# Software structure

Generated by Doxygen 1.9.8

# Chapter 1

# Source content

This folder should contain only hpp/cpp files of your implementation. You can also place hpp files in a separate directory `include`.

You can create a summary of files here. It might be useful to describe file relations, and brief summary of their content.

# Chapter 2

# Hierarchical Index

## 2.1 Class Hierarchy

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

# Chapter 3

# Class Index

## 3.1 Class List

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

# Chapter 4

# File Index

## 4.1 File List

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

# Chapter 5

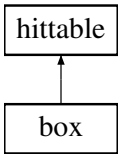
# Class Documentation

## 5.1 box Class Reference

Represents a axis-aligned box in 3D space.

`#include <box.hpp>`

Inheritance diagram for box:




### Public Member Functions

- box (const point3 &p0, const point3 &p1, shared_ptr< material > ptr)

  *Constructs a box with specified parameters.*
- virtual bool hit (const ray &r, interval ray_t, hit_record &rec) const override

  *Determines if a ray hits the box.*

### Public Attributes

- point3 **box_min**
- point3 **box_max**
- hittable_list **sides**

### 5.1.1 Detailed Description

Represents a axis-aligned box in 3D space.

### 5.1.2 Constructor & Destructor Documentation

#### 5.1.2.1 box()

```
box::box (
            const point3 & p0,
            const point3 & p1,
            shared_ptr< material > ptr )
```

Constructs a box with specified parameters.

**Parameters**

| | |
|---|---|
| *p0* | Minimum point of the box. |
| *p1* | Maximum point of the box. |
| *ptr* | Material of the box. |

### 5.1.3 Member Function Documentation

#### 5.1.3.1 hit()

```
bool box::hit (
            const ray & r,
            interval ray_t,
            hit_record & rec ) const  [override], [virtual]
```

Determines if a ray hits the box.

Given a ray, this function checks if it intersects with the box within the specified ray parameter interval.

**Parameters**

| | |
|---|---|
| *r* | The ray. |
| *ray_t* | The valid interval for the ray parameter. |
| *rec* | The hit record to be filled if the ray hits the box. |

**Returns**

True if the ray hits the box, false otherwise.

Implements hittable.

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

- src/box.hpp
- src/box.cpp

## 5.2 camera Class Reference

Represents a movable camera in the world.

```
#include <camera.hpp>
```

**Public Member Functions**

- camera (point3 lookfrom, point3 lookat, vec3 vup, double vfov, double aspect_ratio, double aperture, double focus_dist, double _time0, double _time1)

  *Constructs a camera with the specified parameters.*
- ray get_ray (double s, double t) const

  *Generates a ray from the camera for a given screen position.*
- point3 getOrigin () const

  *Gets the origin of the camera.*
- double getLensRadius () const

  *Gets the radius of the lens.*

## 5.2.1 Detailed Description

Represents a movable camera in the world.

## 5.2.2 Constructor & Destructor Documentation

### 5.2.2.1 camera()

```
camera::camera (
            point3 lookfrom,
            point3 lookat,
            vec3 vup,
            double vfov,
            double aspect_ratio,
            double aperture,
            double focus_dist,
            double _time0 = 0,
            double _time1 = 0 )
```

Constructs a camera with the specified parameters.

**Parameters**

| | |
|---|---|
| *lookfrom* | The camera's position. |
| *lookat* | The point the camera is looking at. |
| *vup* | The "up" vector indicating the camera's orientation. |
| *vfov* | The vertical field-of-view in degrees. |
| *aspect_ratio* | The aspect ratio of the viewport. |
| *_time0* | Time when the shutter opens for blur |
| *_time1* | Time when the shutter closes for blur |

## 5.2.3 Member Function Documentation

### 5.2.3.1 get_ray()

```
ray camera::get_ray (
```

```
            double s,
            double t ) const
```

Generates a ray from the camera for a given screen position.

**Parameters**

| | |
|---|---|
| *s* | The horizontal coordinate on the screen (0 to 1). |
| *t* | The vertical coordinate on the screen (0 to 1). |

**Returns**

The generated ray.

#### 5.2.3.2 getLensRadius()

```
double camera::getLensRadius ( ) const  [inline]
```

Gets the radius of the lens.

**Returns**

Lens radius

#### 5.2.3.3 getOrigin()

```
point3 camera::getOrigin ( ) const  [inline]
```

Gets the origin of the camera.

**Returns**

The camera's origin point.

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

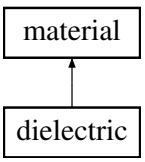
- src/camera.hpp
- src/camera.cpp

## 5.3 dielectric Class Reference

Class representing dielectric material.

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

Inheritance diagram for dielectric:



### Public Member Functions

- dielectric (double index_of_reftraction)

  *Constructor for dielectric material.*
- virtual bool scatter (const ray &r_in, const hit_record &rec, color &attenuation, ray &scattered) const override

  *Calculates the correct scatter of the vector for dielectric material. This should look like glass.*

## Public Member Functions inherited from material

- virtual color emitted (double u, double v, const point3 &p) const

### 5.3.1 Detailed Description

Class representing dielectric material.

### 5.3.2 Constructor & Destructor Documentation

#### 5.3.2.1 dielectric()

```
dielectric::dielectric (
            double index_of_reftraction )
```

Constructor for dielectric material.

**Parameters**

| | |
|---|---|
| *index_of_refraction* | Index of refraction. |

### 5.3.3 Member Function Documentation

#### 5.3.3.1 scatter()

```
bool dielectric::scatter (
            const ray & r_in,
            const hit_record & rec,
            color & attenuation,
            ray & scattered ) const  [override], [virtual]
```

Calculates the correct scatter of the vector for dielectric material. This should look like glass.

**Parameters**

| | |
|---|---|
| *r_in* | The incoming vector |
| *rec* | hit record of hit point. |
| *attenuation* | The color change of the ray after hitting. |
| *scattered* | The scattered ray. |

**Returns**

true if hit.

false otherwise.

Implements material.

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

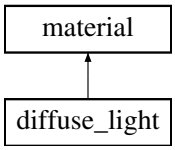
- src/material.hpp
- src/material.cpp

## 5.4 diffuse_light Class Reference

Represents a light source material.

`#include <material.hpp>`

Inheritance diagram for diffuse_light:




**Public Member Functions**

- diffuse_light (color c)

  *Contructor with a color.*
- virtual bool scatter (const ray &r_in, const hit_record &rec, color &attenuation, ray &scattered) const override
- virtual color emitted (double u, double v, const point3 &p) const override

  *Emits light.*

### 5.4.1 Detailed Description

Represents a light source material.

### 5.4.2 Constructor & Destructor Documentation

#### 5.4.2.1 diffuse_light()

```
diffuse_light::diffuse_light (
            color c )
```

Contructor with a color.

**Parameters**

| | |
|---|---|
| *a* | Given color for the light. |

### 5.4.3 Member Function Documentation

#### 5.4.3.1 emitted()

```
color diffuse_light::emitted (
            double u,
            double v,
            const point3 & p ) const  [override], [virtual]
```

Emits light.

**Parameters**

| | |
|---|---|
| *u* | Coordinate of object surface. |
| *v* | Coordinate of object surface. |
| *p* | Coordinate of the 3D world. |

**Returns**

Coordinates for the light ray.

Reimplemented from material.

#### 5.4.3.2 scatter()

```
bool diffuse_light::scatter (
            const ray & r_in,
            const hit_record & rec,
            color & attenuation,
            ray & scattered ) const  [override], [virtual]
```

Implements material.

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

- src/material.hpp
- src/material.cpp

## 5.5 hit_record Class Reference

Stores values related to a hit.

```
#include <hittable.hpp>
```

#### Public Member Functions

- void set_face_normal (const ray &r, const vec3 &outward_normal)

  *Sets the nomal for the surface of the object. And also checks if the ray is coming from the outside of object and changes front_face accordingly.*

#### Public Attributes

- point3 p
- vec3 normal
- shared_ptr< material > mat_ptr
- double t
- double **u**
- double v
- bool front_face

### 5.5.1 Detailed Description

Stores values related to a hit.

### 5.5.2 Member Function Documentation

#### 5.5.2.1 set_face_normal()

```
void hit_record::set_face_normal (
            const ray & r,
            const vec3 & outward_normal )
```

Sets the nomal for the surface of the object. And also checks if the ray is coming from the outside of object and changes front_face accordingly.

**Parameters**

| | |
|---|---|
| *r* | The coming ray. |
| *outward_normal* | The outward normal of the surface. |

### 5.5.3 Member Data Documentation

#### 5.5.3.1 front_face

```
bool hit_record::front_face
```

Holds information about whether the ray is coming from outside (true).

#### 5.5.3.2 mat_ptr

```
shared_ptr<material> hit_record::mat_ptr
```

Pointer to the object material.

#### 5.5.3.3 normal

```
vec3 hit_record::normal
```

The normal of object surface at the hit point.

#### 5.5.3.4 p

```
point3 hit_record::p
```

Position of the hit point.

#### 5.5.3.5 t

```
double hit_record::t
```

The size of t or the length when hit.

#### 5.5.3.6 v

```
double hit_record::v
```

Surface coordinates of the object.

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

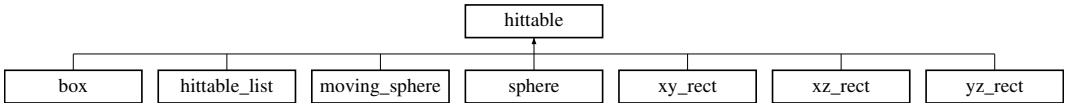
- src/hittable.hpp
- src/hittable.cpp

## 5.6 hittable Class Reference

a base class for different kind of objects.

```
#include <hittable.hpp>
```

Inheritance diagram for hittable:




### Public Member Functions

- virtual bool hit (const ray &r, interval ray_t, hit_record &rec) const =0

  *Chekcs if ray hits. And updates hit record.*

### 5.6.1 Detailed Description

a base class for different kind of objects.

### 5.6.2 Member Function Documentation

#### 5.6.2.1 hit()

```
virtual bool hittable::hit (
            const ray & r,
            interval ray_t,
            hit_record & rec ) const  [pure virtual]
```

Chekcs if ray hits. And updates hit record.

**Parameters**

| | |
|---|---|
| *r* | The coming ray. |
| *ray_t* | The interval to be checked for intersection. |
| *rec* | The hit_record of the possible hit |

Implemented in box, hittable_list, moving_sphere, xy_rect, xz_rect, yz_rect, and sphere.

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

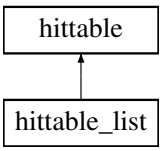
- src/hittable.hpp

## 5.7 hittable_list Class Reference

Represents a list of objects the ray can hit.

`#include <hittable_list.hpp>`

Inheritance diagram for hittable_list:



### Public Member Functions

- **hittable_list** ()

  *Default constructor.*
- hittable_list (shared_ptr< hittable > object)

  *Constructor with an initial hittable object.*
- void **clear** ()

  *Clears the list of objects.*
- void add (shared_ptr< hittable > object)

  *Adds an object to the end of the list.*
- bool hit (const ray &r, interval ray_t, hit_record &rec) const override

  *Checks if a ray hits an object in the list given the parameters. And updates the record of that object.*

### Public Attributes

- std::vector< shared_ptr< hittable > > objects

## 5.7.1 Detailed Description

Represents a list of objects the ray can hit.

## 5.7.2 Constructor & Destructor Documentation

### 5.7.2.1 hittable_list()

```
hittable_list::hittable_list (
            shared_ptr< hittable > object )
```

Constructor with an initial hittable object.

**Parameters**

| | |
|---|---|
| *object* | The object to be added to the list. |

## 5.7.3 Member Function Documentation

### 5.7.3.1 add()

```
void hittable_list::add (
            shared_ptr< hittable > object )
```

Adds an object to the end of the list.

**Parameters**

| | |
|---|---|
| *object* | The object to be added to the list. |

#### 5.7.3.2 hit()

```
bool hittable_list::hit (
            const ray & r,
            interval ray_t,
            hit_record & rec ) const  [override], [virtual]
```

Checks if a ray hits an object in the list given the parameters. And updates the record of that object.

**Parameters**

| | |
|---|---|
| *r* | The ray. |
| *ray_t* | Interval or length of ray to be checked for intersection. |
| *rec* | The hit record to be filled if the ray hits an object. |

**Returns**

True if the ray hits any object in the list, false otherwise.

Implements hittable.

### 5.7.4 Member Data Documentation

#### 5.7.4.1 objects

```
std::vector<shared_ptr<hittable> > hittable_list::objects
```

Vector containing shared pointers to hittable objects.

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

- src/hittable_list.hpp
- src/hittable_list.cpp

## 5.8 interval Class Reference

Represents an interval between two points.

```
#include <interval.hpp>
```

**Public Member Functions**

- **interval** ()
  *Default constructor with empty interval.*
- interval (double _min, double _max)
  *Constructor which sets min and max.*
- bool contains (double x) const
  *Checks if the interval contains certain value within range.*
- bool surrounds (double x) const
  *Chekcs if x is between the pramaters. Not including min and max.*
- double clamp (double x) const
  *Checks if x is outside parameters and returns closest min or max to it.*

#### Public Attributes

- double **min**
- double max

#### Static Public Attributes

- static const interval **empty**
- static const interval **universe**

### 5.8.1 Detailed Description

Represents an interval between two points.

### 5.8.2 Constructor & Destructor Documentation

#### 5.8.2.1 interval()

```
interval::interval (
            double _min,
            double _max )
```

Constructor which sets min and max.

**Parameters**

| | |
|---|---|
| *_min* | Min value |
| *_max* | Max value |

### 5.8.3 Member Function Documentation

#### 5.8.3.1 clamp()

```
double interval::clamp (
            double x ) const
```

Checks if x is outside parameters and returns closest min or max to it.

**Parameters**

| | |
|---|---|
| *x* | |

**Returns**

double

#### 5.8.3.2 contains()

```
bool interval::contains (
            double x ) const
```

Checks if the interval contains certain value within range.

**Parameters**

| | |
|---|---|
| *x* | The value to be checked for. |

**Returns**

true if contains within range

false otherwise

#### 5.8.3.3 surrounds()

```
bool interval::surrounds (
            double x ) const
```

Chekcs if x is between the pramaters. Not including min and max.

**Parameters**

| | |
|---|---|
| *x* | |

**Returns**

true if contains

false otherwise

### 5.8.4 Member Data Documentation

#### 5.8.4.1 max

```
double interval::max
```

The parameters for min and max.

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

- src/interval.hpp
- src/interval.cpp

## 5.9 Is Class Reference

Represents a rectangle parallel to the xy-plane.

### 5.9.1 Detailed Description

Represents a rectangle parallel to the xy-plane.

Represents a rectangle parallel to the yz-plane.

Represents a rectangle parallel to the xz-plane.

aligned rectangle.

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

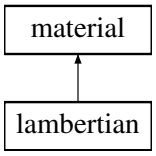
- src/rect.hpp

## 5.10 lambertian Class Reference

Class representing Lambertian diffusion.

`#include <material.hpp>`

Inheritance diagram for lambertian:




#### Public Member Functions

- lambertian (const color &a)
  *Constructor for Lambertian material.*
- virtual bool scatter (const ray &r_in, const hit_record &rec, color &attenuation, ray &scattered) const override
  *Calculates the correct scatter of the vector for lambertian material.*

### Public Member Functions inherited from material

- virtual color emitted (double u, double v, const point3 &p) const

### 5.10.1 Detailed Description

Class representing Lambertian diffusion.

### 5.10.2 Constructor & Destructor Documentation

#### 5.10.2.1 lambertian()

```
lambertian::lambertian (
            const color & a )
```

Constructor for Lambertian material.

**Parameters**

| | |
|---|---|
| *a* | Albedo color. |

### 5.10.3 Member Function Documentation

#### 5.10.3.1 scatter()

```
bool lambertian::scatter (
            const ray & r_in,
            const hit_record & rec,
            color & attenuation,
            ray & scattered ) const  [override], [virtual]
```

Calculates the correct scatter of the vector for lambertian material.

**Parameters**

| | |
|---|---|
| *r_in* | The incoming vector |
| *rec* | hit record of hit point. |
| *attenuation* | The color change of the ray after hitting. |
| *scattered* | The scattered ray. |

**Returns**

true if hit.

false otherwise.

Implements material.

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

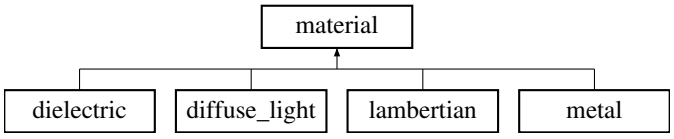
- src/material.hpp
- src/material.cpp

## 5.11 material Class Reference

virtual class for all the materials.

`#include <material.hpp>`

Inheritance diagram for material:



**Public Member Functions**

- virtual color emitted (double u, double v, const point3 &p) const
- virtual bool scatter (const ray &r_in, const hit_record &rec, color &attenuation, ray &scattered) const =0

### 5.11.1 Detailed Description

virtual class for all the materials.

### 5.11.2 Member Function Documentation

#### 5.11.2.1 emitted()

```
virtual color material::emitted (
            double u,
            double v,
            const point3 & p ) const  [inline], [virtual]
```

Reimplemented in diffuse_light.

#### 5.11.2.2 scatter()

```
virtual bool material::scatter (
            const ray & r_in,
            const hit_record & rec,
            color & attenuation,
            ray & scattered ) const  [pure virtual]
```

Implemented in lambertian, metal, and dielectric.

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

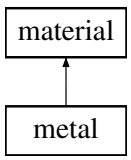
- src/material.hpp

## 5.12 metal Class Reference

Class representing metal material.

`#include <material.hpp>`

Inheritance diagram for metal:



**Public Member Functions**

- metal (const color &a, double f)

  *Constructor for metal material.*
- virtual bool scatter (const ray &r_in, const hit_record &rec, color &attenuation, ray &scattered) const override

  *Calculates the correct scatter of the vector for metal material. Mertal should acct like a mirror.*

## Public Member Functions inherited from material

- virtual color emitted (double u, double v, const point3 &p) const

### 5.12.1 Detailed Description

Class representing metal material.

### 5.12.2 Constructor & Destructor Documentation

#### 5.12.2.1 metal()

```
metal::metal (
            const color & a,
            double f )
```

Constructor for metal material.

**Parameters**

| | |
|---|---|
| *a* | Albedo color. |
| *f* | Fuzziness factor. |

### 5.12.3 Member Function Documentation

#### 5.12.3.1 scatter()

```
bool metal::scatter (
            const ray & r_in,
            const hit_record & rec,
            color & attenuation,
            ray & scattered ) const  [override], [virtual]
```

Calculates the correct scatter of the vector for metal material. Mertal should acct like a mirror.

**Parameters**

| | |
|---|---|
| *r_in* | The incoming vector |
| *rec* | hit record of hit point. |
| *attenuation* | The color change of the ray after hitting. |
| *scattered* | The scattered ray. |

**Returns**

true if hit.

false otherwise.

Implements material.

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

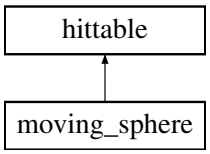
- src/material.hpp
- src/material.cpp

## 5.13 moving_sphere Class Reference

Represents moving sphere.

`#include <moving_sphere.hpp>`

Inheritance diagram for moving_sphere:




### Public Member Functions

- moving_sphere (point3 cen0, point3 cen1, double _time0, double _time1, double r, shared_ptr< material > m)

  *Constructor for the moving_sphere. The moving happens from cen0 to cen1 linearily. _time0 and _time1 are used to calculate the ray intersection point.*
- virtual bool hit (const ray &r, interval ray_t, hit_record &rec) const override

  *Determines if a ray hits the sphere.*
- point3 **center** (double time) const

  *return the center the object. During certain time point.*

### 5.13.1 Detailed Description

Represents moving sphere.

### 5.13.2 Constructor & Destructor Documentation

#### 5.13.2.1 moving_sphere()

```
moving_sphere::moving_sphere (
            point3 cen0,
            point3 cen1,
            double _time0,
            double _time1,
            double r,
            shared_ptr< material > m )  [inline]
```

Constructor for the moving_sphere. The moving happens from cen0 to cen1 linearily. _time0 and _time1 are used to calculate the ray intersection point.

**Parameters**

| | |
|---|---|
| *cen0* | start point of the movement. |
| *cen1* | end point of movement. |
| *_time0* | start time of movement. |
| *_time1* | end point of movement. |
| *r* | radius of the sphere |
| *m* | material of the sphere |

### 5.13.3 Member Function Documentation

#### 5.13.3.1 hit()

```
bool moving_sphere::hit (
            const ray & r,
            interval ray_t,
            hit_record & rec ) const  [override], [virtual]
```

Determines if a ray hits the sphere.

**Parameters**

| | |
|---|---|
| *r* | The ray. |
| *ray_t* | The valid interval for the ray parameter. |
| *rec* | The hit record to be filled if the ray hits the object. |

**Returns**

True if the ray hits the box, false otherwise.

Implements hittable.

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

- src/moving_sphere.hpp
- src/moving_sphere.cpp

## 5.14 PixelInfo Struct Reference

Represents one pixel in the picture.

```
#include <render.hpp>
```

**Public Attributes**

- int **x**
- int **y**
- color **pixel_color**

### 5.14.1 Detailed Description

Represents one pixel in the picture.

Holds information about the pixel, including its position (x, y) and color.

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

- src/render.hpp

## 5.15 ray Class Reference

A class representing a ray with an origin point and a direction vector. Heavily inspired by Peter Shirley https://raytracing.github.io/v3/books/RayTracingInOneWeekend.html.

`#include <ray.hpp>`

#### Public Member Functions

- **ray** ()
  *Default constructor for a ray.*
- ray (const point3 &origin, const vec3 &direction, double time=0.0)
  *Constructor to create a ray with an origin and direction.*
- point3 origin () const
  *Get the origin point of the ray.*
- vec3 direction () const
  *Get the direction vector of the ray.*
- double time () const
  *Get the time of ray appearing.*
- point3 at (double t) const
  *Compute a point on the ray at a given parameter value.*

#### Public Attributes

- point3 **orig**
- vec3 **dir**
- double **tm**

### 5.15.1 Detailed Description

A class representing a ray with an origin point and a direction vector. Heavily inspired by Peter Shirley https://raytracing.github.io/v3/books/RayTracingInOneWeekend.html.

**Version**

0.1

**Date**

2023-11-06

**Copyright**

## 5.15.2 Constructor & Destructor Documentation

### 5.15.2.1 ray()

```
ray::ray (
            const point3 & origin,
            const vec3 & direction,
            double time = 0.0 )  [inline]
```

Constructor to create a ray with an origin and direction.

**Parameters**

| | |
|---|---|
| *origin* | The origin point of the ray. |
| *direction* | The direction vector of the ray. |
| *time* | The time, when ray is existing |

## 5.15.3 Member Function Documentation

### 5.15.3.1 at()

```
point3 ray::at (
            double t ) const
```

Compute a point on the ray at a given parameter value.

**Parameters**

| | |
|---|---|
| *t* | The parameter value that determines the point's position along the ray. |

**Returns**

The point on the ray at the specified parameter value.

### 5.15.3.2 direction()

```
vec3 ray::direction ( ) const  [inline]
```

Get the direction vector of the ray.

**Returns**

The direction vector as a vec3.

#### 5.15.3.3 origin()

```
point3 ray::origin ( ) const  [inline]
```

Get the origin point of the ray.

**Returns**

The origin point as a point3.

#### 5.15.3.4 time()

```
double ray::time ( ) const  [inline]
```

Get the time of ray appearing.

**Returns**

The time of ray appearing.

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

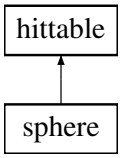
- src/ray.hpp
- src/ray.cpp

## 5.16 sphere Class Reference

Represents a sphere.

`#include <sphere.hpp>`

Inheritance diagram for sphere:




### Public Member Functions

- sphere (point3 _center, double _radius, shared_ptr< material > m)

  *Constructor for the sphere class.*
- bool hit (const ray &r, interval ray_t, hit_record &rec) const override

  *Determines if a ray hits the sphere.*

### 5.16.1 Detailed Description

Represents a sphere.

This class inherits from the hittable class and defines a sphere with a center, radius, and material.

### 5.16.2 Constructor & Destructor Documentation

#### 5.16.2.1 sphere()

```
sphere::sphere (
            point3 _center,
            double _radius,
            shared_ptr< material > m )
```

Constructor for the sphere class.

**Parameters**

| | |
|---|---|
| *_center* | The center of the sphere. |
| *_radius* | The radius of the sphere. |
| *m* | A shared pointer to the material of the sphere. |

### 5.16.3 Member Function Documentation

#### 5.16.3.1 hit()

```
bool sphere::hit (
            const ray & r,
            interval ray_t,
            hit_record & rec ) const  [override], [virtual]
```

Determines if a ray hits the sphere.

**Parameters**

| | |
|---|---|
| *r* | The ray. |
| *ray_t* | The valid interval for the ray parameter. |
| *rec* | The hit record to be filled if the ray hits the object. |

**Returns**

True if the ray hits the object, false otherwise.

Implements hittable.

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

- src/sphere.hpp
- src/spehre.cpp

## 5.17 vec3 Class Reference

A class representing a 3D vector with x, y, and z components.

`#include <vec3.hpp>`

### Public Member Functions

- **vec3** ()

  *Default constructor for a vec3, initializes all components to zero.*
- vec3 (double e0, double e1, double e2)

  *Constructor to create a vec3 with specified components.*
- double x () const

  *Get the x-component of the vec3.*
- double y () const

  *Get the y-component of the vec3.*
- double z () const

  *Get the z-component of the vec3.*
- vec3 operator- () const

  *Unary negation operator, returns the negation of the vector.*
- double operator[ ] (int i) const

  *Array subscript operator to access vector components.*
- double & operator[ ] (int i)

  *Array subscript operator to access vector components (non-const version).*
- vec3 & operator+= (const vec3 &v)

  *Compound addition assignment operator for vector addition.*
- vec3 & operator∗= (const double t)

  *Compound multiplication assignment operator for scalar multiplication.*
- vec3 & operator/= (const double t)

  *Compound division assignment operator for scalar division.*
- double length () const

  *Calculate the length (magnitude) of the vector.*
- double length_squared () const

  *Calculate the squared length (magnitude) of the vector.*
- bool **operator==** (const vec3 &rhs) const
- bool near_zero () const

  *Near_zero bool check.*

### Static Public Member Functions

- static vec3 random ()

  *creates a vec3 with random values from [0,1)*
- static vec3 random (double min, double max)

  *creates a vec3 with ranodom values in chosen range [min, max)*

### Public Attributes

- double **e** [3]

### 5.17.1 Detailed Description

A class representing a 3D vector with x, y, and z components.

### 5.17.2 Constructor & Destructor Documentation

#### 5.17.2.1 vec3()

```
vec3::vec3 (
            double e0,
            double e1,
            double e2 )  [inline]
```

Constructor to create a vec3 with specified components.

**Parameters**

| | |
|---|---|
| *e0* | The x-component. |
| *e1* | The y-component. |
| *e2* | The z-component. |

### 5.17.3 Member Function Documentation

#### 5.17.3.1 length()

```
double vec3::length ( ) const  [inline]
```

Calculate the length (magnitude) of the vector.

**Returns**

The length of the vector as a double.

#### 5.17.3.2 length_squared()

```
double vec3::length_squared ( ) const  [inline]
```

Calculate the squared length (magnitude) of the vector.

**Returns**

The squared length of the vector as a double.

#### 5.17.3.3 near_zero()

```
bool vec3::near_zero ( ) const
```

Near_zero bool check.

**Returns**

true

false

#### 5.17.3.4 operator∗=()

```
vec3 & vec3::operator*= (
            const double t )
```

Compound multiplication assignment operator for scalar multiplication.

Definition of an ∗= operator.

**Parameters**

| | |
|---|---|
| *t* | The scalar value to multiply this vector. |

**Returns**

A reference to this vector after multiplication.

**Parameters**

| | |
|---|---|
| *t* | |

**Returns**

vec3&

#### 5.17.3.5 operator+=()

```
vec3 & vec3::operator+= (
            const vec3 & v )
```

Compound addition assignment operator for vector addition.

Definition of a += operator.

**Parameters**

| | |
|---|---|
| *v* | The vector to be added to this vector. |

**Returns**

A reference to this vector after addition.

**Parameters**

| | |
|---|---|
| *v* | |

**Returns**

vec3&

#### 5.17.3.6 operator-()

```
vec3 vec3::operator- ( ) const  [inline]
```

Unary negation operator, returns the negation of the vector.

**Returns**

The negated vector as a vec3.

#### 5.17.3.7 operator/=()

```
vec3 & vec3::operator/= (
            const double t )  [inline]
```

Compound division assignment operator for scalar division.

**Parameters**

| | |
|---|---|
| *t* | The scalar value to divide this vector. |

**Returns**

A reference to this vector after division.

#### 5.17.3.8 operator[]() [1/2]

```
double & vec3::operator[] (
            int i )  [inline]
```

Array subscript operator to access vector components (non-const version).

**Parameters**

| | |
|---|---|
| *i* | The index (0 for x, 1 for y, 2 for z). |

**Returns**

A reference to the component at the specified index as a double.

#### 5.17.3.9 operator[]() [2/2]

```
double vec3::operator[] (
            int i ) const  [inline]
```

Array subscript operator to access vector components.

**Parameters**

| | |
|---|---|
| *i* | The index (0 for x, 1 for y, 2 for z). |

**Returns**

The component at the specified index as a double (const version).

#### 5.17.3.10 random() [1/2]

```
static vec3 vec3::random ( )  [inline], [static]
```

creates a vec3 with random values from [0,1)

**Returns**

vec3

#### 5.17.3.11 random() [2/2]

```
static vec3 vec3::random (
            double min,
            double max )  [inline], [static]
```

creates a vec3 with ranodom values in chosen range [min, max)

**Parameters**

| | |
|---|---|
| *min* | |
| *max* | |

**Returns**

vec3

#### 5.17.3.12 x()

```
double vec3::x ( ) const  [inline]
```

Get the x-component of the vec3.

**Returns**

The x-component as a double.

#### 5.17.3.13 y()

```
double vec3::y ( ) const  [inline]
```

Get the y-component of the vec3.

**Returns**

The y-component as a double.

#### 5.17.3.14 z()

```
double vec3::z ( ) const  [inline]
```

Get the z-component of the vec3.

**Returns**

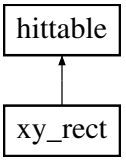
The z-component as a double.

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

- src/vec3.hpp
- src/vec3.cpp

## 5.18 xy_rect Class Reference

Inheritance diagram for xy_rect:



### Public Member Functions

- xy_rect (double _x0, double _x1, double _y0, double _y1, double _k, shared_ptr< material > mat)

  *Constructs an xy_rect with specified parameters.*
- virtual bool hit (const ray &r, interval ray_t, hit_record &rec) const override

  *Determines if a ray hits the rectangle.*

## 5.18.1 Constructor & Destructor Documentation

### 5.18.1.1 xy_rect()

```
xy_rect::xy_rect (
            double _x0,
            double _x1,
            double _y0,
            double _y1,
            double _k,
            shared_ptr< material > mat )
```

Constructs an xy_rect with specified parameters.

**Parameters**

| | |
|---|---|
| *_x0* | Minimum x-coordinate. |
| *_x1* | Maximum x-coordinate. |
| *_y0* | Minimum y-coordinate. |
| *_y1* | Maximum y-coordinate. |
| *_k* | z-coordinate of the rectangle. |
| *mat* | Material of the rectangle. |

## 5.18.2 Member Function Documentation

### 5.18.2.1 hit()

```
bool xy_rect::hit (
            const ray & r,
            interval ray_t,
            hit_record & rec ) const  [override], [virtual]
```

Determines if a ray hits the rectangle.

**Parameters**

| | |
|---|---|
| *r* | The ray. |
| *ray_t* | The valid interval for the ray parameter. |
| *rec* | The hit record to be filled if the ray hits the object. |

**Returns**

True if the ray hits the object, false otherwise.

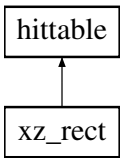
Implements hittable.

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

- src/rect.hpp
- src/rect.cpp

## 5.19 xz_rect Class Reference

Inheritance diagram for xz_rect:




### Public Member Functions

- xz_rect (double _x0, double _x1, double _z0, double _z1, double _k, shared_ptr< material > mat)

  *Constructs an xy_rect with specified parameters.*
- virtual bool hit (const ray &r, interval ray_t, hit_record &rec) const override

  *Determines if a ray hits the rectangle.*

### 5.19.1 Constructor & Destructor Documentation

#### 5.19.1.1 xz_rect()

```
xz_rect::xz_rect (
            double _x0,
            double _x1,
            double _z0,
            double _z1,
            double _k,
            shared_ptr< material > mat )
```

Constructs an xy_rect with specified parameters.

**Parameters**

| | |
|---|---|
| *_x0* | Minimum x-coordinate. |
| *_x1* | Maximum x-coordinate. |
| *_z0* | Minimum z-coordinate. |
| *_z1* | Maximum z-coordinate. |
| *_k* | y-coordinate of the rectangle. |
| *mat* | Material of the rectangle. |

### 5.19.2 Member Function Documentation

#### 5.19.2.1 hit()

```
bool xz_rect::hit (
            const ray & r,
            interval ray_t,
            hit_record & rec ) const  [override], [virtual]
```

Determines if a ray hits the rectangle.

**Parameters**

| | |
|---|---|
| *r* | The ray. |
| *ray_t* | The valid interval for the ray parameter. |
| *rec* | The hit record to be filled if the ray hits the object. |

**Returns**

True if the ray hits the object, false otherwise.

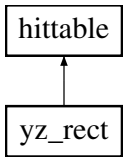
Implements hittable.

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

- src/rect.hpp
- src/rect.cpp

## 5.20 yz_rect Class Reference

Inheritance diagram for yz_rect:




**Public Member Functions**

- yz_rect (double _y0, double _y1, double _z0, double _z1, double _k, shared_ptr< material > mat)

  *Constructs an xy_rect with specified parameters.*
- virtual bool hit (const ray &r, interval ray_t, hit_record &rec) const override

  *Determines if a ray hits the rectangle.*

### 5.20.1 Constructor & Destructor Documentation

#### 5.20.1.1 yz_rect()

```
yz_rect::yz_rect (
            double _y0,
            double _y1,
            double _z0,
            double _z1,
            double _k,
            shared_ptr< material > mat )
```

Constructs an xy_rect with specified parameters.

**Parameters**

| | |
|---|---|
| *_y0* | Minimum y-coordinate. |
| *_y1* | Maximum y-coordinate. |
| *_z0* | Minimum z-coordinate. |
| *_z1* | Maximum z-coordinate. |
| *_k* | x-coordinate of the rectangle. |
| *mat* | Material of the rectangle. |

### 5.20.2 Member Function Documentation

#### 5.20.2.1 hit()

```
bool yz_rect::hit (
            const ray & r,
            interval ray_t,
            hit_record & rec ) const  [override], [virtual]
```

Determines if a ray hits the rectangle.

**Parameters**

| | |
|---|---|
| *r* | The ray. |
| *ray_t* | The valid interval for the ray parameter. |
| *rec* | The hit record to be filled if the ray hits the object. |

**Returns**

True if the ray hits the object, false otherwise.

Implements hittable.

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

- src/rect.hpp
- src/rect.cpp

# Chapter 6

# File Documentation

## 6.1 box.hpp

```
00001 #pragma once
00002
00003 #include "rect.hpp"
00004 #include "hittable_list.hpp"
00005
00010 class box : public hittable {
00011   public:
00012     box() {}
00013
00020     box(const point3& p0, const point3& p1, shared_ptr<material> ptr);
00021
00033     virtual bool hit(const ray& r, interval ray_t, hit_record& rec) const override;
00034
00035   public:
00036     point3 box_min;
00037     point3 box_max;
00038     hittable_list sides;
00039 };
```

## 6.2 src/camera.hpp File Reference

The camera class represents a camera used for ray tracing. Heavily inspired by Peter Shirley https://raytracing.github.io/v3/books/RayTracingInOneWeekend.html.

```
#include "utility.hpp"
#include "ray.hpp"
#include "vec3.hpp"
```

### Classes

- class camera

  *Represents a movable camera in the world.*

### 6.2.1 Detailed Description

The camera class represents a camera used for ray tracing. Heavily inspired by Peter Shirley https://raytracing.github.io/v3/books/RayTracingInOneWeekend.html.

#### Author

Lauri Wilppu ( lauri.wilppu@gmail.com)

#### Version

0.1

#### Date

2023-11-07

#### Copyright

Copyright (c) 2023

## 6.3 camera.hpp

Go to the documentation of this file.

```
00001 #pragma once
00002 #include "utility.hpp"
00003 #include "ray.hpp"
00004 #include "vec3.hpp"
00005
00021 class camera {
00022  public:
00034   camera(
00035       point3 lookfrom,
00036       point3 lookat,
00037       vec3 vup,
00038       double vfov,
00039       double aspect_ratio,
00040       double aperture,
00041       double focus_dist,
00042       double _time0,
00043       double _time1
00044       );
00045
00053   ray get_ray(double s, double t) const;
00054
00059   point3 getOrigin() const {
00060     return origin;
00061   }
00062
00067   double getLensRadius() const {
00068     return lens_radius;
00069   }
00070
00071  private:
00072   point3 origin;
00073   point3 lower_left_corner;
00074   vec3 horizontal;
00075   vec3 vertical;
00076   vec3 u, v, w;
00077   double lens_radius;
00078   double time0, time1;
00079 };
```

## 6.4 hittable.hpp

```
00001 #pragma once
00002 #include "ray.hpp"
00003 #include "interval.hpp"
00004 class material;
00005
00010 class hit_record {
00011   public:
00012     point3 p;
00013     vec3 normal;
00014     shared_ptr<material> mat_ptr;
00015     double t;
00016     double u, v;
00017     bool front_face;
00026     void set_face_normal(const ray& r, const vec3& outward_normal);
00027 };
00028
00029
00034 class hittable {
00035   public:
00036     virtual ~hittable() = default;
00037
00045     virtual bool hit(const ray& r, interval ray_t, hit_record& rec) const = 0;
00046 };
```

## 6.5 hittable_list.hpp

```
00001 #pragma once
00002 #include "hittable.hpp"
00003
00004 #include <memory>
00005 #include <vector>
00006
00007 using std::shared_ptr;
00008 using std::make_shared;
00009
00014 class hittable_list : public hittable {
00015   public:
00016     std::vector<shared_ptr<hittable>> objects;
00021     hittable_list() {}
00022
00027     hittable_list(shared_ptr<hittable> object);
00028
00032     void clear();
00033
00038     void add(shared_ptr<hittable> object);
00039
00040
00049     bool hit(const ray& r, interval ray_t, hit_record& rec) const override;
00050 };
```

## 6.6 interval.hpp

```
00001 #pragma once
00002 #include <limits>
00003
00004 const double infinity = std::numeric_limits<double>::infinity();
00005
00006
00011 class interval {
00012   public:
00013     double min, max;
00019     interval();
00020
00028     interval(double _min, double _max);
00029
00037     bool contains(double x) const;
00038
00046     bool surrounds(double x) const;
00047
00054     double clamp(double x) const;
00055
00056     static const interval empty, universe;
00057 };
00058
00059 const static interval empty    (+infinity, -infinity);
00060 const static interval universe(-infinity, +infinity);
```

## 6.7 src/material.hpp File Reference

The material class represents materials used for ray tracing. Heavily inspired by Peter Shirley https://raytracing.github.io/v3/books/RayTracingInOneWeekend.html.

```
#include "ray.hpp"
#include "hittable.hpp"
#include "utility.hpp"
```

### Classes

- class material

  *virtual class for all the materials.*
- class lambertian

  *Class representing Lambertian diffusion.*
- class metal

  *Class representing metal material.*
- class dielectric

  *Class representing dielectric material.*
- class diffuse_light

  *Represents a light source material.*

### 6.7.1 Detailed Description

The material class represents materials used for ray tracing. Heavily inspired by Peter Shirley https://raytracing.github.io/v3/books/RayTracingInOneWeekend.html.

**Author**

Verneri Hakkarainen

**Version**

0.1

**Date**

2023-11-17

**Copyright**

Copyright (c) 2023

## 6.8 material.hpp

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include "ray.hpp"
00004 #include "hittable.hpp"
00005 #include "utility.hpp"
00006
00024 class material {
00025     public:
00026         virtual color emitted(double u, double v, const point3& p) const {
00027             return color(0,0,0);
00028         }
00029         virtual bool scatter(
00030             const ray& r_in, const hit_record& rec, color& attenuation, ray& scattered
00031         ) const = 0;
00032 };
00033
00038 class lambertian : public material {
00039     public:
00040
00046         lambertian(const color& a);
00047
00058         virtual bool scatter(
00059             const ray& r_in, const hit_record& rec, color& attenuation, ray& scattered
00060         ) const override;
00061
00062     private:
00063         color albedo;
00064 };
00065
00070 class metal : public material {
00071     public:
00072
00079         metal(const color& a, double f);
00080
00092         virtual bool scatter(
00093             const ray& r_in, const hit_record& rec, color& attenuation, ray& scattered
00094         ) const override;
00095
00096     private:
00097         color albedo;
00098         double fuzz;
00099 };
00100
00105 class dielectric : public material {
00106     public:
00107
00113         dielectric(double index_of_reftraction);
00114
00125         virtual bool scatter(
00126             const ray& r_in, const hit_record& rec, color& attenuation, ray& scattered
00127         ) const override;
00128
00129     private:
00130         double ir; // index of refraction
00131
00140         static double reflectance(double cosine, double ref_idx) {
00141             auto r0 = (1-ref_idx) / (1+ref_idx);
00142             r0 = r0*r0;
00143             return r0 + (1-r0)*pow((1-cosine),5);
00144         }
00145 };
00146
00151 class diffuse_light : public material  {
00152     public:
00153
00158         diffuse_light(color c);
00159
00160         virtual bool scatter(
00161             const ray& r_in, const hit_record& rec, color& attenuation, ray& scattered
00162         ) const override;
00163
00172         virtual color emitted(double u, double v, const point3& p) const override;
00173
00174     private:
00175         shared_ptr<color> emit;
00176 };
```

## 6.9 moving_sphere.hpp

```
00001 #pragma once
```

```
00002
00003 #include "hittable.hpp"
00004
00009 class moving_sphere : public hittable {
00010     public:
00011         moving_sphere() {}
00012
00024         moving_sphere(
00025             point3 cen0, point3 cen1, double _time0, double _time1, double r, shared_ptr<material> m)
00026             : center0(cen0), center1(cen1), time0(_time0), time1(_time1), radius(r), mat_ptr(m)
00027         {};
00028
00039         virtual bool hit(const ray& r, interval ray_t, hit_record& rec) const override;
00040
00045         point3 center(double time) const;
00046
00047     private:
00048         point3 center0, center1;
00049         double time0, time1;
00050         double radius;
00051         shared_ptr<material> mat_ptr;
00052 };
```

## 6.10 src/ray.cpp File Reference

Necessary class definitions of ray with an origin point and a direction vector.

`#include "ray.hpp"`

### 6.10.1 Detailed Description

Necessary class definitions of ray with an origin point and a direction vector.

**Author**

Teodors Kerimovs ( teodors.kerimovs@gmail.com)

**Version**

0.1

**Date**

2023-12-10

**Copyright**

Copyright (c) 2023

## 6.11 src/ray.hpp File Reference

`#include "vec3.hpp"`

### Classes

- class ray

  *A class representing a ray with an origin point and a direction vector. Heavily inspired by Peter Shirley https://raytracing.github.io/v3/books/RayTracingInOneWeekend.html.*

### 6.11.1 Detailed Description

**Author**

Teodors Kerimovs ( teodors.kerimovs@gmail.com)

## 6.12 ray.hpp

Go to the documentation of this file.

```
00001 #pragma once
00015 #include "vec3.hpp"
00016
00017
00018 class ray {
00019 public:
00023     ray() {}
00024
00032     ray(const point3& origin, const vec3& direction, double time = 0.0)
00033         : orig(origin), dir(direction), tm(time)
00034     {}
00035
00040     point3 origin() const  { return orig; }
00041
00046     vec3 direction() const { return dir; }
00047
00052     double time() const    { return tm; }
00053
00059     point3 at(double t) const;
00060
00061 public:
00062     point3 orig; // The origin point of the ray.
00063     vec3 dir;    // The direction vector of the ray.
00064     double tm; // Time of vector appearing.
00065 };
```

## 6.13 rect.hpp

```
00001 #pragma once
00002
00003 #include "utility.hpp"
00004 #include "hittable.hpp"
00005
00006
00011 class xy_rect : public hittable {
00012   public:
00013     xy_rect() {}
00014
00024     xy_rect(double _x0, double _x1, double _y0, double _y1, double _k,
00025       shared_ptr<material> mat);
00026
00036     virtual bool hit(const ray& r, interval ray_t, hit_record& rec) const override;
00037
00038   private:
00039     shared_ptr<material> mp;
00040     double x0, x1, y0, y1, k;
00041 };
00042
00048 class xz_rect : public hittable {
00049   public:
00050     xz_rect() {}
00051
00061     xz_rect(double _x0, double _x1, double _z0, double _z1, double _k,
00062       shared_ptr<material> mat);
00063
```

```
00073     virtual bool hit(const ray& r, interval ray_t, hit_record& rec) const override;
00074
00075   private:
00076     shared_ptr<material> mp;
00077     double x0, x1, z0, z1, k;
00078 };
00079
00085 class yz_rect : public hittable {
00086   public:
00087   yz_rect() {}
00088
00098   yz_rect(double _y0, double _y1, double _z0, double _z1, double _k,
00099     shared_ptr<material> mat);
00100
00110   virtual bool hit(const ray& r, interval ray_t, hit_record& rec) const override;
00111   private:
00112     shared_ptr<material> mp;
00113     double y0, y1, z0, z1, k;
00114 };
```

## 6.14 render.hpp

```
00001 #pragma once
00002
00003 #include "vec3.hpp"
00004 #include "interval.hpp"
00005 #include "fstream"
00006 #include "ray.hpp"
00007 #include "hittable_list.hpp"
00008 #include "camera.hpp"
00009 #include "material.hpp"
00010 #include "interval.hpp"
00011 #include "fstream"
00012 #include <iostream>
00013 #include <thread>
00014 #include <mutex>
00015 #include <chrono>
00016
00022 struct PixelInfo {
00023     int x;
00024     int y;
00025     color pixel_color;
00026 };
00027
00036 color ray_color(const ray& r, const color& background, const hittable& world, int depth, bool
      skylight);
00037
00038
00049 void render(int image_width, int image_height, int samples_per_pixel, int max_depth, camera cam,
      hittable_list& world, color background, bool skylight);
```

## 6.15 sphere.hpp

```
00001 #pragma once
00002 #include "hittable.hpp"
00003 #include "vec3.hpp"
00004 #include "interval.hpp"
00005
00011 class sphere : public hittable {
00012   public:
00019     sphere(point3 _center, double _radius, shared_ptr<material> m);
00020
00030     bool hit(const ray& r, interval ray_t, hit_record& rec) const override;
00031
00032   private:
00033     point3 center;
00034     double radius;
00035     shared_ptr<material> mat_ptr;
00036 };
```

## 6.16 utility.hpp

```
00001 #pragma once
00002
00003
```

```
00004 #include <cmath>
00005 #include <cstdlib>
00006 #include <limits>
00007 #include <memory>
00008
00009
00010 // Usings
00011
00012 using std::shared_ptr;
00013 using std::make_shared;
00014 using std::sqrt;
00015
00016 // Constants
00017 const double pi = 3.1415926535897932385;
00018
00019 // Utility Functions
00020
00026 double degrees_to_radians(double degrees);
00027
00035 double clamp(double x, double min, double max);
00036
00041 double random_double();
00042
00049 double random_double(double min, double max);
00050
00057 int random_int(int min, int max);
```

## 6.17 src/vec3.cpp File Reference

Cpp files with definitions of vec3 members and non-members.

```
#include "vec3.hpp"
```

### Functions

- std::ostream & operator<< (std::ostream &out, const vec3 &v)

  *Overloaded output stream operator to print a vec3.*
- vec3 operator+ (const vec3 &u, const vec3 &v)

  *Overloaded addition operator for vector addition.*
- vec3 operator- (const vec3 &u, const vec3 &v)

  *Overloaded subtraction operator for vector subtraction.*
- vec3 operator* (const vec3 &u, const vec3 &v)

  *Overloaded multiplication operator for component-wise vector multiplication.*
- vec3 operator* (double t, const vec3 &v)

  *Overloaded multiplication operator for scalar multiplication.*
- vec3 operator* (const vec3 &v, double t)

  *Overloaded multiplication operator for scalar multiplication (reversed order).*
- vec3 operator/ (vec3 v, double t)

  *Overloaded division operator for scalar division.*
- double dot (const vec3 &u, const vec3 &v)

  *Calculate the dot product of two vectors.*
- vec3 cross (const vec3 &u, const vec3 &v)

  *Calculate the cross product of two vectors.*
- vec3 unit_vector (vec3 v)

  *Calculate the unit vector (normalized vector) of a given vector.*
- vec3 refract (const vec3 &uv, const vec3 &n, double etai_over_etat)

  *Computes the refraction direction for a given incident vector (uv), surface normal (n), and the ratio of refractive indices (etai_over_etat).*
- vec3 random_in_unit_sphere ()

*chooses a random point inside a unit sphere. Used in reflections of different materials.*

- vec3 random_unit_vector ()

  *Chooses a random point on the surface unit circle cirlce.*

- vec3 random_in_hemisphere (const vec3 &normal)

  *TODO different kind of diffuse.*

- vec3 **reflect** (const vec3 &v, const vec3 &n)
- vec3 random_in_unit_disk ()

  *Generates a random point in the unit disk.*

### 6.17.1 Detailed Description

Cpp files with definitions of vec3 members and non-members.

**Author**

Teodors Kerimovs ( teodors.kerimovs@gmail.com)

**Version**

0.1

**Date**

2023-12-10

**Copyright**

Copyright (c) 2023

### 6.17.2 Function Documentation

#### 6.17.2.1 cross()

```
vec3 cross (
            const vec3 & u,
            const vec3 & v )
```

Calculate the cross product of two vectors.

**Parameters**

| | |
|---|---|
| *u* | The first vector. |
| *v* | The second vector. |

**Returns**

The cross product of the two vectors as a vec3.

#### 6.17.2.2 dot()

```
double dot (
            const vec3 & u,
            const vec3 & v )
```

Calculate the dot product of two vectors.

**Parameters**

| | |
|---|---|
| *u* | The first vector. |
| *v* | The second vector. |

**Returns**

The dot product of the two vectors as a double.

#### 6.17.2.3 operator∗() [1/3]

```
vec3 operator* (
            const vec3 & u,
            const vec3 & v )
```

Overloaded multiplication operator for component-wise vector multiplication.

**Parameters**

| | |
|---|---|
| *u* | The first vector. |
| *v* | The second vector. |

**Returns**

The result of multiplying the two vectors component-wise as a vec3.

#### 6.17.2.4 operator∗() [2/3]

```
vec3 operator* (
            const vec3 & v,
            double t )
```

Overloaded multiplication operator for scalar multiplication (reversed order).

**Parameters**

| | |
|---|---|
| *v* | The vector to be multiplied. |
| *t* | The scalar value. |

**Returns**

The result of multiplying the vector by the scalar as a vec3.

#### 6.17.2.5 operator*() [3/3]

```
vec3 operator* (
            double t,
            const vec3 & v )
```

Overloaded multiplication operator for scalar multiplication.

**Parameters**

| | |
|---|---|
| *t* | The scalar value. |
| *v* | The vector to be multiplied. |

**Returns**

The result of multiplying the vector by the scalar as a vec3.

#### 6.17.2.6 operator+()

```
vec3 operator+ (
            const vec3 & u,
            const vec3 & v )
```

Overloaded addition operator for vector addition.

**Parameters**

| | |
|---|---|
| *u* | The first vector. |
| *v* | The second vector. |

**Returns**

The result of adding the two vectors as a vec3.

#### 6.17.2.7 operator-()

```
vec3 operator- (
            const vec3 & u,
            const vec3 & v )
```

Overloaded subtraction operator for vector subtraction.

**Parameters**

| | |
|---|---|
| *u* | The first vector. |
| *v* | The second vector. |

**Returns**

The result of subtracting the second vector from the first as a vec3.

#### 6.17.2.8 operator/()

```
vec3 operator/ (
            vec3 v,
            double t )
```

Overloaded division operator for scalar division.

**Parameters**

| | |
|---|---|
| *v* | The vector to be divided. |
| *t* | The scalar value. |

**Returns**

The result of dividing the vector by the scalar as a vec3.

#### 6.17.2.9 operator<<()

```
std::ostream & operator<< (
            std::ostream & out,
            const vec3 & v )
```

Overloaded output stream operator to print a vec3.

**Parameters**

| | |
|---|---|
| *out* | The output stream. |
| *v* | The vec3 to be printed. |

**Returns**

The output stream after printing the vec3.

#### 6.17.2.10 random_in_hemisphere()

```
vec3 random_in_hemisphere (
            const vec3 & normal )
```

TODO different kind of diffuse.

**Parameters**

| | |
|---|---|
| *normal* | |

**Returns**

** vec3

#### 6.17.2.11 random_in_unit_disk()

```
vec3 random_in_unit_disk ( )
```

Generates a random point in the unit disk.

This function continues generating points until it finds one within the unit disk.

**Returns**

A random point in the unit disk.

#### 6.17.2.12 random_in_unit_sphere()

```
vec3 random_in_unit_sphere ( )
```

chooses a random point inside a unit sphere. Used in reflections of different materials.

**Returns**

vec3

#### 6.17.2.13 random_unit_vector()

```
vec3 random_unit_vector ( )
```

Chooses a random point on the surface unit circle cirlce.

**Returns**

vec3

#### 6.17.2.14 refract()

```
vec3 refract (
            const vec3 & uv,
            const vec3 & n,
            double etai_over_etat )
```

Computes the refraction direction for a given incident vector (uv), surface normal (n), and the ratio of refractive indices (etai_over_etat).

This function ensures the cosine of the angle between the incident vector and the normal is within [-1, 1], and then computes the refracted ray using Snell's Law.

**Parameters**

| | |
|---|---|
| *uv* | |
| *n* | |
| *etai_over_etat* | |

**Returns**

vec3

#### 6.17.2.15 unit_vector()

```
vec3 unit_vector (
            vec3 v )
```

Calculate the unit vector (normalized vector) of a given vector.

**Parameters**

| | |
|---|---|
| *v* | The vector to be normalized. |

**Returns**

The unit vector (normalized vector) of the input vector as a vec3.

## 6.18 src/vec3.hpp File Reference

A C++ header file defining a 3D vector class and related utility functions. Heavily inspired by Peter Shirley https://raytracing.github.io/v3/books/RayTracingInOneWeekend.html.

```
#include <cmath>
#include <iostream>
#include "utility.hpp"
```

### Classes

- class vec3

  *A class representing a 3D vector with x, y, and z components.*

### Typedefs

- using **point3** = vec3
- using **color** = vec3

### Functions

- std::ostream & operator<< (std::ostream &out, const vec3 &v)

  *Overloaded output stream operator to print a vec3.*
- vec3 operator+ (const vec3 &u, const vec3 &v)

  *Overloaded addition operator for vector addition.*
- vec3 operator- (const vec3 &u, const vec3 &v)

  *Overloaded subtraction operator for vector subtraction.*
- vec3 operator* (const vec3 &u, const vec3 &v)

  *Overloaded multiplication operator for component-wise vector multiplication.*
- vec3 operator* (double t, const vec3 &v)

  *Overloaded multiplication operator for scalar multiplication.*
- vec3 operator* (const vec3 &v, double t)

  *Overloaded multiplication operator for scalar multiplication (reversed order).*
- vec3 operator/ (vec3 v, double t)

  *Overloaded division operator for scalar division.*
- double dot (const vec3 &u, const vec3 &v)

  *Calculate the dot product of two vectors.*
- vec3 cross (const vec3 &u, const vec3 &v)

  *Calculate the cross product of two vectors.*
- vec3 unit_vector (vec3 v)

  *Calculate the unit vector (normalized vector) of a given vector.*
- vec3 random_in_unit_sphere ()

  *chooses a random point inside a unit sphere. Used in reflections of different materials.*
- vec3 random_unit_vector ()

  *Chooses a random point on the surface unit circle cirlce.*
- vec3 random_in_hemisphere (const vec3 &normal)

  *TODO different kind of diffuse.*
- vec3 **reflect** (const vec3 &v, const vec3 &n)
- vec3 random_in_unit_disk ()

  *Generates a random point in the unit disk.*
- vec3 refract (const vec3 &uv, const vec3 &n, double etai_over_etat)

  *Computes the refraction direction for a given incident vector (uv), surface normal (n), and the ratio of refractive indices (etai_over_etat).*

## 6.18.1 Detailed Description

A C++ header file defining a 3D vector class and related utility functions. Heavily inspired by Peter Shirley https://raytracing.github.io/v3/books/RayTracingInOneWeekend.html.

**Author**

Teodors Kerimovs ( teodors.kerimovs@gmail.com)

**Version**

0.1

**Date**

2023-11-06

**Copyright**

Copyright (c) 2023

## 6.18.2 Function Documentation

### 6.18.2.1 cross()

```
vec3 cross (
            const vec3 & u,
            const vec3 & v )
```

Calculate the cross product of two vectors.

**Parameters**

| | |
|---|---|
| *u* | The first vector. |
| *v* | The second vector. |

**Returns**

The cross product of the two vectors as a vec3.

### 6.18.2.2 dot()

```
double dot (
            const vec3 & u,
            const vec3 & v )
```

Calculate the dot product of two vectors.

**Parameters**

| | |
|---|---|
| *u* | The first vector. |
| *v* | The second vector. |

**Returns**

The dot product of the two vectors as a double.

### 6.18.2.3 operator∗() [1/3]

```
vec3 operator* (
            const vec3 & u,
            const vec3 & v )
```

Overloaded multiplication operator for component-wise vector multiplication.

**Parameters**

| | |
|---|---|
| *u* | The first vector. |
| *v* | The second vector. |

**Returns**

The result of multiplying the two vectors component-wise as a vec3.

#### 6.18.2.4 operator∗() [2/3]

```
vec3 operator* (
            const vec3 & v,
            double t )
```

Overloaded multiplication operator for scalar multiplication (reversed order).

**Parameters**

| | |
|---|---|
| *v* | The vector to be multiplied. |
| *t* | The scalar value. |

**Returns**

The result of multiplying the vector by the scalar as a vec3.

#### 6.18.2.5 operator∗() [3/3]

```
vec3 operator* (
            double t,
            const vec3 & v )
```

Overloaded multiplication operator for scalar multiplication.

**Parameters**

| | |
|---|---|
| *t* | The scalar value. |
| *v* | The vector to be multiplied. |

**Returns**

The result of multiplying the vector by the scalar as a vec3.

#### 6.18.2.6 operator+()

```
vec3 operator+ (
            const vec3 & u,
            const vec3 & v )
```

Overloaded addition operator for vector addition.

**Parameters**

| | |
|---|---|
| *u* | The first vector. |
| *v* | The second vector. |

**Returns**

The result of adding the two vectors as a vec3.

#### 6.18.2.7 operator-()

```
vec3 operator- (
            const vec3 & u,
            const vec3 & v )
```

Overloaded subtraction operator for vector subtraction.

**Parameters**

| | |
|---|---|
| *u* | The first vector. |
| *v* | The second vector. |

**Returns**

The result of subtracting the second vector from the first as a vec3.

#### 6.18.2.8 operator/()

```
vec3 operator/ (
            vec3 v,
            double t )
```

Overloaded division operator for scalar division.

**Parameters**

| | |
|---|---|
| *v* | The vector to be divided. |
| *t* | The scalar value. |

**Returns**

The result of dividing the vector by the scalar as a vec3.

#### 6.18.2.9 operator<<()

```
std::ostream & operator<< (
            std::ostream & out,
            const vec3 & v )
```

Overloaded output stream operator to print a vec3.

**Parameters**

| | |
|---|---|
| *out* | The output stream. |
| *v* | The vec3 to be printed. |

**Returns**

The output stream after printing the vec3.

#### 6.18.2.10 random_in_hemisphere()

```
vec3 random_in_hemisphere (
            const vec3 & normal )
```

TODO different kind of diffuse.

**Parameters**

| | |
|---|---|
| *normal* | |

**Returns**

** vec3

#### 6.18.2.11 random_in_unit_disk()

```
vec3 random_in_unit_disk ( )
```

Generates a random point in the unit disk.

This function continues generating points until it finds one within the unit disk.

**Returns**

A random point in the unit disk.

#### 6.18.2.12 random_in_unit_sphere()

```
vec3 random_in_unit_sphere ( )
```

chooses a random point inside a unit sphere. Used in reflections of different materials.

**Returns**

vec3

#### 6.18.2.13 random_unit_vector()

```
vec3 random_unit_vector ( )
```

Chooses a random point on the surface unit circle cirlce.

**Returns**

vec3

#### 6.18.2.14 refract()

```
vec3 refract (
            const vec3 & uv,
            const vec3 & n,
            double etai_over_etat )
```

Computes the refraction direction for a given incident vector (uv), surface normal (n), and the ratio of refractive indices (etai_over_etat).

This function ensures the cosine of the angle between the incident vector and the normal is within [-1, 1], and then computes the refracted ray using Snell's Law.

**Parameters**

| | |
|---|---|
| *uv* | The incident vector. |
| *n* | The surface normal. |
| *etai_over_etat* | The ratio of refractive indices. |

**Returns**

The refracted ray direction.

This function ensures the cosine of the angle between the incident vector and the normal is within [-1, 1], and then computes the refracted ray using Snell's Law.

**Parameters**

| | |
|---|---|
| *uv* | |
| *n* | |
| *etai_over_etat* | |

**Returns**

vec3

#### 6.18.2.15 unit_vector()

```
vec3 unit_vector (
            vec3 v )
```

Calculate the unit vector (normalized vector) of a given vector.

**Parameters**

| | |
|---|---|
| *v* | The vector to be normalized. |

**Returns**

The unit vector (normalized vector) of the input vector as a vec3.

## 6.19 vec3.hpp

Go to the documentation of this file.

```
00001 #pragma once
00014 #include <cmath>
00015 #include <iostream>
00016 #include "utility.hpp"
00017
00018 using std::sqrt;
00023 class vec3 {
00024     public:
00028         vec3() : e{0,0,0} {}
00035         vec3(double e0, double e1, double e2) : e{e0, e1, e2} {}
00040         double x() const { return e[0]; }
00045         double y() const { return e[1]; }
00050         double z() const { return e[2]; }
00051
00056         vec3 operator-() const { return vec3(-e[0], -e[1], -e[2]); }
00057
00063         double operator[](int i) const { return e[i]; }
00064
00070         double& operator[](int i) { return e[i]; }
00071
00072
00078         vec3& operator+=(const vec3 &v);
00079
00085         vec3& operator*=(const double t);
00086
00092         vec3& operator/=(const double t) {
00093             return *this *= 1/t;
00094         }
00095
00100         double length() const {
00101             return sqrt(length_squared());
00102         }
00103
00108         double length_squared() const {
00109             return e[0]*e[0] + e[1]*e[1] + e[2]*e[2];
00110         }
00111
00117     inline static vec3 random() {
00118     return vec3(random_double(), random_double(), random_double());
00119     }
00120
00128     inline static vec3 random(double min, double max) {
00129         return vec3(random_double(min,max), random_double(min,max), random_double(min,max));
00130     }
00131
00132     inline bool operator==(const vec3& rhs) const {
00133         return e[0] == rhs.e[0] && e[1] == rhs.e[1] && e[2] == rhs.e[2];
00134     }
00135
00136     bool near_zero() const;
00137
00138     public:
00139         double e[3]; // Components of the vec3.
00140 };
00141
00142 // Type aliases for vec3
00143 using point3 = vec3;   // 3D point
00144 using color = vec3;    // RGB color
00145
00146
00147
00148 // vec3 Utility Functions
00155 std::ostream& operator<<(std::ostream &out, const vec3 &v);
00156
00163 vec3 operator+(const vec3 &u, const vec3 &v);
00164
```

```
00171 vec3 operator-(const vec3 &u, const vec3 &v);
00172
00179 vec3 operator*(const vec3 &u, const vec3 &v);
00180
00187 vec3 operator*(double t, const vec3 &v);
00188
00195 vec3 operator*(const vec3 &v, double t);
00196
00203 vec3 operator/(vec3 v, double t);
00204
00211 double dot(const vec3 &u, const vec3 &v);
00218 vec3 cross(const vec3 &u, const vec3 &v);
00219
00225 vec3 unit_vector(vec3 v);
00226
00227
00233 vec3 random_in_unit_sphere();
00234
00240 vec3 random_unit_vector();
00241
00248 vec3 random_in_hemisphere(const vec3& normal);
00249
00250 vec3 reflect(const vec3& v, const vec3& n);
00251
00259 vec3 random_in_unit_disk();
00260
00273 vec3 refract(const vec3& uv, const vec3& n, double etai_over_etat);
```

# Index