 
00336             Vector rotation;
00337             try
00338             {
00339                 rotation = LoadVector(rect, "Rotation");
00340             }
00341             catch(const InvalidKeyException& e)
00342             {
00343                 std::cout << e.what() << std::endl;
00344                 std::cout << "Standard rotation (0, 0, 0) will be used." << std::endl;
00345                 rotation = Vector(0, 0, 0);
00346             }
00347             rotation = (rotation / 180.0) * M_PI;
00348 
00349             rect_ptr->rotate(rotation[0], Vector::UnitX());
00350             rect_ptr->rotate(rotation[1], Vector::UnitY());
00351             rect_ptr->rotate(rotation[2], Vector::UnitZ());
00352 
00353             return rect_ptr;
00354         }
00355 
00361         std::list<std::shared_ptr<Object>> LoadObjects() {
00362             YAML::Node objects = loadParams("Objects");
00363             std::list<std::shared_ptr<Object>> object_list;
00364             for (YAML::const_iterator it=objects.begin(); it!=objects.end(); ++it) {
00365                 if((*it)["Object"]["Type"].as<std::string>() == "Ball") {
00366                     object_list.push_back(LoadBall((*it)["Object"]));
00367                 }
00368                 if((*it)["Object"]["Type"].as<std::string>() == "Box") {
00369                     object_list.push_back(LoadBox((*it)["Object"]));
00370                 }
00371                 if((*it)["Object"]["Type"].as<std::string>() == "TriangleMesh") {
00372                     object_list.push_back(LoadTriangleMesh((*it)["Object"]));
00373                 }
00374                 if((*it)["Object"]["Type"].as<std::string>() == "Rectangle") {
00375                     object_list.push_back(LoadRectangle((*it)["Object"]));
00376                 }
00377             }
00378             return object_list;
00379         }
00380 
00386         Camera LoadCamera() {
```

```
00387             YAML::Node params = loadParams("Camera");
00388
00389             YAML::Node angle_node = params["Angle"];
00390             float angle = angle_node.IsDefined() ? angle_node.as<float>() : 0; // Set angle to zero if
    it not defined in yaml
00391
00392             YAML::Node DoF_node = params["DepthOfField"];
00393             float DoF = DoF_node.IsDefined() ? DoF_node.as<float>() : 0; // Set depth of field to zero
    if not defined in yaml
00394
00395             float fow = params["Fov"].as<float>();
00396             float focus = params["FocusDistance"].as<float>();
00397             if (fow < 0)
00398             {
00399                 throw NegativeFOVException(filepath_, fow, params["Fov"].Mark().line);
00400             }
00401             else if (focus < 0)
00402             {
00403                 throw NegativeFocusException(filepath_, focus, params["FocusDistance"].Mark().line);
00404             }
00405             Camera camera;
00406             camera.position = LoadVector(params, "Position");
00407             camera.lookingAt = LoadVector(params, "LookingAt");
00408             camera.direction = (camera.lookingAt - camera.position).normalized();
00409             Eigen::AngleAxisd rotation(angle*M_PI/180, camera.direction); // Rotation around the axis
    of camera looking direction
00410             Vector left = Vector(-camera.direction[1], camera.direction[0], 0).normalized(); // Vector
    towards left of the image plane (90 degrees with respect to cam dir)
00411             camera.left = rotation * left; // Rotate the camera's left direction according to rotation
00412             camera.up = camera.direction.cross(camera.left); // Up direction dynamically determined
    from camera direction and left direction
00413             camera.fov = M_PI * fow;
00414             camera.focus_distance = focus;
00415             camera.DoF = DoF;
00416             return camera;
00417         }
00418
00424         YAML::Node loadParams(std::string key) {
00425             YAML::Node params = YAML::LoadFile(filepath_)[key];
00426             if (!params.IsDefined()) {
00427                 throw InvalidKeyException(filepath_, key);
00428             }
00429             else
00430             {
00431                 return params;
00432             }
00433         }
00434
00435
00436
00437         std::string filepath_;
00438         std::shared_ptr<Scene> scene_;
00439 };
```

5.2 fileloader_ex.hpp

```
00001 #ifndef FILELOADER_EXCEPTION
00002 #define FILELOADER_EXCEPTION
00003
00004 #include <exception>
00005 #include <iostream>
00006 #include <string>
00007
00012 class FileLoaderException : public std::exception {
00013     public:
00014         FileLoaderException() {}
00015
00019         virtual const char* what() const noexcept = 0;
00020
00021         virtual ~FileLoaderException() = default;
00022
00023 };
00024
00029 class InvalidFilepathException : public FileLoaderException {
00030     public:
00036         InvalidFilepathException(std::string filepath) : FileLoaderException() {
00037             msg_ = "FileLoader exception caught:\nInvalid filepath: " + filepath;
00038         }
00039
00045         virtual const char* what() const noexcept {
00046             return msg_.c_str();
00047         }
00048
```

```
00049     private:
00050         std::string msg_;
00051
00052 };
00053
00058 class NegativeRadiusException : public FileLoaderException {
00059     public:
00067         NegativeRadiusException(std::string filepath, float radius, int line) : FileLoaderException()
      {
00068             msg_ = "FileLoader exception caught:\nNegative radius (" +
00069                     std::to_string(radius) + ") in file: " +
00070                     filepath + ", on line: " + std::to_string(line);
00071         }
00072
00078         virtual const char* what() const noexcept {
00079             return msg_.c_str();
00080         }
00081
00082     private:
00083         std::string msg_;
00084
00085 };
00086
00091 class InvalidSizeVectorException : public FileLoaderException {
00092     public:
00100         InvalidSizeVectorException(std::string filepath, size_t size, int line) :
      FileLoaderException() {
00101             msg_ = "FileLoader exception caught:\nInvalid vector of size " +
00102                     std::to_string(size) + " in file: " + filepath +
00103                     ", on line: " + std::to_string(line);
00104         }
00105
00111         virtual const char* what() const noexcept {
00112             return msg_.c_str();
00113         }
00114
00115     private:
00116         std::string msg_;
00117
00118 };
00119
00124 class NegativeFOVException : public FileLoaderException {
00125     public:
00133         NegativeFOVException(std::string filepath, float fow, int line) : FileLoaderException() {
00134             msg_ = "FileLoader exception caught:\nNegative FOW: " + std::to_string(fow) +
00135                     ", in file: " + filepath + ", on line: " + std::to_string(line);
00136         }
00137
00143         virtual const char* what() const noexcept {
00144             return msg_.c_str();
00145         }
00146
00147     private:
00148         std::string msg_;
00149
00150 };
00151
00156 class NegativeFocusException : public FileLoaderException {
00157     public:
00165         NegativeFocusException(std::string filepath, float focus, int line) : FileLoaderException() {
00166             msg_ = "FileLoader exception caught:\nNegative focus distance: " +
00167                     std::to_string(focus) + " in file: " + filepath +
00168                     ", on line: " + std::to_string(line);
00169         }
00170
00176         virtual const char* what() const noexcept {
00177             return msg_.c_str();
00178         }
00179
00180     private:
00181         std::string msg_;
00182
00183 };
00184
00189 class InvalidKeyException : public FileLoaderException {
00190     public:
00197         InvalidKeyException(std::string filepath, std::string key) : FileLoaderException() {
00198             msg_ = "FileLoader exception caught:\nInvalid key: " + key + ", for file: " + filepath;
00199         }
00200
00206         virtual const char* what() const noexcept {
00207             return msg_.c_str();
00208         }
00209
00210     private:
00211         std::string msg_;
00212
```

```
00213 };
00214
00219 class MaterialNotFoundException : public FileLoaderException {
00220     public:
00227         MaterialNotFoundException(std::string filepath, int line) : FileLoaderException() {
00228             msg_ = "FileLoader exception caught:\nMaterial not defined for object at line: " +
00229                     std::to_string(line) + ", in file: " + filepath + ".";
00230         }
00231
00237         virtual const char* what() const noexcept {
00238             return msg_.c_str();
00239         }
00240
00241     private:
00242         std::string msg_;
00243 };
00244
00249 class RadiusNotFoundException : public FileLoaderException {
00250     public:
00257         RadiusNotFoundException(std::string filepath, int line) : FileLoaderException() {
00258             msg_ = "FileLoader exception caught:\nRadius not defined for ball at line: " +
00259                     std::to_string(line) + ", in file: " + filepath + ".";
00260         }
00261
00267         virtual const char* what() const noexcept {
00268             return msg_.c_str();
00269         }
00270
00271     private:
00272         std::string msg_;
00273 };
00274
00279 class ParameterNotFoundException : public FileLoaderException {
00280     public:
00287         ParameterNotFoundException(std::string filepath, int line) : FileLoaderException() {
00288             msg_ = "FileLoader exception caught:\nMissing parameters in line: " +
00289                     std::to_string(line) + ", in file: " + filepath + ".";
00290         }
00291
00297         virtual const char* what() const noexcept {
00298             return msg_.c_str();
00299         }
00300
00301     private:
00302         std::string msg_;
00303 };
00304
00309 class NegativeDimensionException : public FileLoaderException {
00310     public:
00318         NegativeDimensionException(std::string filepath, float value, int line) :
      FileLoaderException() {
00319             msg_ = "FileLoader exception caught:\nNegative dimension: " +
00320                     std::to_string(value) + " in file: " + filepath +
00321                     ", on line: " + std::to_string(line);
00322         }
00323
00329         virtual const char* what() const noexcept {
00330             return msg_.c_str();
00331         }
00332
00333     private:
00334         std::string msg_;
00335
00336 };
00337
00342 class InvalidMaterialTypeException : public FileLoaderException {
00343     public:
00350         InvalidMaterialTypeException(std::string filepath, int line) : FileLoaderException() {
00351             msg_ = "FileLoader exception caught:\nInvalid material type in file: " +
00352                     filepath + ", for material starting on line: " + std::to_string(line) + ".";
00353         }
00354
00360         virtual const char* what() const noexcept {
00361             return msg_.c_str();
00362         }
00363
00364     private:
00365         std::string msg_;
00366 };
00367
00368
00369 #endif
```

5.3 button.hpp

```
00001 #pragma once
00002
00003 #include <iostream>
00004 #include <SFML/Graphics.hpp>
00009 class Button {
00010     public:
00011
00021         Button(std::string button_text, sf::Color textColor, int text_size, sf::Vector2f button_size,
      sf::Color button_color)  {
00022             button_text_.setString(button_text);
00023             button_text_.setFillColor(textColor);
00024             button_text_.setCharacterSize(text_size);
00025
00026             button_.setFillColor(button_color);
00027             button_.setSize(button_size);
00028         }
00029
00034         ~Button() = default;
00035
00036
00042         void setPos(sf::Vector2f pos) {
00043             button_.setPosition(pos);
00044
00045             float text_pos_x = (pos.x + button_.getLocalBounds().width / 2) -
      (button_text_.getGlobalBounds().width / 2);
00046             float text_pos_y = (pos.y + button_.getLocalBounds().height / 2) -
      (button_text_.getGlobalBounds().height / 2);
00047             button_text_.setPosition({text_pos_x, text_pos_y});
00048         }
00049
00055         void setColor(sf::Color color) {
00056             button_.setFillColor(color);
00057         }
00058
00064         void setFont(sf::Font &font) {
00065             button_text_.setFont(font);
00066         }
00067
00073         void draw(sf::RenderWindow &window) {
00074             window.draw(button_);
00075             window.draw(button_text_);
00076         }
00077
00085         bool onButton(sf::RenderWindow &window) {
00086             float mousePos_x = sf::Mouse::getPosition(window).x;
00087             float mousePos_y = sf::Mouse::getPosition(window).y;
00088             float butPos_x = button_.getPosition().x;
00089             float butPos_y = button_.getPosition().y;
00090             float butSize_x = button_.getSize().x;
00091             float butSize_y = button_.getSize().y;
00092
00093             bool onButton_x = (mousePos_x >= butPos_x) && (mousePos_x <= (butSize_x + butPos_x));
00094             bool onButton_y = (mousePos_y >= butPos_y) && (mousePos_y <= (butSize_y + butPos_y));
00095
00096             return (onButton_x && onButton_y);
00097
00098         }
00099
00100
00101     private:
00102     sf::Text button_text_;
00103     sf::RectangleShape button_;
00104 };
```

5.4 gui.hpp

```
00001 #pragma once
00002
00003 #include <SFML/Graphics.hpp>
00004 #include "renderer.hpp"
00005 #include "button.hpp"
00006 #include "textbox.hpp"
00007 #include "fileloader.hpp"
00008 #include "types.hpp"
00009 #include "gui_ex.hpp"
00010
00015 class Gui : public Interface {
00016 public:
00017
00025     std::shared_ptr<Scene> titleScreen() {
00026         sf::Font arial;
```

```
00027         std::string arialpath = "../fonts/Arial.ttf";
00028         if(!arial.loadFromFile(arialpath)) {
00029             throw FontException(arialpath);
00030         }
00031         sf::Font comic;
00032         std::string comicpath = "../fonts/ComicSansMS.ttf";
00033         if(!comic.loadFromFile(comicpath)) {
00034             throw FontException(comicpath);
00035         }
00036 
00037         selectedBox = nullptr;
00038 
00039         sf::RenderWindow window(sf::VideoMode(700, 400), "Path Tracer", sf::Style::Titlebar |
      sf::Style::Close);
00040 
00041         //Creates all UI elements
00042         sf::Text title;
00043         title.setFillColor(sf::Color::White);
00044         title.setFont(comic);
00045         title.setCharacterSize(50);
00046         title.setString("Path Tracer");
00047         title.setPosition({175, 150});
00048 
00049         sf::Text invalidPath;
00050         invalidPath.setFillColor(sf::Color::Red);
00051         invalidPath.setFont(arial);
00052         invalidPath.setCharacterSize(25);
00053         invalidPath.setString("Invalid filepath");
00054         invalidPath.setPosition({175, 250});
00055 
00056         bool error = false;
00057 
00058         Textbox filepathBox(25, sf::Color::White, false, 30, "Enter filename: ");
00059         filepathBox.setFont(arial);
00060         filepathBox.setPos({175, 225});
00061 
00062         while(window.isOpen()) {
00063             sf::Event event;
00064             //Checks user created events constantly
00065             while(window.pollEvent(event)) {
00066                 switch(event.type) {
00067                     case sf::Event::Closed:
00068                         window.close();
00069                         break;
00070                     case sf::Event::MouseButtonPressed:
00071                         if(filepathBox.onButton(window)) {
00072                             filepathBox.setSelected();
00073                             filepathBox.setColor(sf::Color::Green);
00074                         }
00075                         break;
00076                     case sf::Event::TextEntered:
00077                         filepathBox.typedOn(event);
00078                         break;
00079                     case sf::Event::KeyPressed:
00080                         if(event.key.code == sf::Keyboard::Enter){
00081                             try
00082                             {
00083                                 std::string filepath = filepathBox.getInput().substr();
00084                                 FileLoader loader(filepath);
00085                                 window.close();
00086                                 std::shared_ptr<Scene> loadedScene = loader.loadSceneFile();
00087                                 return loadedScene;
00088                             }
00089                             catch(FileLoaderException& ex) {
00090                                 error = true;
00091                             }
00092                         } else if(event.key.code == sf::Keyboard::Escape){
00093                             filepathBox.setColor(sf::Color::White);
00094                         }
00095                         break;
00096                     default:
00097                         break;
00098                 }
00099             }
00100 
00101             window.clear();
00102             window.draw(title);
00103             if(error) {
00104                 window.draw(invalidPath);
00105             }
00106             filepathBox.draw(window);
00107             window.display();
00108 
00109         }
00110         throw TitleScreenException();
00111     }
00112 
```

```
00113
00125     void openSettings(std::shared_ptr<Scene> loadedScene) {
00126         sf::Image image;
00127         sf::Sprite sprite;
00128         sf::Texture texture;
00129         sf::Font arial;
00130         std::string arialpath = "../fonts/Arial.ttf";
00131         if(!arial.loadFromFile(arialpath)) {
00132             throw FontException(arialpath);
00133         }
00134
00135         sf::RenderWindow window(sf::VideoMode(700, 400), "Path Tracer", sf::Style::Titlebar |
      sf::Style::Close);
00136
00137         //Creates all the UI elements
00138         sf::Vector2f size(200, 100);
00139         Button preview("Preview", sf::Color::Black, 20, size, sf::Color::Green);
00140         preview.setFont(arial);
00141         preview.setPos({0, 300});
00142
00143         Textbox resXbox(20, sf::Color::White, false, 4, "ResX: ");
00144         resXbox.setFont(arial);
00145         resXbox.setPos({0, 0});
00146
00147         Textbox resYbox(20, sf::Color::White, false, 4, "ResY: ");
00148         resYbox.setFont(arial);
00149         resYbox.setPos({0, 25});
00150
00151         Textbox sampleBox(20, sf::Color::White, false, 4, "Samples: ");
00152         sampleBox.setFont(arial);
00153         sampleBox.setPos({0, 50});
00154
00155         Textbox bounceBox(20, sf::Color::White, false, 4, "Light bounces: ");
00156         bounceBox.setFont(arial);
00157         bounceBox.setPos({0, 75});
00158
00159         Textbox fovBox(20, sf::Color::White, false, 4, "Fov: ");
00160         fovBox.setFont(arial);
00161         fovBox.setPos({0, 125});
00162
00163         Textbox dofBox(20, sf::Color::White, false, 4, "Depth of field: ");
00164         dofBox.setFont(arial);
00165         dofBox.setPos({0, 150});
00166
00167         Textbox focusBox(20, sf::Color::White, false, 4, "Focus distance: ");
00168         focusBox.setFont(arial);
00169         focusBox.setPos({0, 175});
00170
00171         sf::Text errorText;
00172         errorText.setFont(arial);
00173         errorText.setFillColor(sf::Color::Red);
00174         errorText.setCharacterSize(20);
00175         errorText.setPosition({0, 225});
00176
00177         selectedBox = nullptr;
00178
00179         while (window.isOpen())
00180         {
00181             sf::Event event;
00182             //Checks user created events constantly
00183             while (window.pollEvent(event))
00184             {
00185                 switch(event.type) {
00186                     case sf::Event::Closed:
00187                         window.close();
00188                         break;
00189                     case sf::Event::MouseMoved:
00190                         if(preview.onButton(window)) {
00191                             preview.setColor(sf::Color::White);
00192                         }else {
00193                             preview.setColor(sf::Color::Green);
00194                         }
00195                         break;
00196                     case sf::Event::MouseButtonPressed:
00197                         //Checks if preview button can be clicked
00198                         if(preview.onButton(window)) {
00199                             if(checkIfPosFloat(fovBox.getInput()) && fovBox.getInput() != ""){
00200                                 loadedScene->setFov((M_PI * std::stof(fovBox.getInput()) / 180));
00201                             }
00202
00203                             if(checkIfPosFloat(focusBox.getInput()) && focusBox.getInput() != ""){
00204                                 loadedScene->setFocusDist(std::stof(focusBox.getInput()));
00205                             }
00206
00207                             Renderer previewCreator(500, 400, loadedScene);
00208
00209                             if(checkIfPosFloat(dofBox.getInput()) && dofBox.getInput() != ""){
```

```
00210                                 previewCreator.setDof(std::stof(dofBox.getInput()));
00211                             }
00212                             createImg(previewCreator.parallelRender(1));
00213                             saveImage("preview.png");
00214                             image.loadFromFile("preview.png");
00215                             texture.loadFromImage(image);
00216                             sprite.setTexture(texture);
00217                             sprite.setPosition(200, 0);
00218                             }
00219                         //checks if any textboxes can be clicked
00220                         clickBox(resXbox, window);
00221                         clickBox(resYbox, window);
00222                         clickBox(sampleBox, window);
00223                         clickBox(bounceBox, window);
00224                         clickBox(fovBox, window);
00225                         clickBox(dofBox, window);
00226                         clickBox(focusBox, window);
00227                         break;
00228 
00229                     case sf::Event::TextEntered:
00230                         fovBox.typedOn(event);
00231                         resXbox.typedOn(event);
00232                         resYbox.typedOn(event);
00233                         dofBox.typedOn(event);
00234                         sampleBox.typedOn(event);
00235                         focusBox.typedOn(event);
00236                         bounceBox.typedOn(event);
00237                         break;
00238 
00239                     case sf::Event::KeyPressed:
00240                         if(event.key.code == sf::Keyboard::Enter){
00241                             int resX;
00242                             int resY;
00243                             int sampleSize;
00244                             int bounceAmount;
00245                             float dof = 0;
00246 
00247                             if(checkIfPosFloat(fovBox.getInput()) && fovBox.getInput() != ""){
00248                                 loadedScene->setFov((M_PI * std::stof(fovBox.getInput()) / 180));
00249                             }
00250 
00251                             if(checkIfPosFloat(focusBox.getInput()) && focusBox.getInput() != ""){
00252                                 loadedScene->setFocusDist(std::stof(focusBox.getInput()));
00253                             }
00254 
00255                             if(checkIfPosFloat(dofBox.getInput()) && dofBox.getInput() != ""){
00256                                 dof = std::stof(dofBox.getInput());
00257                             }
00258 
00259                             if(checkIfPosNum(resXbox.getInput()) && resXbox.getInput() != ""){
00260                                 resX = std::stof(resXbox.getInput());
00261                             }else {
00262                                 errorText.setString("Invalid resX");
00263                                 break;
00264                             }
00265 
00266                             if(checkIfPosNum(resYbox.getInput()) && resYbox.getInput() != ""){
00267                                 resY = std::stof(resYbox.getInput());
00268                             }else {
00269                                 errorText.setString("Invalid resY");
00270                                 break;
00271                             }
00272 
00273                             if(checkIfPosNum(sampleBox.getInput()) && sampleBox.getInput() != ""){
00274                                 sampleSize = std::stoi(sampleBox.getInput());
00275                             }else {
00276                                 errorText.setString("Invalid sample amount");
00277                                 break;
00278                             }
00279 
00280                             if(checkIfPosNum(bounceBox.getInput()) && bounceBox.getInput() != ""){
00281                                 bounceAmount = std::stoi(bounceBox.getInput());
00282                             }else {
00283                                 errorText.setString("Invalid bounce amount");
00284                                 break;
00285                             }
00286 
00287                             selectedBox = nullptr;
00288                             window.close();
00289                             openRender(resX, resY, loadedScene, sampleSize, dof, bounceAmount);
00290                             break;
00291                         }
00292                     default:
00293                         break;
00294 
00295                     }
00296                 }
```

```
00297
00298             window.clear();
00299             preview.draw(window);
00300             window.draw(sprite);
00301             window.draw(errorText);
00302             fovBox.draw(window);
00303             resXbox.draw(window);
00304             resYbox.draw(window);
00305             focusBox.draw(window);
00306             dofBox.draw(window);
00307             sampleBox.draw(window);
00308             bounceBox.draw(window);
00309             window.display();
00310
00311             }
00312
00313         }
00314
00315     void openRender(int resX, int resY, std::shared_ptr<Scene> loadedScene, int sampleSize, float dof,
    int bounces) {
00316         sf::RenderWindow window(sf::VideoMode(resX, resY), "Path Tracer", sf::Style::Close);
00317         window.setSize(sf::Vector2u(resX, resY));
00318         window.setFramerateLimit(0);
00319         sf::Image image;
00320         sf::Sprite sprite;
00321         sf::Texture texture;
00322         Renderer sceneRenderer(resX, resY, loadedScene);
00323         sceneRenderer.setMaxBounces(bounces);
00324         sceneRenderer.setDof(dof);
00325         std::vector<std::vector<Color> combinedSamples;
00326
00327         int i = 0;
00328
00329             while (window.isOpen())
00330             {
00331                 sf::Event event;
00332                 while (window.pollEvent(event))
00333                 {
00334                     switch(event.type) {
00335                         case sf::Event::Closed:
00336                             window.close();
00337                             break;
00338                         default:
00339                             break;
00340                     }
00341                 }
00342                 while(i < sampleSize) {
00343                     auto result = sceneRenderer.parallelRender(1);
00344                     float weight = 1.0 / (1 + i);
00345                     if(i == 0) {
00346                         combinedSamples = result;
00347                     } else {
00348                         for (int pixel = 0; pixel < resX * resY; ++pixel)
00349                         {
00350                             int x = pixel % resX;
00351                             int y = pixel / resX;
00352
00353                             combinedSamples[x][y] = clamp(combinedSamples[x][y] * (1 - weight) +
    weight * result[x][y]);
00354
00355                         }
00356                     }
00357                     createImg(combinedSamples);
00358                     saveImage("image.png");
00359                     image.loadFromFile("image.png");
00360                     texture.loadFromImage(image);
00361                     sprite.setTexture(texture);
00362                     window.clear();
00363                     window.draw(sprite);
00364                     window.display();
00365                     i++;
00366
00367
00368                 }
00369         }
00370
00371         }
00372
00373 private:
00374     Textbox* selectedBox = nullptr;
00375
00383     bool checkIfPosNum(std::string text) {
00384         for(auto i : text) {
00385             if(!std::isdigit(i)) {
00386                 return false;
00387             }
00388         }
```

```
00389         return true;
00390     }
00391
00399     bool checkIfPosFloat(std::string text) {
00400         int j = text.length();
00401         for(auto i : text) {
00402             if(!std::isdigit(i)) {
00403                 if(j != 2 && std::to_string(text[text.length()-2]) != ".") {
00404                     return false;
00405                 }
00406             }
00407             j--;
00408         }
00409         return true;
00410     }
00411
00421     void clickBox(Textbox &box, sf::RenderWindow &window) {
00422         if(box.onButton(window)) {
00423              if(selectedBox) {
00424                 selectedBox->setUnselected();
00425                 selectedBox->setColor(sf::Color::White);
00426                 }
00427                 box.setSelected();
00428                 box.setColor(sf::Color::Green);
00429                 selectedBox = &box;
00430                 }
00431     }
00432
00439     Color clamp(Color input) {
00440         float R = input(0) > 1 ? 1 : input(0);
00441         float G = input(1) > 1 ? 1 : input(1);
00442         float B = input(2) > 1 ? 1 : input(2);
00443         return Color(R, G, B);
00444     }
00445 };
```

5.5 gui_ex.hpp

```
00001 #ifndef GUI_EXCEPTION
00002 #define GUI_EXCEPTION
00003
00004 #include <exception>
00005 #include <iostream>
00006 #include <string>
00007
00012 class GuiException : public std::exception {
00013     public:
00014         GuiException() {}
00015
00016         virtual const char* what() const noexcept = 0;
00017
00018         virtual ~GuiException() = default;
00019 };
00020
00025 class TitleScreenException : public GuiException {
00026     public:
00027         TitleScreenException() : GuiException() {
00028             msg_ = "Title window closed";
00029         }
00030
00031         virtual const char* what() const noexcept {
00032             return msg_.c_str();
00033         }
00034
00035     private:
00036         std::string msg_;
00037 };
00038
00043 class FontException : public GuiException {
00044     public:
00045         FontException(std::string fontPath) : GuiException() {
00046             msg_ = "Did not find font in path: " + fontPath;
00047         }
00048
00049         virtual const char* what() const noexcept {
00050             return msg_.c_str();
00051         }
00052
00053     private:
00054         std::string msg_;
00055 };
00056
00057 #endif
```

5.6 interface.hpp

```
00001 #pragma once
00002
00003 #include <SFML/Graphics.hpp>
00004 #include <vector>
00005
00006 #include "types.hpp"
00007
00012 class Interface {
00013 private:
00014     sf::Image img;
00015
00021     sf::Uint8 scale(float x) {
00022         return floor(255*x);
00023     }
00024
00025 public:
00026
00032     void createImg(std::vector<std::vector<Color» pixels) {
00033         int width = pixels.size();
00034         int height = pixels[0].size();
00035         img.create(width, height);
00036         for (int i = 0; i < width; ++i) {
00037             for (int j = 0; j < height; ++j) {
00038                 const sf::Uint8 R = scale(pixels[i][j](0));
00039                 const sf::Uint8 G = scale(pixels[i][j](1));
00040                 const sf::Uint8 B = scale(pixels[i][j](2));
00041                 img.setPixel(i, j, sf::Color(R, G, B));
00042             }
00043         }
00044     }
00045
00051     bool saveImage(const std::string &filename) {
00052         return img.saveToFile(filename);
00053     }
00054
00055     sf::Image getImage() { return img; }
00056
00057 };
```

5.7 textbox.hpp

```
00001 #pragma once
00002
00003 #include <iostream>
00004 #include <sstream>
00005 #include <SFML/Graphics.hpp>
00006
00007 #define BACKSPACE_KEY 8
00008 #define ESCAPE_KEY 27
00009 #define TAB_KEY 9
00010 #define ENTER_KEY 11
00011
00016 class Textbox {
00017 public:
00018
00028     Textbox(int size, sf::Color color, bool isSelected, int limit, std::string preText) {
00029         textbox_.setCharacterSize(size);
00030         size_ = size;
00031         textbox_.setFillColor(color);
00032         textbox_.setString(preText);
00033         isSelected_ = isSelected;
00034         limit_ = limit + preText.length() - 1;
00035         preTextLength = preText.length();
00036         text_ « preText;
00037     }
00038
00043     ~Textbox() = default;
00044
00050     void setFont(sf::Font &font) {
00051         textbox_.setFont(font);
00052     }
00053
00058     void setSelected() {
00059         isSelected_ = true;
00060     }
00061
00066     void setUnselected() {
00067         isSelected_ = false;
00068     }
00069
00075     void setPos(sf::Vector2f pos) {
```

```
00076         textbox_.setPosition(pos);
00077     }
00078
00084     void setColor(sf::Color textColor) {
00085         textbox_.setFillColor(textColor);
00086     }
00087
00093     void draw(sf::RenderWindow &window) {
00094         window.draw(textbox_);
00095     }
00096
00104     bool onButton(sf::RenderWindow &window) {
00105             float mousePos_x = sf::Mouse::getPosition(window).x;
00106             float mousePos_y = sf::Mouse::getPosition(window).y;
00107             float boxPos_x = textbox_.getPosition().x;
00108             float boxPos_y = textbox_.getPosition().y;
00109             float boxSize_x = (size_/1.618)*limit_;
00110             float boxSize_y = size_;
00111
00112             bool onBox_x = (mousePos_x >= boxPos_x) && (mousePos_x <= (boxSize_x + boxPos_x));
00113             bool onBox_y = (mousePos_y >= boxPos_y) && (mousePos_y <= (boxSize_y + boxPos_y));
00114
00115             return (onBox_x && onBox_y);
00116
00117         }
00118
00124     std::string getInput() {
00125             return text_.str().substr(preTextLength);
00126         }
00127
00137     void typedOn(sf::Event input) {
00138         if(isSelected_ && input.key.code != sf::Keyboard::Enter) {
00139             int inputChar = input.text.unicode;
00140             if(inputChar < 128 && inputChar != 0 && inputChar != TAB_KEY && inputChar != ENTER_KEY) {
00141                 if(text_.str().length() <= limit_) {
00142                     checkInput(inputChar);
00143                 }else if(inputChar == BACKSPACE_KEY) {
00144                     deleteLastChar();
00145                 }
00146             }
00147         }
00148     }
00149
00150 private:
00151     sf::Text textbox_;
00152     std::ostringstream text_;
00153     bool isSelected_ = false;
00154     int limit_;
00155     int preTextLength;
00156     int size_;
00157
00166     void checkInput(int inputChar) {
00167         if(inputChar != BACKSPACE_KEY && inputChar != ESCAPE_KEY) {
00168             text_ << static_cast<char>(inputChar);
00169         }
00170         else if(inputChar == BACKSPACE_KEY) {
00171             deleteLastChar();
00172         }
00173         else if(inputChar == ESCAPE_KEY && isSelected_ == true) {
00174             setUnselected();
00175             textbox_.setFillColor(sf::Color::White);
00176         }
00177         textbox_.setString(text_.str());
00178     }
00179
00184     void deleteLastChar() {
00185         if(text_.str().length() > preTextLength){
00186             std::string newTxt = text_.str();
00187             newTxt.pop_back();
00188             text_.str(std::string());
00189             text_ << newTxt;
00190             textbox_.setString(text_.str());
00191         }
00192     }
00193
00194 };
```

5.8 ball.hpp

```
00001 #pragma once
00002
00003 #include "object.hpp"
00004 #include "fileloader_ex.hpp"
```

```
00005 #include "material.hpp"
00006
00007 #include <memory>
00008
00013 class Ball : public Object
00014 {
00015 private:
00016     float radius_;
00017
00018 public:
00019     Ball(Vector position, float radius, std::shared_ptr<Material> material) : Object(position,
      material), radius_(radius) {}
00020
00031     void collision(Ray& ray, Hit &rayHit, float& smallestDistance) {
00032
00033         Vector toBall = ray.origin - this->getPosition();
00034
00035         float a = ray.direction.dot(ray.direction);
00036         float b = 2 * ray.direction.dot(toBall);
00037         float c = toBall.dot(toBall) - radius_ * radius_;
00038
00039         float discriminant = b*b - 4*a*c;
00040
00041         if (discriminant >= 0)
00042         {
00043             float distance = (-b - sqrt(discriminant)) / (2*a);
00044
00045             if (distance > 0 && distance < smallestDistance)
00046             {
00047                 smallestDistance = distance;
00048                 rayHit.distance = distance;
00049                 rayHit.material = this->getMaterial();
00050                 rayHit.did_hit = true;
00051                 rayHit.point = ray.origin + ray.direction * distance;
00052                 rayHit.normal = (rayHit.point - this->getPosition()).normalized();
00053             }
00054         }
00055
00056         return;
00057     }
00058
00059     float getRadius() const { return radius_; }
00060
00067     void printInfo(std::ostream& out) const {
00068         out « "Ball at: (" « this->getPosition().transpose() « ") with radius: " « this->getRadius() «
      ", with material: " « (*getMaterial()).getName() « std::endl;
00069     }
00070
00071 };
```

5.9 box.hpp

```
00001 #pragma once
00002
00003 #include "object.hpp"
00004 #include <vector>
00005
00012 class Box : public Object
00013 {
00014 private:
00015     float width_; /* Associated with y direction */
00016     float height_; /* Associated with z direction */
00017     float depth_; /* Associated with x direction */
00018
00019     std::vector<Vector> corners_;
00020     /*
00021     *       4.......7
00022     *     / :     / :
00023     *   0-------3   :
00024     *   |   :   |   :
00025     *   |   5...|...6
00026     *   | /     | /
00027     *   1-------2
00028     *
00029     * When looking towards positive x-direction
00030     */
00031
00032    std::list<std::vector<int» sides_ = {   {0, 1, 2},
00033                                            {3, 2, 6},
00034                                            {7, 6, 5},
00035                                            {4, 5, 1},
00036                                            {4, 0, 3},
00037                                            {1, 5, 6}};
```

```
00038
00039 public:
00040     Box(Vector position, float width, float height, float depth, std::shared_ptr<Material> material)
00041             : Object(position, material), width_(width), height_(height), depth_(depth) {
00042
00043         corners_.push_back(Vector(-depth_/2, width_/2, height_/2) + position);
00044         corners_.push_back(Vector(-depth_/2, width_/2, -height_/2) + position);
00045         corners_.push_back(Vector(-depth_/2, -width_/2, -height_/2) + position);
00046         corners_.push_back(Vector(-depth_/2, -width_/2, height_/2) + position);
00047
00048         corners_.push_back(Vector(depth_/2, width_/2, height_/2) + position);
00049         corners_.push_back(Vector(depth_/2, width_/2, -height_/2) + position);
00050         corners_.push_back(Vector(depth_/2, -width_/2, -height_/2) + position);
00051         corners_.push_back(Vector(depth_/2, -width_/2, height_/2) + position);
00052     }
00053
00064     void collision(Ray& ray, Hit &rayHit, float& smallestDistance) {
00065
00066         // Check first whether ray collides with the bounding ball
00067         Vector toBall = ray.origin - this->getPosition();
00068         float radius = width_ * width_ + height_ * height_ + depth_ * depth_;
00069         float a = ray.direction.dot(ray.direction);
00070         float b = 2 * ray.direction.dot(toBall);
00071         float c = toBall.dot(toBall) - radius;
00072         float discriminant = b*b - 4*a*c;
00073         if (discriminant < 0) return;
00074
00075         // If inside ball, check sides
00076         for (auto side : sides_) {
00077
00078             Vector topLeft = corners_[side[0]];
00079             Vector bottomLeft = corners_[side[1]];
00080             Vector bottomRight = corners_[side[2]];
00081
00082             Vector s1 = -bottomLeft + bottomRight;
00083             Vector s2 = -bottomLeft + topLeft;
00084
00085             Vector normal = s1.cross(s2).normalized();
00086
00087             float distance = (bottomLeft - ray.origin).dot(normal) / ray.direction.dot(normal);
00088             Vector intersection = -bottomLeft + ray.origin + ray.direction * distance;
00089
00090             if (intersection.dot(s1) <= s1.squaredNorm()
00091                 && intersection.dot(s1) >= 0
00092                 && intersection.dot(s2) <= s2.squaredNorm()
00093                 && intersection.dot(s2) >= 0
00094             )
00095             {
00096                 if (distance > 0 && distance < smallestDistance)
00097                 {
00098                     smallestDistance = distance;
00099                     rayHit.distance = distance;
00100                     rayHit.material = this->getMaterial();
00101                     rayHit.did_hit = true;
00102                     rayHit.point = ray.origin + ray.direction * distance;
00103                     rayHit.normal = normal;
00104                 }
00105             }
00106         }
00107     }
00108
00109     float getWidth() const { return width_; }
00110     float getHeight() const { return height_; }
00111     float getDepth() const { return depth_; }
00112
00121     void rotate(float angle, Vector axis) {
00122         Eigen::AngleAxisd rotation(angle, axis);
00123
00124         for (auto& corner : corners_) {
00125             corner = (rotation * (corner - this->getPosition())) + this->getPosition();
00126         }
00127     }
00128
00135     void printInfo(std::ostream& out) const {
00136         out << "Box at: (" << this->getPosition().transpose() << ") with dimensions: " << width_ << "x" <<
      height_ << "x" << depth_ << ", with material: " << (*getMaterial()).getName() << std::endl;
00137     }
00138
00139 };
```

5.10 bvh.hpp

```
00001 #pragma once
```

```
00002
00003 #include "types.hpp"
00004 #include "triangle.hpp"
00005 #include <vector>
00006
00011 struct AABB
00012 {
00013     Vector min, max;
00014 };
00015
00020 struct Node
00021 {
00022     AABB box;
00023     int leftChild;
00024     bool isLeaf() { return triCount > 0; }
00025     int firstTriIdx, triCount;
00026 };
00027
00034 class BVH
00035 {
00036 public:
00041     BVH() { }
00042
00048     BVH(std::vector<Triangle> tris) {
00049         //Initialize variables
00050         n = tris.size();
00051         triangles = tris;
00052         std::vector<Node> bvhNodes(n*2 - 1);
00053         rootNodeIdx = 0;
00054         nodesUsed = 1;
00055         for(int i = 0; i < n; i++) {
00056             triIdx.push_back(i);
00057         }
00058         Node& root = bvhNodes[rootNodeIdx];
00059         root.leftChild = 0;
00060         root.triCount = n;
00061
00062         //Create the structure
00063         UpdateNodeBounds(bvhNodes, rootNodeIdx);
00064         Subdivide(bvhNodes, rootNodeIdx);
00065
00066         //Copy structure to a private variable
00067         for(int i = 0; i < bvhNodes.size(); i++) {
00068             nodes.push_back(bvhNodes[i]);
00069         }
00070     }
00071
00078     void UpdateNodeBounds(std::vector<Node>& bvhNodes, int nodeIdx) {
00079         Node& currentNode = bvhNodes[nodeIdx];
00080
00081         //Set bounds to "infinity" in the beginning
00082         currentNode.box.min = Vector(1e30f, 1e30f, 1e30f);
00083         currentNode.box.max = Vector(-1e30f, -1e30f, -1e30f);
00084
00085         //Loop over triangles and set new bounds accordingly
00086         for(int first = currentNode.firstTriIdx, i=0; i < currentNode.triCount; i++) {
00087             int leafTriIdx = triIdx[first + i];
00088             Triangle &leafTriangle = triangles[leafTriIdx];
00089             std::vector<Vector> vertices = leafTriangle.getVertexPos();
00090             currentNode.box.min = currentNode.box.min.cwiseMin(vertices[0]);
00091             currentNode.box.min = currentNode.box.min.cwiseMin(vertices[1]);
00092             currentNode.box.min = currentNode.box.min.cwiseMin(vertices[2]);
00093             currentNode.box.max = currentNode.box.max.cwiseMax(vertices[0]);
00094             currentNode.box.max = currentNode.box.max.cwiseMax(vertices[1]);
00095             currentNode.box.max = currentNode.box.max.cwiseMax(vertices[2]);
00096         }
00097     }
00098
00105     void Subdivide(std::vector<Node>& bvhNodes, int nodeIdx) {
00106         //Terminates recursion
00107         Node &currentNode = bvhNodes[nodeIdx];
00108         if(currentNode.triCount <= 2) return;
00109
00110         //Determine the split
00111         Vector extent = currentNode.box.max - currentNode.box.min;
00112         int axis = 0;
00113         if(extent(1) > extent(0)) axis = 1;
00114         if(extent(2) > extent(axis)) axis = 2;
00115         float splitPos = currentNode.box.min(axis) + extent(axis)*0.5f;
00116
00117         //Partition triangles
00118         int i = currentNode.firstTriIdx;
00119         int j = i + currentNode.triCount - 1;
00120         while(i <= j) {
00121             if(triangles[triIdx[i]].getCentroid()(axis) < splitPos) {
00122                 i++;
00123             }else {
```

```
00124                 std::swap(triIdx[i], triIdx[j--]);
00125             }
00126         }
00127 
00128         //Check if one of the sides is empty
00129         int leftCount = i - currentNode.firstTriIdx;
00130         if(leftCount == 0 || leftCount == currentNode.triCount) return;
00131 
00132         //create child nodes
00133         int leftChildIdx = nodesUsed++;
00134         int rightChildIdx = nodesUsed++;
00135         bvhNodes[leftChildIdx].firstTriIdx = currentNode.firstTriIdx;
00136         bvhNodes[leftChildIdx].triCount = leftCount;
00137         bvhNodes[rightChildIdx].firstTriIdx = i;
00138         bvhNodes[rightChildIdx].triCount = currentNode.triCount - leftCount;
00139         currentNode.leftChild = leftChildIdx;
00140         currentNode.triCount = 0;
00141 
00142         //Update node bounds
00143         UpdateNodeBounds(bvhNodes, leftChildIdx);
00144         UpdateNodeBounds(bvhNodes, rightChildIdx);
00145         //Recursion
00146         Subdivide(bvhNodes, leftChildIdx);
00147         Subdivide(bvhNodes, rightChildIdx);
00148     }
00149 
00158     void BVHCollision(Ray& ray, Hit& rayHit, float& smallestDistance, const int nodeIdx) {
00159         Node& currentNode = nodes[nodeIdx];
00160         if(!AABBCollision(currentNode.box, ray, smallestDistance)) return;
00161         if(currentNode.isLeaf()) {
00162             for(int i = 0; i < currentNode.triCount; i++) {
00163                 triangles[triIdx[currentNode.firstTriIdx + i]].collision(ray, rayHit,
      smallestDistance);
00164             }
00165         }else {
00166             BVHCollision(ray, rayHit, smallestDistance, currentNode.leftChild);
00167             BVHCollision(ray, rayHit, smallestDistance, currentNode.leftChild + 1);
00168         }
00169     }
00170 
00180     bool AABBCollision(AABB box, Ray ray, float smallestDistance) {
00181         float tx1 = (box.min(0) - ray.origin(0))/ray.direction(0);
00182         float tx2 = (box.max(0) - ray.origin(0))/ray.direction(0);
00183         float tmin = std::min(tx1, tx2);
00184         float tmax = std::max(tx1, tx2);
00185 
00186         float ty1 = (box.min(1) - ray.origin(1))/ray.direction(1);
00187         float ty2 = (box.max(1) - ray.origin(1))/ray.direction(1);
00188         tmin = std::max(tmin, std::min(ty1, ty2));
00189         tmax = std::min(tmax, std::max(ty1, ty2));
00190 
00191         float tz1 = (box.min(2) - ray.origin(2))/ray.direction(2);
00192         float tz2 = (box.max(2) - ray.origin(2))/ray.direction(2);
00193         tmin = std::max(tmin, std::min(tz1, tz2));
00194         tmax = std::min(tmax, std::max(tz1, tz2));
00195 
00196         return tmax >= tmin && tmin < smallestDistance && tmax > 0;
00197     }
00198 
00204     int getRootNodeIdx() const {
00205         return rootNodeIdx;
00206     }
00207 
00213     std::vector<Triangle> getTriangles() const {
00214         return triangles;
00215     }
00216 
00222     std::vector<Node> getNodes() const {
00223         return nodes;
00224     }
00225 
00226 
00227 private:
00228     std::vector<Triangle> triangles;
00229     std::vector<int> triIdx;
00230     std::vector<Node> nodes;
00231     int rootNodeIdx;
00232     int nodesUsed;
00233     int n;
00234 };
```

5.11 object.hpp

```
00001 #pragma once
```

```
00002
00003 #include "types.hpp"
00004 #include "material.hpp"
00005
00006 #include <memory>
00007
00012 class Object
00013 {
00014 private:
00015     Vector position_;
00016     std::shared_ptr<Material> material_;
00017     std::string name_;
00018
00019 public:
00020     Object(Vector position, std::shared_ptr<Material> material)
00021                     : position_(position), material_(material) {}
00022
00023     virtual ~Object() { }
00024
00025     Vector getPosition() const { return position_; }
00026     std::shared_ptr<Material> getMaterial() const { return material_; }
00027
00038     virtual void collision(Ray& ray, Hit &rayHit, float& smallestDistance) = 0;
00039
00047     virtual void printInfo(std::ostream& out) const = 0;
00048 };
00049
00059 std::ostream &operator«(std::ostream& out, const Object& object) {
00060     object.printInfo(out);
00061     return out;
00062 }
```

5.12 rectangle.hpp

```
00001 #pragma once
00002
00003 #include "object.hpp"
00004 #include <vector>
00005 #include <memory>
00006
00013 class Rectangle : public Object
00014 {
00015 private:
00016     float width_; /* Associated with y direction */
00017     float height_; /* Associated with z direction */
00018
00019     std::vector<Vector> corners_;
00020     /*
00021     *
00022     *   0-------3
00023     *   |       |
00024     *   |       |
00025     *   |       |
00026     *   1-------2
00027     *
00028     * When looking towards positive x-direction
00029     */
00030
00031 public:
00032     Rectangle(Vector position, float width, float height, std::shared_ptr<Material> material)
00033             : Object(position, material), width_(width), height_(height) {
00034
00035         corners_.push_back(Vector(0, width_/2, height_/2) + position);
00036         corners_.push_back(Vector(0, width_/2, -height_/2) + position);
00037         corners_.push_back(Vector(0, -width_/2, -height_/2) + position);
00038         corners_.push_back(Vector(0, -width_/2, height_/2) + position);
00039     }
00040
00051     void collision(Ray& ray, Hit &rayHit, float& smallestDistance) {
00052
00053         Vector topLeft = corners_[0];
00054         Vector bottomLeft = corners_[1];
00055         Vector bottomRight = corners_[2];
00056
00057         Vector s1 = -bottomLeft + bottomRight;
00058         Vector s2 = -bottomLeft + topLeft;
00059
00060         Vector normal = s1.cross(s2).normalized();
00061
00062         float distance = (bottomLeft - ray.origin).dot(normal) / ray.direction.dot(normal);
00063         Vector intersection = -bottomLeft + ray.origin + ray.direction * distance;
00064
00065         if (intersection.dot(s1) <= s1.squaredNorm()
```

```
00066             && intersection.dot(s1) >= 0
00067             && intersection.dot(s2) <= s2.squaredNorm()
00068             && intersection.dot(s2) >= 0
00069         )
00070         {
00071             if (distance > 0 && distance < smallestDistance)
00072             {
00073                 smallestDistance = distance;
00074                 rayHit.distance = distance;
00075                 rayHit.material = this->getMaterial();
00076                 rayHit.did_hit = true;
00077                 rayHit.point = ray.origin + ray.direction * distance;
00078                 rayHit.normal = normal;
00079             }
00080         }
00081     }
00082
00083     float getWidth() const { return width_; }
00084     float getHeight() const { return height_; }
00085
00094     void rotate(float angle, Vector axis) {
00095         Eigen::AngleAxisd rotation(angle, axis);
00096
00097         for (auto& corner : corners_) {
00098             corner = (rotation * (corner - this->getPosition())) + this->getPosition();
00099         }
00100     }
00101
00108     void printInfo(std::ostream& out) const {
00109         out « "Rectangle at: (" « this->getPosition().transpose() « ") with dimensions: " « width_ «
      "x" « height_ « ", with material: " « getMaterial()->getName() « std::endl;
00110     }
00111
00112 };
```

5.13 triangle.hpp

```
00001 #pragma once
00002
00003 #include "types.hpp"
00004 #include "object.hpp"
00005 #include <vector>
00006
00011 class Triangle : public Object
00012 {
00013 public:
00022     Triangle(Vector v0, Vector v1, Vector v2, std::shared_ptr<Material> m) : Object(v0, m) {
00023         //Vectors pointing at the vertices
00024         a = v0;
00025         b = v1;
00026         c = v2;
00027
00028         //Vectors on the triangle plane
00029         e1 = b-a;
00030         e2 = c-a;
00031
00032         //Normal vector of the triangle
00033         n = e1.cross(e2).normalized();
00034
00035         //Centroid of the triangle
00036         centroid = Vector(v0.mean(), v1.mean(), v2.mean());
00037     }
00038
00049     void collision(Ray& ray, Hit &rayHit, float& smallestDistance) {
00050         //Compute determinant
00051         Vector p = ray.direction.cross(e2);
00052         float det = e1.dot(p);
00053
00054         //Check if the ray is on the same plane as the triangle
00055         if(det == 0) return;
00056
00057         float invDet = 1 / det;
00058
00059         //Compute beta
00060         Vector s = ray.origin - a;
00061         float beta = s.dot(p)*invDet;
00062
00063         //If beta < 0 or beta > 1 the ray does not intersect the triangle
00064         if(beta < 0 || beta > 1) return;
00065
00066         //Compute gamma
00067         Vector q = s.cross(e1);
00068         float gamma = ray.direction.dot(q)*invDet;
```

```
00069
00070         //If gamma < 0 or beta + gamma > 1 the ray does not intersect the triangle
00071         if(gamma < 0 || beta + gamma > 1) return;
00072
00073         //Compute t
00074         float t = e2.dot(q)*invDet;
00075
00076         //If t < smallestDistance the ray intersects the triangle
00077         if(t < smallestDistance) {
00078             smallestDistance = t;
00079             rayHit.distance = t;
00080             rayHit.material = this->getMaterial();
00081             rayHit.did_hit = true;
00082             rayHit.point = ray.origin + ray.direction*t;
00083             rayHit.normal = this->n;
00084         }
00085
00086         return;
00087     }
00088
00094     void printInfo(std::ostream& out) const {
00095         std::vector<Vector> v = getVertexPos();
00096         out « "Triangle at: (" « v[0].transpose() « "), (" « v[1].transpose()
00097         « "), (" « v[2].transpose() « "), with normal: (" « getNormal().transpose()
00098         « "), with material: " « getMaterial()->getName() « std::endl;
00099     }
00100
00106     std::vector<Vector> getVertexPos() const {
00107         std::vector<Vector> v;
00108         v.push_back(a);
00109         v.push_back(b);
00110         v.push_back(c);
00111         return v;
00112     }
00113
00119     std::vector<Vector> getPlaneVec() const {
00120         std::vector<Vector> v;
00121         v.push_back(e1);
00122         v.push_back(e2);
00123         return v;
00124     }
00125
00131     Vector getNormal() const {
00132         return n;
00133     }
00134
00140     Vector getCentroid() const {
00141         return centroid;
00142     }
00143
00144 private:
00145     Vector a, b, c;
00146     Vector e1, e2;
00147     Vector n;
00148     Vector centroid;
00149 };
00150
00151
```

5.14 trianglemesh.hpp

```
00001 #pragma once
00002
00003 #include "triangle.hpp"
00004 #include "bvh.hpp"
00005 #include "../libs/tiny_obj_loader/tiny_obj_loader.cc"
00006
00007 #include <iostream>
00008 #include <vector>
00009 #include <string>
00010
00015 class TriangleMesh : public Object {
00016 public:
00026     TriangleMesh(std::string obj_filepath, Vector scenePos, std::shared_ptr<Material> m, Vector
      rotation, double scale) : Object(scenePos, m) {
00027         unsigned long pos = obj_filepath.find_last_of("/");
00028         std::string basepath = obj_filepath.substr(0, pos+1);
00029         std::string obj_name = obj_filepath.substr(pos+1, obj_filepath.length());
00030
00031         //Name of the object wihthout .obj
00032         name = obj_name.substr(0, obj_name.length()-4);
00033
00034         tinyobj::attrib_t attributes;
```

```
00035         std::vector<Triangle> triangles;
00036         std::vector<tinyobj::shape_t> objects;
00037         std::vector<tinyobj::material_t> materials;
00038         std::string warnings;
00039         std::string errors;
00040         std::vector<Vector> vertices;
00041 
00042         //Load data from object file
00043         bool r = tinyobj::LoadObj(&attributes, &objects, &materials, &warnings, &errors,
      obj_filepath.c_str(), basepath.c_str());
00044 
00045         //Check for errors in loading the data
00046         if(!errors.empty()) {
00047             std::cout << "Error: " << errors << std::endl;
00048         }
00049         if(!r) {
00050             std::cout << "Failed to load object file." << std::endl;
00051             exit(1);
00052         }
00053 
00054         //Loop over objects (shapes)
00055         for(size_t o = 0; o < objects.size(); o++) {
00056             size_t o_offset = 0;
00057             //Loop over triangles (faces)
00058             for(size_t t = 0; t < objects[o].mesh.num_face_vertices.size(); t++) {
00059                 int fv = objects[o].mesh.num_face_vertices[t];
00060 
00061                 //Looping over vertices in this face
00062                 for(size_t v = 0; v < fv; v++) {
00063                     tinyobj::index_t idx = objects[o].mesh.indices[o_offset + v];
00064                     tinyobj::real_t vx = attributes.vertices[3*idx.vertex_index+0];
00065                     tinyobj::real_t vz = attributes.vertices[3*idx.vertex_index+1];
00066                     tinyobj::real_t vy = attributes.vertices[3*idx.vertex_index+2];
00067 
00068                     //Orientating the object according to the rotation vector
00069                     Eigen::AngleAxisd xRotation(rotation[0], Vector::UnitX());
00070                     Eigen::AngleAxisd yRotation(rotation[1], Vector::UnitY());
00071                     Eigen::AngleAxisd zRotation(rotation[2], Vector::UnitZ());
00072 
00073                     Vector relVertex = Vector(vx, vy, vz);
00074 
00075                     relVertex = xRotation * relVertex;
00076                     relVertex = yRotation * relVertex;
00077                     relVertex = zRotation * relVertex;
00078 
00079                     //Creating one vertex
00080                     Vector vertex = relVertex*scale+scenePos;
00081                     vertices.push_back(vertex);
00082                 }
00083                 o_offset += fv;
00084             }
00085         }
00086 
00087         //Loops over vertices and creates triangles
00088         for(int i = 0; i < vertices.size()/3; ++i) {
00089             triangles.push_back(Triangle(vertices[i*3], vertices[i*3+1], vertices[i*3+2], m));
00090         }
00091 
00092         bvh = BVH(triangles);
00093 
00094         std::cout << "Object file: " << obj_name << ", succesfully opened!" << std::endl;
00095 
00096         triangles.clear();
00097         objects.clear();
00098         materials.clear();
00099     }
00100 
00108     void collision(Ray& ray, Hit& rayHit, float& smallestDistance) {
00109 
00110         bvh.BVHCollision(ray, rayHit, smallestDistance, bvh.getRootNodeIdx());
00111 
00112         return;
00113     }
00114 
00120     void printInfo(std::ostream& out) const {
00121         out << "TriangleMesh object: " << name << ", at :" << this->getPosition().transpose() << ", with
      material: " << (*getMaterial()).getName() << std::endl;
00122     }
00123 
00129     BVH getBVH() const {
00130         return bvh;
00131     }
00132 
00138     std::string getName() const {
00139         return name;
00140     }
00141 
```

```
00142 private:
00143     BVH bvh;
00144     std::string name;
00145 };
```

5.15 renderer.hpp

```
00001 #pragma once
00002
00003 #include "types.hpp"
00004 #include <vector>
00005 #include "randomgenerator.hpp"
00006 #include <iostream>
00007 #include <omp.h>
00008 #include <chrono>
00009 #include <memory>
00010
00015 class Renderer
00016 {
00017 private:
00018
00019     std::shared_ptr<Scene> scene_;
00020     Camera camera_;
00021     RandomGenerator rnd_;
00022
00023     int resolution_x;
00024     int resolution_y;
00025     std::vector<std::vector<Color» result;
00026
00027     float anti_alias_radius = 1;
00028
00029     int max_bounces = 5; // Default value if anyone do not change
00030
00031     float view_width;
00032     float view_height;
00033
00034     Vector topleft_pixel;
00035     Vector pixel_x;
00036     Vector pixel_y;
00037
00038     int progressBarWidth = 100;
00039
00046     Color clamp(Color input) {
00047         float R = input(0) > 1 ? 1 : input(0);
00048         float G = input(1) > 1 ? 1 : input(1);
00049         float B = input(2) > 1 ? 1 : input(2);
00050         return Color(R, G, B);
00051     }
00052
00063     Ray createRay(int x, int y) {
00064
00065         // Depth of field effect randomizes the origin
00066         Vector2 jiggle = rnd_.randomInCircle() * camera_.DoF;
00067         Vector randomShift = jiggle(0) * pixel_x + jiggle(1) * pixel_y;
00068         Point origin = camera_.position + randomShift;
00069
00070         // Anti-aliasing randomizes the target
00071         jiggle = rnd_.randomInCircle() * anti_alias_radius;
00072         randomShift = jiggle(0) * pixel_x + jiggle(1) * pixel_y;
00073         Vector target = topleft_pixel + pixel_y * y + pixel_x * x + randomShift;
00074
00075         Vector direction = (target-origin).normalized();
00076
00077         return Ray{.origin = origin, .direction = direction};
00078     }
00079
00088     Hit rayCollision(Ray& ray) {
00089
00090         float closestHit = INFINITY;
00091         Hit rayHit = { .did_hit = false };
00092
00093         for (auto object : (*scene_).getObjects()) {
00094             object->collision(ray, rayHit, closestHit);
00095         }
00096
00097         return rayHit;
00098     }
00099
00106     Light trace(Ray& ray) {
00107         for (int bounce = 0; bounce < max_bounces; ++bounce)
00108         {
00109             Hit hit = rayCollision(ray);
00110
```

```
00111             if (hit.did_hit && hit.distance > 0.0001) {
00112                 // Update ray according to material properties
00113                 (*hit.material).updateRay(ray, hit);
00114             }
00115             else
00116             {
00117                 ray.light += (*scene_).getEnvironment().getLight(ray).cwiseProduct(ray.color);
00118                 break;
00119             }
00120         }
00121         return ray.light;
00122     }
00123
00130     void progressBar(int sample, int samples) {
00131         std::cout « "[";
00132         int pos = progressBarWidth * (sample + 1)/samples;
00133         for (int i = 0; i < progressBarWidth; ++i) {
00134             if (i < pos) std::cout « "=";
00135             else if (i == pos) std::cout « ">";
00136             else std::cout « " ";
00137         }
00138         std::cout « "] " « std::round((sample + 1) * 100.0 / samples) « " %\r";
00139         std::cout.flush();
00140     }
00141
00142 public:
00143
00151     Renderer(int res_x, int res_y, std::shared_ptr<Scene> sceneToRender) {
00152         resolution_x = res_x;
00153         resolution_y = res_y;
00154         result = std::vector<std::vector<Color»(resolution_x, std::vector<Color> (resolution_y));
00155         scene_ = sceneToRender;
00156         camera_ = (*scene_).getCamera();
00157         view_width = camera_.focus_distance * tan(camera_.fov / 2);
00158         view_height = view_width * (resolution_y - 1) / (resolution_x - 1);
00159         pixel_x = -2 * view_width / (resolution_x - 1) * camera_.left;
00160         pixel_y = - 2 * view_height / (resolution_y - 1) * camera_.up;
00161         topleft_pixel = camera_.position + camera_.focus_distance * camera_.direction + view_width *
      camera_.left + view_height * camera_.up;
00162     }
00163
00169     void setMaxBounces(int bounces) {
00170         max_bounces = bounces;
00171     }
00172
00178     auto parallelRender(int samples) {
00179
00180         auto startTime = std::chrono::high_resolution_clock::now();
00181         std::vector<std::vector<Color» result(resolution_x, std::vector<Color> (resolution_y));
00182
00183         std::cout « "Rendering started..." « std::endl;
00184
00185         for (int sample = 0; sample < samples; ++sample)
00186         {
00187             float weight = 1.0 / (sample + 1);
00188
00189             #pragma omp parallel for num_threads(omp_get_max_threads())
00190             for (int x = 0; x < resolution_x; ++x)
00191             {
00192                 for (int y = 0; y < resolution_y; ++y)
00193                 {
00194                     Ray ray = createRay(x, y);
00195                     Light totalLight = trace(ray);
00196                     result[x][y] = clamp(result[x][y] * (1 - weight) + weight *
      totalLight.cwiseSqrt());
00197                 }
00198             }
00199             progressBar(sample, samples);
00200         }
00201
00202         std::cout « std::endl;
00203         auto endTime = std::chrono::high_resolution_clock::now();
00204         std::chrono::duration<float> duration = endTime - startTime;
00205         std::cout « "Used " « omp_get_max_threads() « " threads.\n" « std::endl;
00206         std::cout « "Rendering completed in " « duration.count() « " seconds.\n" « std::endl;
00207
00208         return result;
00209     }
00210
00216     void setDof(float dof) {
00217         camera_.DoF = dof;
00218     }
00219 };
```

5.16 environment.hpp

```
00001 #pragma once
00002
00003 #include "types.hpp"
00004
00011 class Environment
00012 {
00013 private:
00014     Color skyColor_;
00015     Color horizonColor_;
00016     Color groundColor_;
00017 public:
00018
00023     Environment() : skyColor_(Color(0, 0, 0)), horizonColor_(Color(0, 0, 0)), groundColor_(Color(0, 0,
      0)) {}
00024
00034     void setSky(Color skyColor = Color(0.2, 0.5, 1.0), Color horizonColor = Color(0.7, 0.8, 0.8),
      Color groundColor = Color(0.1, 0.1, 0.1)) {
00035         skyColor_ = skyColor;
00036         horizonColor_ = horizonColor;
00037         groundColor_ = groundColor;
00038     }
00039
00048     Light getLight(Ray& ray) {
00049
00050         if (ray.direction(2) >= 0)
00051         {
00052             return horizonColor_ + (skyColor_ - horizonColor_) * pow(abs(ray.direction(2)), 0.8);
00053         }
00054         else
00055         {
00056             return horizonColor_ + (groundColor_ - horizonColor_) * pow(abs(ray.direction(2)), 0.1);
00057         }
00058     }
00059
00060     // Getters
00061     Color getHorizonColor() { return horizonColor_; }
00062     Color getGroundColor() { return groundColor_; }
00063     Color getSkyColor() { return skyColor_; }
00064 };
```

5.17 material.hpp

```
00001 #ifndef MATERIAL_CLASS
00002 #define MATERIAL_CLASS
00003
00004 #include "types.hpp"
00005 #include "randomgenerator.hpp"
00006
00007 #include <vector>
00008
00012 class Material {
00013     private:
00014         Color color_;
00015         std::string name_;
00016
00017     public:
00018         RandomGenerator rnd_;
00019
00025         Color getColor() { return color_; }
00026
00032         std::string getName() { return name_; }
00033
00040         Material(Color color, std::string name) : color_(color), name_(name) {}
00041
00048         Vector reflectionDir(Ray& ray, Hit& hit) {
00049             return ray.direction - 2 * ray.direction.dot(hit.normal) * hit.normal;
00050         }
00051
00061         Vector diffuseDir(Hit& hit) {
00062             return (rnd_.randomDirection() + hit.normal).normalized();
00063         }
00064
00072         void updateColor(Ray& ray) {
00073             ray.color = ray.color.cwiseProduct(getColor());
00074             return;
00075         }
00076
00082         virtual void updateRay(Ray& ray, Hit& hit) = 0;
00083 };
00084
00089 class Diffuse : public Material {
```

```
00090     private:
00091
00092         bool emitting_;
00093         Color emission_color_;
00094         float emission_strength_;
00095
00104         void diffuseEmission(Ray& ray) {
00105             if (!isEmitting()) { return; } // If not emitting, do nothing
00106             else {
00107                 ray.light += (getEmStrength() * getEmColor()).cwiseProduct(ray.color);
00108                 return;
00109             }
00110         }
00111
00112     public:
00113
00122         Diffuse(Color color, std::string name, float emission_strength, Color emission_color) :
00123         Material(color, name), emission_strength_(emission_strength), emission_color_(emission_color)
      {
00124             // Boolean value, to determine if material is emitting
00125             (emission_strength > 0) ? (emitting_ = true) : (emitting_ = false);
00126         }
00127
00133         Color getEmColor() { return emission_color_; }
00134
00140         float getEmStrength() { return emission_strength_; }
00141
00147         bool isEmitting() { return emitting_; }
00148
00154         void updateRay(Ray& ray, Hit& hit) {
00155             ray.origin = hit.point;
00156             Vector diffused_dir = diffuseDir(hit);
00157             ray.direction = diffused_dir;
00158             updateColor(ray);
00159             diffuseEmission(ray);
00160             return;
00161         }
00162 };
00163
00168 class Reflective : public Material {
00169     private:
00170
00171         float specularity_;
00172
00173     public:
00174
00182         Reflective(Color color, std::string name, float specularity) :
00183             Material(color, name), specularity_(specularity) {}
00184
00190         float getSpecularity() { return specularity_; }
00191
00197         void updateRay(Ray& ray, Hit& hit) {
00198             ray.origin = hit.point;
00199             updateColor(ray);
00200             Vector reflectedRay = reflectionDir(ray, hit);
00201             Vector diffusedRay = diffuseDir(hit);
00202             // Weight direction of the reflection based on specularity
00203             ray.direction = diffusedRay + getSpecularity() * (reflectedRay - diffusedRay);
00204             return;
00205         }
00206 };
00207
00213 class ClearCoat : public Reflective {
00214     private:
00215
00216         float clearcoat_;
00217         Color clearcoat_color_;
00218
00224         bool clearCoatBounce() { return (clearcoat_ >= rnd_.randomZeroToOne()); }
00225
00233         void updateColor(Ray& ray, bool bounce) {
00234             ray.color = ray.color.cwiseProduct(bounce ? getClearCoatColor() : getColor());
00235             return;
00236         }
00237
00238
00239     public:
00240
00246         Color getClearCoatColor() { return clearcoat_color_; }
00247
00253         float getClearCoat() { return clearcoat_; }
00254
00264         ClearCoat(Color color, std::string name, float specularity, float clearcoat, Color
      clearcoat_color) :
00265             Reflective(color, name, specularity), clearcoat_(clearcoat),
      clearcoat_color_(clearcoat_color) {}
00266
```

```
00272         void updateRay(Ray& ray, Hit& hit) {
00273             bool bounce = clearCoatBounce();
00274             ray.origin = hit.point;
00275             updateColor(ray, bounce);
00276             Vector diffused_dir = diffuseDir(hit);
00277             if (bounce) {
00278                 ray.direction = diffused_dir + getSpecularity() * (reflectionDir(ray, hit) -
      diffused_dir);
00279             }
00280             else {
00281                 ray.direction = diffused_dir;
00282             }
00283             return;
00284         }
00285 };
00286
00291 class Refractive : public Material {
00292     private:
00293         float refraction_ratio_; // From outside to inside the material
00294
00295         // Theta is angle between incoming ray and surface normal (vectors has to be normalized)
00296         float cosTheta(Vector a, Vector b) {
00297             return a.dot(b);
00298         }
00299
00308         Vector refractionPerpendicular(Vector in, Vector normal, float ref_ratio) {
00309             float cos_theta = cosTheta(-in, normal);
00310             return ref_ratio * (in + cos_theta * normal);
00311         }
00312
00321         Vector refractionDir(Ray& ray, Hit& hit, float ref_ratio) {
00322             Vector perpendicular = refractionPerpendicular(ray.direction, hit.normal, ref_ratio);
00323             Vector parallel = -sqrt(1 - perpendicular.dot(perpendicular)) * hit.normal;
00324             return perpendicular + parallel;
00325         }
00326
00336         float reflectance(float cos_theta, float ref_ratio) {
00337             float r0 = (1 - ref_ratio) / (1 + ref_ratio);
00338             r0 = r0 * r0;
00339             return r0 + (1 - r0) * pow((1 - cos_theta), 5);
00340         }
00341
00342     public:
00343
00351         Refractive(Color color, std::string name, float refraction_ratio) :
00352             Material(color, name), refraction_ratio_(refraction_ratio) {}
00353
00360         void updateRay(Ray& ray, Hit& hit) {
00361             ray.origin = hit.point;
00362             updateColor(ray);
00363
00364             // Real refraction ratio depends on the direction
00365             float ref_ratio = ray.inside_material ? 1 / refraction_ratio_ : refraction_ratio_;
00366
00367             // Real glass reflects depending on the intersection angle and refraction ratio
00368             float cos_theta = cosTheta(-ray.direction, hit.normal);
00369             float sin_theta = sqrt(1 - cos_theta*cos_theta);
00370             bool must_reflect = ref_ratio * sin_theta > 1;
00371             float reflectanceProb = reflectance(cos_theta, ref_ratio);
00372
00373             if (must_reflect || reflectanceProb > rnd_.randomZeroToOne()) {
00374                 // Reflects
00375                 Vector reflectedDir = reflectionDir(ray, hit);
00376                 ray.direction = reflectedDir;
00377             } else {
00378                 // Refracts
00379                 Vector refractedDir = refractionDir(ray, hit, ref_ratio);
00380                 ray.inside_material = !(ray.inside_material);
00381                 ray.direction = refractedDir;
00382             }
00383             return;
00384         }
00385 };
00386
00387 #endif
```

5.18 scene.hpp

```
00001 #pragma once
00002
00003 #include <memory>
00004 #include <list>
00005 #include <iostream>
```

```
00006 #include "ball.hpp"
00007 #include "types.hpp"
00008 #include "environment.hpp"
00009
00014 class Scene
00015 {
00016 private:
00017     std::list<std::shared_ptr<Object» objects_;
00018     Camera camera_;
00019     Environment environment_;
00020
00021 public:
00026     Scene() = default;
00027
00034     Scene(Camera camera, std::list<std::shared_ptr<Object» objects) : camera_(camera),
      objects_(objects) {}
00035
00036
00041     ~Scene() {}
00042
00043     // Default copying for now - list is copied (are the objects copied as well?)
00044     Scene& operator=(const Scene& that) = default;
00045     Scene(const Scene& that) = default;
00046
00047     Camera getCamera() const { return camera_; }
00048
00049     Environment& getEnvironment() { return environment_; }
00050
00051     std::list<std::shared_ptr<Object» getObjects() const { return objects_; }
00052
00053     void setFov(float fov) {
00054         camera_.fov = fov;
00055     }
00056
00057     void setFocusDist(float dof) {
00058         camera_.focus_distance = dof;
00059     }
00060
00068     friend std::ostream &operator«(std::ostream& out, const Scene& scene) {
00069         out « "\n======================================== SCENE INFORMATION
      ========================================" « std::endl;
00070
00071         for (auto object : scene.objects_)
00072         {
00073             out « (*object);
00074         }
00075
00076         out « std::endl;
00077
00078         out « "Camera at: (" « scene.camera_.position.transpose() « ") looking at point: ("
00079             « scene.camera_.lookingAt.transpose() « ") with FOV: " « scene.camera_.fov / M_PI * 180
00080             « " degrees, DOF: " « scene.camera_.DoF « " and focus distance; " «
      scene.camera_.focus_distance « "." « std::endl;
00081
00082         out «
      "================================================================================================\n"
      « std::endl;
00083
00084         return out;
00085     }
00086 };
```

5.19 randomgenerator.hpp

```
00001 #pragma once
00002
00003 #include <random>
00004 #include "types.hpp"
00005
00010 class RandomGenerator
00011 {
00012 private:
00013
00014     std::random_device randomDevice; /* Random numbers provided by the OS */
00015     std::mt19937 randomInt; /* Uniformly distributed random integers */
00016     std::uniform_real_distribution<float> randZeroToOne; /* Uniformly distributed reals between 0 and
      1 */
00017     std::normal_distribution<float> normal;  /* Random numbers satisfying the standard normal
      distribution */
00018
00019 public:
00020     RandomGenerator() : randomInt(randomDevice()), randZeroToOne(0, 1), normal(0, 1) {}
00021     ~RandomGenerator() = default;
```

```
00022
00028     Vector randomDirection() {
00029         return Vector(normal(randomInt), normal(randomInt), normal(randomInt)).normalized();
00030     }
00031
00037     float randomZeroToOne() { return randZeroToOne(randomInt); }
00038
00044     Vector2 randomInCircle() {
00045         float angle = randZeroToOne(randomInt) * 2 * M_PI;
00046         float distance = randZeroToOne(randomInt);
00047
00048         return Vector2(cos(angle), sin(angle)) * sqrt(distance);
00049     }
00050 };
```

5.20 types.hpp

```
00001 #pragma once
00002
00003 #include <Eigen/Dense>
00004 #include <string>
00005 #include <memory>
00006
00007 typedef Eigen::Vector3d Vector;
00008 typedef Eigen::Vector3d Point;
00009 typedef Eigen::Vector3d Color;
00010 typedef Eigen::Vector3d Light;
00011 typedef Eigen::Vector2d Vector2;
00012 typedef Eigen::Matrix<double, 3, 3> Matrix;
00013
00014 // Forward declaration for Material class, such that the Hit struct knows the existence
00015 class Material;
00016
00021 struct Camera
00022 {
00023     Point position;
00024     Vector lookingAt;
00025     Vector direction;
00026     Vector up;
00027     Vector left;
00028     float fov;
00029     float focus_distance;
00030     float DoF;
00031 };
00032
00037 struct Ray
00038 {
00039     Point origin;
00040     Vector direction;
00041     bool inside_material = false;
00042     Color color = Color(1.0, 1.0, 1.0);
00043     Light light = Color(0.0, 0.0, 0.0);
00044 };
00045
00050 struct Hit
00051 {
00052     bool did_hit = false;
00053     std::shared_ptr<Material> material; // Has to be pointer, since compiler do not yet know anything
      about Material class
00054     Vector normal;
00055     Point point;
00056     float distance;
00057 };
```

Index