Zero-Renderer

0.1

Generated by Doxygen 1.9.5

Chapter 1

README

Chapter 2

Todo List

Member BoundingBoxUnionIntersect (const BoundingBox3< BaseType > &b1, const BoundingBox3< BaseType > &b2)

The function name "BoundingBoxOverlap" might better

Class CoordConvertor

A namespace contains function might better?

Class RGBSpectrum

To be finished

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

Chapter 5

Class Index

5.1 Class List

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

Chapter 6

File Index

6.1 File List

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

Chapter 7

Module Documentation

7.1 Geometry

Classes

- class BoundingBox3< BaseType >

  Axis-aligned bounding box 3D base type, the box edges are mutually perpendicular and aligned to axes.
- class CoordConvertor

  Provides transformations between different coordinate systems.

7.1.1 Detailed Description

Chapter 8

Class Documentation

8.1 BoundingBox3< BaseType > Class Template Reference

Axis-aligned bounding box 3D base type, the box edges are mutually perpendicular and aligned to axes.

```
#include <BoundingBox.h>
```

Public Member Functions

- std::optional< Point2d > Intersection (const Ray &r)

  Intersect of ray and bounding box.
- double SurfaceArea ()

  The surface area of the bounding box.

8.1.1 Detailed Description

template<typename BaseType>
class BoundingBox3< BaseType >

Axis-aligned bounding box 3D base type, the box edges are mutually perpendicular and aligned to axes.

Parameters

BaseType	the type of the bounding box, int/float/double.

8.1.2 Member Function Documentation

8.1.2.1 Intersection()

```
template<typename BaseType >
std::optional< Point2d > BoundingBox3< BaseType >::Intersection (
            const Ray & r )  [inline]
```

Intersect of ray and bounding box.

Returns

A pointer to the pair (t0, t1)

8.1.2.2 SurfaceArea()

```
template<typename BaseType >
double BoundingBox3< BaseType >::SurfaceArea ( )  [inline]
```

The surface area of the bounding box.

Returns

A double represents the bounding box's surface area

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

- /mnt/renderer/Zero/src/CoreLayer/Geometry/BoundingBox.h

8.2 CoefficientSpectrum< nSamples > Class Template Reference

Spectrum of nSamples sample points.

```
#include <Color.h>
```

Public Member Functions

- **CoefficientSpectrum** ()

  All coefficients initialized as 0.0f.
- **CoefficientSpectrum** (double val)

  All coefficients initialized as val.
- double operator[ ] (int i) const
- double & operator[ ] (int i)
- CoefficientSpectrum operator/ (const CoefficientSpectrum &s) const
- CoefficientSpectrum & operator/= (const CoefficientSpectrum &s)
- CoefficientSpectrum operator/ (double v) const
- CoefficientSpectrum & operator/= (double v)
- virtual XYZ3 toXYZ3 () const

Friends

- CoefficientSpectrum sqrt (const CoefficientSpectrum &s)

8.2.1 Detailed Description

template<int nSamples>
class CoefficientSpectrum< nSamples >

Spectrum of nSamples sample points.

8.2.2 Member Function Documentation

8.2.2.1 operator/() [1/2]

```
template<int nSamples>
CoefficientSpectrum CoefficientSpectrum< nSamples >::operator/ (
             const CoefficientSpectrum< nSamples > & s ) const  [inline]
```

Attention

There may be NaNs in result

8.2.2.2 operator/() [2/2]

```
template<int nSamples>
CoefficientSpectrum CoefficientSpectrum< nSamples >::operator/ (
             double v ) const  [inline]
```

Attention

There may be NaNs in result

8.2.2.3 operator/=() [1/2]

```
template<int nSamples>
CoefficientSpectrum & CoefficientSpectrum< nSamples >::operator/= (
             const CoefficientSpectrum< nSamples > & s )  [inline]
```

Attention

There may be NaNs in result

8.2.2.4 operator/=() [2/2]

```
template<int nSamples>
CoefficientSpectrum & CoefficientSpectrum< nSamples >::operator/= (
             double v )  [inline]
```

Attention

There may be NaNs in result

8.2.2.5 operator[]() [1/2]

```
template<int nSamples>
double & CoefficientSpectrum< nSamples >::operator[] (
             int i )  [inline]
```

Attention

No bounding check

8.2.2.6 operator[]() [2/2]

```
template<int nSamples>
double CoefficientSpectrum< nSamples >::operator[] (
             int i ) const  [inline]
```

Attention

No bounding check

8.2.2.7 toXYZ3()

```
template<int nSamples>
virtual XYZ3 CoefficientSpectrum< nSamples >::toXYZ3 ( ) const  [inline], [virtual]
```

Attention

This function is just used for debugging

Reimplemented in SampledSpectrum.

8.2.3 Friends And Related Function Documentation

8.2.3.1 sqrt

```
template<int nSamples>
CoefficientSpectrum sqrt (
            const CoefficientSpectrum< nSamples > & s )  [friend]
```

Attention

Does not check if each component is greater than 0

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

- /mnt/renderer/Zero/src/CoreLayer/ColorSpace/Color.h

8.3 CoordConvertor Class Reference

Provides transformations between different coordinate systems.

```
#include <CoordConvertor.h>
```

Static Public Member Functions

- static Point3d cartesian2Spherical (const Point2d &p)

  Convert a spherical coordinate (ϕ, θ) *to cartesian coordinate.*
- static Vec3d **cartesian2SphericalVec** (const Point2d &p)

  Convert $[0, 1]^2$ *to a point on unit sphere,.*

8.3.1 Detailed Description

Provides transformations between different coordinate systems.

Todo A namespace contains function might better?

8.3.2 Member Function Documentation

8.3.2.1 cartesian2Spherical()

```
Point3d CoordConvertor::cartesian2Spherical (
            const Point2d & p )  [static]
```

Convert a spherical coordinate (ϕ, θ) to cartesian coordinate.

Attention

Sperical2cartesian? The function name doesm't match

Parameters

p	(ϕ, θ)

Returns

The correspoding coordinate in cartesian

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

- /mnt/renderer/Zero/src/CoreLayer/Geometry/CoordConvertor.h
- /mnt/renderer/Zero/src/CoreLayer/Geometry/CoordConvertor.cpp

8.4 PixelSampler Class Reference

Base class for pixel sampler, which generate specific samples before rendering each pixel need the dimensions and samplesPerPixels when construct.

```
#include <Sampler.h>
```

Inherits Sampler.

Inherited by StratifiedSampler.

8.4.1 Detailed Description

Base class for pixel sampler, which generate specific samples before rendering each pixel need the dimensions and samplesPerPixels when construct.

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

- /mnt/renderer/Zero/src/FunctionLayer/Sampler/Sampler.h

8.5 RandomNumberGenerator Class Reference

RandomNumberGenerator, using pcg to generate random number.

```
#include <Random.h>
```

Public Member Functions

- double **operator()** ()

  Generate uniformly distributed random double in [0, 1)
- int **operator()** (int begin, int end)

  Generate uniformly distributed random int in [begin, end)

8.5.1 Detailed Description

RandomNumberGenerator, using pcg to generate random number.

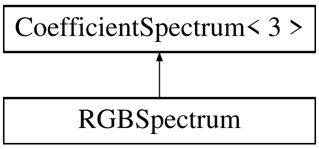
The documentation for this class was generated from the following file:

- /mnt/renderer/Zero/src/CoreLayer/Adapter/Random.h

8.6 RGBSpectrum Class Reference

```
#include <Color.h>
```

Inheritance diagram for RGBSpectrum:



Additional Inherited Members

8.6.1 Detailed Description

Todo To be finished

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

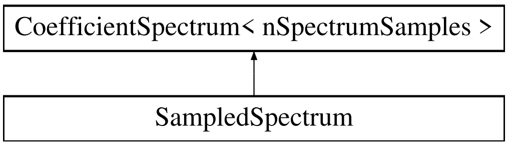
- /mnt/renderer/Zero/src/CoreLayer/ColorSpace/Color.h

8.7 SampledSpectrum Class Reference

The specturm samples uniformly. Actually used in program.

```
#include <Color.h>
```

Inheritance diagram for SampledSpectrum:



Public Member Functions

- virtual XYZ3 toXYZ3 () const override

Static Public Member Functions

- static void **init** ()

  Global init of static values. should be called before any constructor of SampledSpectrum.
- static SampledSpectrum **fromSampled** (std::vector< SpectrumSample > v)

  generate SampledSpectrum from a set of SpectrumSample.

8.7.1 Detailed Description

The specturm samples uniformly. Actually used in program.

8.7.2 Member Function Documentation

8.7.2.1 toXYZ3()

```
XYZ3 SampledSpectrum::toXYZ3 ( ) const  [override], [virtual]
```

Attention

> This function is just used for debugging

Reimplemented from CoefficientSpectrum< nSpectrumSamples >.

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

- /mnt/renderer/Zero/src/CoreLayer/ColorSpace/Color.h
- /mnt/renderer/Zero/src/CoreLayer/ColorSpace/Spectrum.cpp

8.8 SpectrumSample Struct Reference

One sample point from a spectrum.

`#include <Color.h>`

Public Member Functions

- bool **operator>** (const SpectrumSample &s) const

  Sorted by lambda.
- bool **operator<** (const SpectrumSample &s) const

  Sorted by lambda.

8.8.1 Detailed Description

One sample point from a spectrum.

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

- /mnt/renderer/Zero/src/CoreLayer/ColorSpace/Color.h

Chapter 9

File Documentation

9.1 Attribute.h

```

#pragma once
#include "rapidjson/document.h"
#include <vector>
#include <string>
#include <unordered_map>
#include <iostream>
#include <memory>

struct Value {
    Value() = default;
    virtual ~Value() = default;

    virtual void print() const = 0;
};

class Attribute {
public:
    Attribute() : m_type(EValueType::EInvalid), m_value(nullptr) { }

    Attribute(const rapidjson::Document &document);

    Attribute(const std::string &key, bool b);

    Attribute(const std::string &key, int i);

    Attribute(const std::string &key, double d);

    Attribute(const std::string &key, const std::string &str);

    Attribute(const std::string &key,
              const rapidjson::GenericValue<rapidjson::UTF8<»::ConstArray &array);

    Attribute(const std::string &key,
              const rapidjson::GenericValue<rapidjson::UTF8<»::ConstObject &object);


    void print() const {
        std::cout « "Key = " « m_key « ", ";
        m_value->print();
    }

    Attribute operator[](const std::string &key) const;

    Attribute operator[](int i) const;

    bool getBool() const;

    int getInt() const;

    double getDouble() const;

    std::string getString() const;

private:

    enum class EValueType {
        EInvalid = 0,
        EBool,
        EInt,
        EDouble,
        EString,
        EArray,
        EObject
    } m_type;

    std::string m_key;

    std::shared_ptr<Value> m_value;

};


struct BoolValue : public Value {
    BoolValue() = default;

    BoolValue(bool b) : data(b) { };

    virtual ~BoolValue() = default;

    virtual void print() const override {
        std::cout « "BoolValue : " « (data ? "true" : "false") « std::endl;
    }

    bool data;

};

struct IntValue : public Value {
    IntValue() = default;

    IntValue(int i) : data(i) { }

    virtual ~IntValue() = default;

    int data;

    virtual void print() const override {
        std::cout « "IntValue : " « data « std::endl;
    }
};


struct DoubleValue : public Value {
    DoubleValue() = default;

    DoubleValue(double d) : data(d) { }

    virtual ~DoubleValue() = default;

    double data;

    virtual void print() const override {
        std::cout « "DoubleValue : " « data « std::endl;
    }
};


struct ArrayValue : public Value {
    ArrayValue() = default;

    virtual ~ArrayValue() {
        for (int i = 0; i < data.size(); ++i) {
            delete data[i];
        }
    }

    std::vector<Attribute *> data;

    virtual void print() const override {
        std::cout « "ArrayValue : \n";
        for (int i = 0; i < data.size(); ++i) {
            std::cout « "\t";
            data[i]->print();
        }
    }
};

struct StringValue : public Value {
    StringValue() = default;

    StringValue(const std::string &str) : data(str) { }

    virtual ~StringValue() = default;

    virtual void print() const override {
        std::cout « "StringValue : " « data « std::endl;
    }

    std::string data;

};

struct ObjectValue : public Value {
    ObjectValue() = default;

    virtual ~ObjectValue() {
        for (auto itr = data.begin(); itr != data.end(); ++itr) {
            delete itr->second;
        }
    }

    virtual void print() const override {
        std::cout « "ObjectValue : {\n";
        for (auto itr = data.begin(); itr!=data.end(); ++itr) {
            std::cout « "\t";
            itr->second->print();
        }
        std::cout «"}"«std::endl;
    }

    std::unordered_map<std::string, Attribute *> data;
};


inline Attribute::Attribute(const rapidjson::Document &document) {
    m_type = EValueType::EObject;
    m_key = "root";
    m_value = std::make_shared<ObjectValue>();

    ObjectValue* object_value = static_cast<ObjectValue *>(m_value.get());

    for (auto itr = document.MemberBegin(); itr != document.MemberEnd(); ++itr) {
        Attribute *attribute = nullptr;
        const std::string &member_key = itr->name.GetString();
        const auto &member_value = itr->value;
        if (member_value.IsBool()) {
            attribute = new Attribute(member_key, member_value.GetBool());
        } else if (member_value.IsInt()) {
            attribute = new Attribute(member_key, member_value.GetInt());
        } else if (member_value.IsDouble()) {
            attribute = new Attribute(member_key, member_value.GetDouble());
        } else if (member_value.IsString()) {
            attribute = new Attribute(member_key, std::string(member_value.GetString()));
        } else if (member_value.IsArray()) {
            attribute = new Attribute(member_key, member_value.GetArray());
        } else if (member_value.IsObject()) {
            attribute = new Attribute(member_key, member_value.GetObject());
        }
        if (attribute == nullptr){
            std::cout « "Error type\n";
            std::exit(1);
        }
        object_value->data[member_key] = attribute;
    }
}

inline Attribute::Attribute(const std::string &key, bool b) :
    m_type(EValueType::EBool),
    m_key(key),
    m_value(new BoolValue {b}) { }


inline Attribute::Attribute(const std::string &key, int i) :
    m_type(EValueType::EInt),
    m_key(key),
    m_value(new IntValue {i}) { }

inline Attribute::Attribute(const std::string &key, double d) :
    m_type(EValueType::EDouble),
    m_key(key),
    m_value(new DoubleValue {d}) { }

inline Attribute::Attribute(const std::string &key, const std::string &str) :
    m_type(EValueType::EString),
    m_key(key),
    m_value(new StringValue {str}) { }


inline Attribute::Attribute(
    const std::string &key,
    const rapidjson::GenericValue<rapidjson::UTF8<»::ConstArray &array) :
```

```
    m_type(EValueType::EArray),
    m_key(key) {

    m_value = std::make_shared<ArrayValue>();
    ArrayValue *array_value = static_cast<ArrayValue *>(m_value.get());
    for (auto itr = array.Begin(); itr != array.End(); ++itr) {
        Attribute *attribute = nullptr;
        if (itr->IsBool()) {
            attribute = new Attribute("", itr->GetBool());
        } else if (itr->IsInt()) {
            attribute = new Attribute("", itr->GetInt());
        } else if (itr->IsDouble()) {
            attribute = new Attribute("", itr->GetDouble());
        } else if (itr->IsString()) {
            attribute = new Attribute("", std::string(itr->GetString()));
        } else if (itr->IsArray()) {
            attribute = new Attribute("", itr->GetArray());
        } else if (itr->IsObject()) {
            attribute = new Attribute("", itr->GetObject());
        }
        if (attribute == nullptr) {
            std::cout « "Error Type\n";
            std::exit(1);
        }
        array_value->data.emplace_back(attribute);
    }
}

inline Attribute::Attribute(
    const std::string &key,
    const rapidjson::GenericValue<rapidjson::UTF8<»::ConstObject &object) :
    m_type(EValueType::EObject),
    m_key(key) {

    m_value = std::make_shared<ObjectValue>();
    ObjectValue *object_value = static_cast<ObjectValue *>(m_value.get());
    for (auto itr = object.MemberBegin(); itr != object.MemberEnd(); ++itr) {
        Attribute *attribute = nullptr;
        const std::string &member_key = itr->name.GetString();
        const auto &member_value = itr->value;
        if (member_value.IsBool()) {
            attribute = new Attribute(member_key, member_value.GetBool());
        } else if (member_value.IsInt()) {
            attribute = new Attribute(member_key, member_value.GetInt());
        } else if (member_value.IsDouble()) {
            attribute = new Attribute(member_key, member_value.GetDouble());
        } else if (member_value.IsString()) {
            attribute = new Attribute(member_key, std::string(member_value.GetString()));
        } else if (member_value.IsArray()) {
            attribute = new Attribute(member_key, member_value.GetArray());
        } else if (member_value.IsObject()) {
            attribute = new Attribute(member_key, member_value.GetObject());
        }
        if (attribute == nullptr){
            std::cout « "Error type\n";
            std::exit(1);
        }
        object_value->data[member_key] = attribute;
    }
}

inline Attribute Attribute::operator[](const std::string &key) const {
    assert(m_type == EValueType::EObject);
    ObjectValue *object_value = static_cast<ObjectValue *>(m_value.get());
    const auto attributes = object_value->data;
    if (auto itr = attributes.find(key); itr == attributes.end()) {
        std::cout « "No such a key \"" « key «  "\"\n";
        std::exit(1);
    } else {
        return *(itr->second);
    }
}

inline Attribute Attribute::operator[](int i) const {
    assert(m_type == EValueType::EArray);
    ArrayValue *array_value = static_cast<ArrayValue *>(m_value.get());
    const auto attribute_list = array_value->data;
    assert(i < attribute_list.size());
    return *attribute_list[i];
}

inline bool Attribute::getBool() const {
    assert(m_type == EValueType::EBool);
    BoolValue *bool_value = static_cast<BoolValue *>(m_value.get());
    return bool_value->data;
}

```

```
inline int Attribute::getInt() const {
    assert(m_type == EValueType::EInt);
    IntValue *int_value = static_cast<IntValue *>(m_value.get());
    return int_value->data;
}

inline double Attribute::getDouble() const {
    assert(m_type == EValueType::EDouble);
    DoubleValue *double_value = static_cast<DoubleValue *>(m_value.get());
    return double_value->data;
}

inline std::string Attribute::getString() const {
    assert(m_type == EValueType::EString);
    StringValue *string_value = static_cast<StringValue *>(m_value.get());
    return string_value->data;
}

```

9.2 /mnt/renderer/Zero/src/CoreLayer/Adapter/Random.h File Reference

The class used to generate random numbers.

```
#include <random>
#include "pcg/pcg_random.hpp"
#include "CoreLayer/Math/Common.h"
```

Classes

- class RandomNumberGenerator

  RandomNumberGenerator, using pcg to generate random number.

9.2.1 Detailed Description

The class used to generate random numbers.

Author

Chenxi Zhou

Version

0.1

Date

2022-06-24

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.3 Random.h

Go to the documentation of this file.

```

#pragma once
#include <random>
#include "pcg/pcg_random.hpp"
#include "CoreLayer/Math/Common.h"

class RandomNumberGenerator {
    std::uniform_real_distribution<> dist;
    pcg_extras::seed_seq_from<std::random_device> seedSource;
    pcg32 rng;
public:
    RandomNumberGenerator():dist(0, ONEMINUSEPSILON), rng(seedSource) { }

    double operator()() {
        return dist(rng);
    }

    int operator()(int begin, int end) {
        int sample = begin + (end - begin) * dist(rng);
        return std::min(end - 1, sample);
    }

};
```

9.4 /mnt/renderer/Zero/src/CoreLayer/ColorSpace/Color.h File Reference

Color representation, including rgb, xyz and spectrum.

```
#include <vector>
#include <cmath>
#include <cfloat>
```

Classes

- class CoefficientSpectrum< nSamples >

  Spectrum of nSamples sample points.
- struct SpectrumSample

  One sample point from a spectrum.
- class SampledSpectrum

  The specturm samples uniformly. Actually used in program.
- class RGBSpectrum

Enumerations

- enum class SpectrumType

  types of spectrum. different strategies will be applied.

Functions

- double **mathClamp** (double source, double low=0.0, double high=DBL_MAX)

  Mathematical clamp.
- double **mathLerp** (double ratio, double source0, double source1)

  Mathematical lerp.

Variables

- static const int **nSpectrumSamples** = 60

  The number of uniform samples for SampledSpectrum.

9.4.1 Detailed Description

Color representation, including rgb, xyz and spectrum.

XYZ3 implemention.

Spectrum implemention and preset Spectrums.

RGB3 implemention.

Author

orbitchen

Version

0.1

Date

2022-04-30

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.5 Color.h

Go to the documentation of this file.

```

#pragma once

#include <vector>
#include <cmath>
#include <cfloat>

class RGB3;
class XYZ3;
class SampledSpectrum;

// TODO: should be defined by cmake marco.
using Spectrum = SampledSpectrum;

//using Spectrum = RGB3;

static const double sampledLambdaStart = 400.0;
static const double sampledLambdaEnd = 700.0;

static const int nSpectrumSamples = 60;

double mathClamp(double source, double low=0.0, double high=DBL_MAX);

double mathLerp(double ratio, double source0, double source1);

enum class SpectrumType { REFLECTANCE, ILLUMINANT };

```

```
class RGB3
{
    double rgbData[3];

public:
    RGB3();

    RGB3(double r, double g, double b);

    // @brief initialize all rgb value as val.
    RGB3(double val);

    double operator[](int i) const;
    double &operator[](int i);

    RGB3 operator+(const RGB3 &rgb);
    RGB3 operator-(const RGB3 &rgb);
    RGB3 operator*(const RGB3 &rgb);
    RGB3 operator/(const RGB3 &rgb);

    RGB3 &operator+=(const RGB3 &rgb);
    RGB3 &operator-=(const RGB3 &rgb);
    RGB3 &operator*=(const RGB3 &rgb);
    RGB3 &operator/=(const RGB3 &rgb);

    RGB3 operator*(double v);
    RGB3 operator/(double v);

    RGB3 &operator*=(double v);
    RGB3 &operator/=(double v);

    friend RGB3 operator*(double v, const RGB3 &rgb);

    XYZ3 toXYZ3() const;

    // @brief convert RGB3 to SampledSpectrum.
    Spectrum toSpectrum(SpectrumType type=SpectrumType::REFLECTANCE) const;
};

class XYZ3
{
    double xyzData[3];

public:
    XYZ3();

    XYZ3(double x, double y, double z);

    // @brief initialize all xyz value as val.
    XYZ3(double val);

    double operator[](int i) const;
    double &operator[](int i);

    XYZ3 operator+(const XYZ3 &xyz);
    XYZ3 operator-(const XYZ3 &xyz);
    XYZ3 operator*(const XYZ3 &xyz);
    XYZ3 operator/(const XYZ3 &xyz);

    XYZ3 &operator+=(const XYZ3 &xyz);
    XYZ3 &operator-=(const XYZ3 &xyz);
    XYZ3 &operator*=(const XYZ3 &xyz);
    XYZ3 &operator/=(const XYZ3 &xyz);

    XYZ3 operator*(double v);
    XYZ3 operator/(double v);

    XYZ3 &operator*=(double v);
    XYZ3 &operator/=(double v);

    friend XYZ3 operator*(double v, const XYZ3 &xyz);

    RGB3 toRGB3() const;
};

template <int nSamples>
class CoefficientSpectrum
{
protected:
    double coefficients[nSamples];

public:
    CoefficientSpectrum() {
        for (int i = 0; i < nSamples; i++) {
            coefficients[i] = 0.0;
        }
    }

    CoefficientSpectrum(double val) {
        for (int i = 0; i < nSamples; i++) {
            coefficients[i] = val;
        }
    }

    double operator[](int i) const {
        return coefficients[i];
    }

    double &operator[](int i) {
        return coefficients[i];
    }

    CoefficientSpectrum operator+(const CoefficientSpectrum&s) const {
        CoefficientSpectrum retVal = *this;
        for (int i = 0; i < nSamples; i++) {
            retVal.coefficients[i] += s.coefficients[i];
        }
        return retVal;
    }

    CoefficientSpectrum operator-(const CoefficientSpectrum&s) const {
        CoefficientSpectrum retVal = *this;
        for (int i = 0; i < nSamples; i++) {
            retVal.coefficients[i] -= s.coefficients[i];
        }
        return retVal;
    }

    CoefficientSpectrum operator*(const CoefficientSpectrum&s) const {
        CoefficientSpectrum retVal = *this;
        for (int i = 0; i < nSamples; i++) {
            retVal.coefficients[i] *= s.coefficients[i];
        }
        return retVal;
    }

    CoefficientSpectrum operator/(const CoefficientSpectrum&s) const {
        CoefficientSpectrum retVal = *this;
        for (int i = 0; i < nSamples; i++) {
            retVal.coefficients[i] /= s.coefficients[i];    // NaN
        }
        return retVal;
    }

    CoefficientSpectrum& operator+=(const CoefficientSpectrum&s) {
        for (int i = 0; i < nSamples; i++) {
            this->coefficients[i] += s.coefficients[i];
        }
        return *this;
    }

    CoefficientSpectrum& operator-=(const CoefficientSpectrum&s) {
        for (int i = 0; i < nSamples; i++) {
            this->coefficients[i] -= s.coefficients[i];
        }
        return *this;
    }

    CoefficientSpectrum& operator*=(const CoefficientSpectrum&s) {
        for (int i = 0; i < nSamples; i++) {
            this->coefficients[i] *= s.coefficients[i];
        }
        return *this;
    }

    CoefficientSpectrum& operator/=(const CoefficientSpectrum&s) {
        for (int i = 0; i < nSamples; i++) {
            this->coefficients[i] /= s.coefficients[i]; // NaN
        }
        return *this;
    }

    CoefficientSpectrum operator*(double v) const {
        CoefficientSpectrum retVal = *this;
        for (int i = 0; i < nSamples; i++) {
            retVal.coefficients[i] *= v;
        }
        return retVal;
    }

    CoefficientSpectrum operator/(double v) const {
        CoefficientSpectrum retVal = *this;
        for (int i = 0; i < nSamples; i++) {
            retVal.coefficients[i] /= v;    // NaN
        }
        return retVal;
    }

    CoefficientSpectrum& operator*=(double v) {
        for (int i = 0; i < nSamples; i++) {
            this->coefficients[i] *= v;
        }
        return *this;
    }

    CoefficientSpectrum& operator/=(double v) {
        for (int i = 0; i < nSamples; i++) {
            this->coefficients[i] /= v; // NaN
        }
        return *this;
    }

    friend CoefficientSpectrum operator*(double v, const CoefficientSpectrum& s) {
        return s * v;
    }

    friend CoefficientSpectrum sqrt(const CoefficientSpectrum& s) {
        CoefficientSpectrum ret;
        for (int i = 0; i < nSamples; i++)
            ret[i] = std::sqrt(s[i]);
        return ret;
    }

    friend CoefficientSpectrum pow(const CoefficientSpectrum&s, double e) {
        CoefficientSpectrum ret;
        for (int i = 0; i < nSamples; i++)
            ret[i] = std::pow(s[i], e);
        return ret;
    }

    friend CoefficientSpectrum exp(const CoefficientSpectrum&s) {
        CoefficientSpectrum ret;
        for (int i = 0; i < nSamples; i++)
            ret[i] = std::exp(s[i]);
        return ret;
    }

    bool isBlack() const {
        for (int i = 0; i < nSamples; i++) {
            if (coefficients[i] != 0.0)
                return false;
        }
    }

    bool hasNaN() const {
        for (int i = 0; i <nSamples; i++)
            if (std::isnan(coefficients[i]))
                return true;
        return false;
    }

    inline CoefficientSpectrum clamp(double low = 0.0, double high = DBL_MAX) const {
        CoefficientSpectrum retVal;
        for (int i = 0; i < nSamples; i++) {
            retVal[i] = mathClamp(coefficients[i], low, high);
        }
        return retVal;
    }

    double sum() const {
        double sum = 0;
        for (int i = 0; i < nSamples; i++) {
            sum += coefficients[i];
        }
        return sum;
    }

    double average() const {
        return sum() / nSamples;
    }

    virtual XYZ3 toXYZ3() const {
        //DEBUG this function should never be called.
        return XYZ3(0.0);
    }
};

struct SpectrumSample
{
    double lambda;
    double value;

    SpectrumSample(double _lambda, double _value) {
        lambda = _lambda;
        value = _value;
    }

    bool operator>(const SpectrumSample& s) const {
        return lambda > s.lambda;
    }

    bool operator<(const SpectrumSample& s) const {
        return lambda < s.lambda;
    }
};

class SampledSpectrum
    : public CoefficientSpectrum<nSpectrumSamples>
{
    // these spectrums should be calculated at compile time.
    static SampledSpectrum X;
    static SampledSpectrum Y;
    static SampledSpectrum Z;

    static SampledSpectrum rgbRefl2SpectWhite;
    static SampledSpectrum rgbRefl2SpectCyan;
    static SampledSpectrum rgbRefl2SpectMagenta;
    static SampledSpectrum rgbRefl2SpectYellow;
    static SampledSpectrum rgbRefl2SpectRed;
    static SampledSpectrum rgbRefl2SpectGreen;
    static SampledSpectrum rgbRefl2SpectBlue;

    static SampledSpectrum rgbIllum2SpectWhite;
    static SampledSpectrum rgbIllum2SpectCyan;
    static SampledSpectrum rgbIllum2SpectMagenta;
    static SampledSpectrum rgbIllum2SpectYellow;
    static SampledSpectrum rgbIllum2SpectRed;
    static SampledSpectrum rgbIllum2SpectGreen;
    static SampledSpectrum rgbIllum2SpectBlue;

public:
    friend class RGB3;
    friend class XYZ3;

    static void init();

    SampledSpectrum();

    SampledSpectrum(double val);

    SampledSpectrum(const CoefficientSpectrum& s);

    static SampledSpectrum fromSampled(std::vector<SpectrumSample> v);

    virtual XYZ3 toXYZ3() const override;
};

class RGBSpectrum : public CoefficientSpectrum<3>
{
    // TODO RGBSpectrum
};
```

9.6 /mnt/renderer/Zero/src/CoreLayer/Geometry/BoundingBox.cpp File Reference

Bounding box impl.

```
#include "BoundingBox.h"
```

9.6.1 Detailed Description

Bounding box impl.

Author

orbitchen

Version

0.1

Date

2022-05-07

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.7 /mnt/renderer/Zero/src/CoreLayer/Geometry/BoundingBox.h File Reference

Bounding box declaration.

```
#include "Geometry.h"
#include "CoreLayer/Ray/Ray.h"
#include <optional>
#include <algorithm>
#include <limits>
```

Classes

- class BoundingBox3< BaseType >

  Axis-aligned bounding box 3D base type, the box edges are mutually perpendicular and aligned to axes.

Functions

- template<typename BaseType >
  static BoundingBox3< BaseType > BoundingBoxUnion (const BoundingBox3< BaseType > &b1, const BoundingBox3< BaseType > &b2)

  Union of two bounding boxes.
- template<typename BaseType >
  static BoundingBox3< BaseType > BoundingBoxUnionIntersect (const BoundingBox3< BaseType > &b1, const BoundingBox3< BaseType > &b2)

  Overlap of two bounding boxes.

9.7.1 Detailed Description

Bounding box declaration.

Author

orbitchen

Version

0.1

Date

2022-04-30

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.7.2 Function Documentation

9.7.2.1 BoundingBoxUnion()

```
template<typename BaseType >
static BoundingBox3< BaseType > BoundingBoxUnion (
            const BoundingBox3< BaseType > & b1,
            const BoundingBox3< BaseType > & b2 )  [static]
```

Union of two bounding boxes.

Returns

The smallest bounding box that bounds both b1 and b2.

9.7.2.2 BoundingBoxUnionIntersect()

```
template<typename BaseType >
static BoundingBox3< BaseType > BoundingBoxUnionIntersect (
            const BoundingBox3< BaseType > & b1,
            const BoundingBox3< BaseType > & b2 )  [static]
```

Overlap of two bounding boxes.

Returns

The largest bounding box that bounded by both b1 and b2.

Todo The function name "BoundingBoxOverlap" might better

9.8 BoundingBox.h

Go to the documentation of this file.

```

#pragma once

#include "Geometry.h"
#include "CoreLayer/Ray/Ray.h"
#include <optional>
#include <algorithm>
#include <limits>

template <typename BaseType>
class BoundingBox3
{
public:
    TPoint3<BaseType> pMin;
    TPoint3<BaseType> pMax;

    BoundingBox3() {
        pMin[0] = pMin[1] = pMin[2] = std::numeric_limits<BaseType>::max();
        pMax[0] = pMax[1] = pMax[2] = std::numeric_limits<BaseType>::min();
    };

    BoundingBox3(TPoint3<BaseType> _p)
    {
        pMin = pMax = _p;
    }
    BoundingBox3(TPoint3<BaseType> _pMin, TPoint3<BaseType> _pMax)
    {
        pMin = _pMin;
        pMax = _pMax;
    }

    std::optional<Point2d> Intersection(const Ray& r) {
        double t0 = 0, t1 = DBL_MAX;
        for (int i = 0; i < 3; i++) {
            double inv = 1.0 / r.direction[i];
            double tNear = (pMin[i] - r.origin[i]) * inv;
            double tFar = (pMax[i] - r.origin[i]) * inv;
            if (tNear > tFar) std::swap(tNear, tFar);
            if (t0 < tNear) t0 = tNear;
            if (t1 > tFar) t1 = tFar;
        }
        if (t0 > t1) return std::nullopt;
        return std::make_optional(Point2d(t0, t1));
    }

    double SurfaceArea() {
        double x = pMax[0] - pMin[0];
        double y = pMax[1] - pMin[1];
        double z = pMax[2] - pMin[2];
        return 2 * (x * y + x * z + y * z);
    }
};

template <typename BaseType>
static BoundingBox3<BaseType> BoundingBoxUnion(const BoundingBox3<BaseType>& b1,
                                               const BoundingBox3<BaseType>& b2)
{
    TPoint3<BaseType> _pMin(std::min(b1.pMin[0], b2.pMin[0]), std::min(b1.pMin[1], b2.pMin[1]),
    std::min(b1.pMin[2], b2.pMin[2]));
    TPoint3<BaseType> _pMax(std::max(b1.pMax[0], b2.pMax[0]), std::max(b1.pMax[1], b2.pMax[1]),
    std::max(b1.pMax[2], b2.pMax[2]));
    return BoundingBox3(_pMin, _pMax);
}
template <typename BaseType>
static BoundingBox3<BaseType> BoundingBoxUnionIntersect(const BoundingBox3<BaseType>& b1,
                                                        const BoundingBox3<BaseType>& b2)
{
    TPoint3<BaseType> _pMin(std::max(b1.pMin[0], b2.pMin[0]), std::max(b1.pMin[1], b2.pMin[1]),
    std::max(b1.pMin[2], b2.pMin[2]));
    TPoint3<BaseType> _pMax(std::min(b1.pMax[0], b2.pMax[0]), std::min(b1.pMax[1], b2.pMax[1]),
    std::min(b1.pMax[2], b2.pMax[2]));
    return BoundingBox3(_pMin, _pMax);
}

using BoundingBox3f = BoundingBox3<double>;
```

9.9 /mnt/renderer/Zero/src/CoreLayer/Geometry/CoordConvertor.cpp File Reference

Provide convertor between standard coordinates.

```
#include "CoordConvertor.h"
#include <cmath>
```

9.9.1 Detailed Description

Provide convertor between standard coordinates.

Author

Zhimin Fan

Version

0.1

Date

2022-05-27

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.10 /mnt/renderer/Zero/src/CoreLayer/Geometry/CoordConvertor.h File Reference

Provide convertor between standard coordinates.

```
#include "Geometry.h"
```

Classes

- class CoordConvertor

  Provides transformations between different coordinate systems.

9.10.1 Detailed Description

Provide convertor between standard coordinates.

Author

Zhimin Fan

Version

0.1

Date

2022-05-27

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.11 CoordConvertor.h

Go to the documentation of this file.

```
#include "Geometry.h"

class CoordConvertor
{
public:
    static Point3d cartesian2Spherical(const Point2d &p);

    static Vec3d cartesian2SphericalVec(const Point2d &p);

    static Vec3d world2TBN(const Vec3d &v, const Vec3d &t, const Vec3d &b, const Vec3d &n);

    static Vec3d TBN2World(const Vec3d &v, const Vec3d &t, const Vec3d &b, const Vec3d &n);
};
```

9.12 /mnt/renderer/Zero/src/CoreLayer/Geometry/Frame.h File Reference

```
#include "Geometry.h"
#include "CoreLayer/Math/Common.h"
```

9.12.1 Detailed Description

Author

Junping Yuan

Version

0.1

Date

2022/06/06

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.13 Frame.h

Go to the documentation of this file.

```
//
// Created by yjp on 2022/6/6.
//

#pragma once

#include "Geometry.h"
#include "CoreLayer/Math/Common.h"


static void coordinateSystem(const Normal3d &a, Vec3d  &b, Vec3d  &c) {
    if (std::abs(a.x) > std::abs(a.y)) {
        float invLen = 1.0f / std::sqrt(a.x * a.x + a.z * a.z);
        c = Vec3d (a.z * invLen, 0.0f, -a.x * invLen);
    } else {
        float invLen = 1.0f / std::sqrt(a.y * a.y + a.z * a.z);
        c = Vec3d(0.0f, a.z * invLen, -a.y * invLen);
    }
    Vec3d  _a(a.x,a.y,a.z);
    b = cross(_a,c);
}

struct Frame {
    Vec3d s, t;
    Normal3d  n;

    Frame() { }

    Frame(const Vec3d  &s, const Vec3d &t, const Normal3d  &n)
            : s(s), t(t), n(n) { }

    Frame(const Vec3d &x, const Vec3d &y, const Vec3d &z)
            : s(x), t(y), n(z.x,z.y,z.z) { }

    Frame(const Normal3d n) : n(n){
        coordinateSystem(n, s, t);
    }

    Vec3d toLocal(const Vec3d &v) const {
        return Vec3d(
                dot(v,s), dot(v,t), dot(v,n)
        );
    }

    Vec3d toWorld(const Vec3d &v) const {
        return s * v.x + t * v.y   + n * v.z ;
    }

    static float cosTheta(const Vec3d &v) {
        return v.z;
    }

    static float sinTheta(const Vec3d &v) {
        float temp = sinTheta2(v);
        if (temp <= 0.0f)
            return 0.0f;
        return std::sqrt(temp);
    }

    static float tanTheta(const Vec3d &v) {
        float temp = 1 - v.z*v.z;
        if (temp <= 0.0f)
            return 0.0f;
        return std::sqrt(temp) / v.z;
    }

    static float sinTheta2(const Vec3d &v) {
        return 1.0f - v.z * v.z;
    }

    static float sinPhi(const Vec3d &v) {
        float sinTheta = Frame::sinTheta(v);
        if (sinTheta == 0.0f)
            return 1.0f;
        return clamp(v.y / sinTheta, -1.0f, 1.0f);
    }

    static float cosPhi(const Vec3d &v) {
        float sinTheta = Frame::sinTheta(v);
        if (sinTheta == 0.0f)
            return 1.0f;
        return clamp(v.x / sinTheta, -1.0f, 1.0f);
    }

    static float sinPhi2(const Vec3d &v) {
        return clamp(v.y * v.y / sinTheta2(v), 0.0f, 1.0f);
    }

    static float cosPhi2(const Vec3d &v) {
        return clamp(v.x * v.x / sinTheta2(v), 0.0f, 1.0f);
    }

    bool operator==(const Frame &frame) const {
        return frame.s == s && frame.t == t && frame.n == n;
    }

    bool operator!=(const Frame &frame) const {
        return !operator==(frame);
    }


};
```

9.14 /mnt/renderer/Zero/src/CoreLayer/Geometry/Geometry.h File Reference

The geometry part of whole program, including vector, point and normal.

```
#include "normal.h"
#include "point.h"
#include "vector.h"
```

9.14.1 Detailed Description

The geometry part of whole program, including vector, point and normal.

Author

zcx

Version

0.1

Date

2022-04-22

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.15 Geometry.h

Go to the documentation of this file.

```

#pragma once

#include "normal.h"
#include "point.h"
#include "vector.h"


// vector.h
template <typename T>
struct TVector2;
template <typename T>
struct TVector3;
using Vec2f = TVector2<float>;
using Vec2d = TVector2<double>;
using Vec2i = TVector2<int>;
using Vec3f = TVector3<float>;
using Vec3d = TVector3<double>;
using Vec3i = TVector3<int>;
// point.h
template <typename T>
struct TPoint2;
template <typename T>
struct TPoint3;
using Point2f = TPoint2<float>;
using Point2d = TPoint2<double>;
using Point2i = TPoint2<int>;
using Point3f = TPoint3<float>;
using Point3d = TPoint3<double>;
using Point3i = TPoint3<int>;
// normal.h
struct Normal3d;
```

9.16 /mnt/renderer/Zero/src/CoreLayer/Geometry/Matrix.cpp File Reference

Matrix impl.

```
#include "Matrix.h"
```

9.16.1 Detailed Description

Matrix impl.

Author

orbitchen

Version

0.1

Date

2022-05-10

Copyright

9.17 /mnt/renderer/Zero/src/CoreLayer/Geometry/Matrix.h File Reference

Matrix4x4 (Eigen backend) and TransformMatrix3D.

```
#include "Geometry.h"
#include "Eigen/Dense"
```

9.17.1 Detailed Description

Matrix4x4 (Eigen backend) and TransformMatrix3D.

Author

orbitchen

Version

0.2

Date

2022-05-10

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.18 Matrix.h

Go to the documentation of this file.

```

#pragma once

#include "Geometry.h"

#include "Eigen/Dense"

 /*
  * @brief Angle type. Providing convenient angle transformation between deg and rad.
  */
class Angle
{
    double deg;
    double rad;

public:
    enum class AngleType
    {
        ANGLE_DEG,
        ANGLE_RAD
    };
    /*
     * @brief init an angle.
     * @param v value of angle, deg or rad.
     * @param type the type of v.
     */
    Angle(double v, AngleType type);

    double getDeg() const;

    double getRad() const;
};

enum class EulerType
{
    EULER_XYZ,
    EULER_ZYX
};

// @brief a simple wrap of Eigen Matrix 4x4.
class Matrix4x4
{
private:

    // @brief Eigen data. inaccessible from outside.
    Eigen::Matrix4d matrix=Eigen::Matrix4d::Identity();

    // @brief init from Eigen data. inaccessible from outside.
    Matrix4x4(const Eigen::Matrix4d& _matrix);

public:

    Matrix4x4();

    Matrix4x4 operator*(const Matrix4x4& mat) const;

    Vec3d operator*(const Vec3d& v) const;
    Point3d operator*(const Point3d& p) const;
    Normal3d operator*(const Normal3d& n) const;

    // model
    static Matrix4x4 translate(double x, double y, double z);
    static Matrix4x4 scale(double x, double y, double z);
    static Matrix4x4 scale(double ratio);
    static Matrix4x4 rotateEuler(const Angle& x, const Angle& y, const Angle& z, EulerType type = EulerType::EULER_XYZ);
    static Matrix4x4 rotateQuaternion(double w, double x, double y, double z);
    static Matrix4x4 rotateAxis(const Vec3d& axis, const Angle& angle);

    // view & projection
    static Matrix4x4 lookAt(const Point3d& lookFrom, const Vec3d& vecLookAt, const Vec3d& up);
    static Matrix4x4 orthographic(double left,double right,double up,double down,double near,double far);
    static Matrix4x4 perspective(const Angle& fov,double aspect, double near, double far);

    Matrix4x4 inverse();
    Matrix4x4 transpose();

    friend void printMatrix(const Matrix4x4 &mat);
};

/*
 * @brief Encapsulated transform matrix. By default, it will be initialized as identity matrix.
 */
class TransformMatrix3D
{
    // @brief the matrix that applies rotate, scale and translate.
    Matrix4x4 matrixAll;

    Matrix4x4 matrixRotate;
    Matrix4x4 matrixScale;
    Matrix4x4 matrixTranslate;

    // @brief true iff matrixAll!=matrixTranslate*matrixScale*matrixRotate.
    bool dirty;

    void update();

public:
    TransformMatrix3D();

    void setTranslate(double x, double y, double z);

    void setScale(double x, double y, double z);
    void setScale(double ratio);

    void setRotateEuler(const Angle& x, const Angle& y, const Angle& z, EulerType type = EulerType::EULER_XYZ);

    void setRotateQuaternion(double w, double x, double y, double z);

    // @brief Rotate by axis. Counterclockwise rotate.
    void setRotateAxis(const Vec3d& axis, const Angle& angle);

    Vec3d operator*(const Vec3d &v);
    Point3d operator*(const Point3d &p);
    Normal3d operator*(const Normal3d &n);
};
```

```

// TODO TransformMatrix2D
```

9.19 /mnt/renderer/Zero/src/CoreLayer/Geometry/normal.h File Reference

```
#include "vector.h"
```

9.19.1 Detailed Description

Author

zcx

Version

0.1

Date

2022-04-22

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.20 normal.h

Go to the documentation of this file.

```

#pragma once
#include "vector.h"

struct Normal3d : public TVector3<double> {

    Normal3d() { }

    Normal3d (const TVector3<double> &v) : TVector3<double>(normalize(v)) { }

    Normal3d (double _x, double _y, double _z) : Normal3d(TVector3<double> {_x, _y, _z})  { }


};
```

9.21 /mnt/renderer/Zero/src/CoreLayer/Geometry/point.h File Reference

```
#include "vector.h"
```

9.21.1 Detailed Description

Author

zcx

Version

0.1

Date

2022-04-22

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.22 point.h

Go to the documentation of this file.

```

#pragma once
#include "vector.h"

template <typename T>
struct TPoint2 {
    /*--- data field ---*/
    T x, y;

    /*--- constructor ---*/
    TPoint2() { }

    TPoint2(T _x, T _y) : x(_x), y(_y) { }

    explicit TPoint2(T t) : x(t), y(t) { }

    /*--- operator overloading ---*/
    TPoint2 operator+(const TVector2<T> &rhs) const {
        return TPoint2(x+rhs.x, y+rhs.y);
    }

    TPoint2 operator+(const TPoint2<T>& rhs) const {
        return TPoint2(x + rhs.x, y + rhs.y);
    }

    TPoint2& operator+=(const TVector2<T> &rhs) {
        x += rhs.x, y += rhs.y;
        return *this;
    }

    TPoint2 operator-(const TVector2<T> &rhs) const {
        return TPoint2(x-rhs.x, y-rhs.y);
    }

    TVector2<T> operator-(const TPoint2<T>& rhs) const {
        return TVector2(x - rhs.x, y - rhs.y);
    }

    TPoint2& operator-=(const TVector2<T> &rhs) {
        x -= rhs.x, y -= rhs.y;
        return *this;
    }

    TPoint2 operator*(T t) const {
        return TPoint2(x*t, y*t);
    }

    TPoint2& operator*=(T t) {
        x *= t, y *= t;
        return *this;
    }

    TPoint2 operator/(T t) const {
        assert(t != 0);
        T recip = (T) 1 / t;
        return TPoint2(x*recip, y*recip);
    }

    TPoint2& operator/=(T t) {
        assert(t != 0);
        T recip = (T) 1 / t;
        x *= recip, y *= recip;
        return *this;
    }

    TPoint2 operator-() const {
        return TPoint2(-x, -y);
    }

    bool operator==(const TPoint2 &rhs) const {
        return x==rhs.x && y==rhs.y;
    }

    bool operator!=(const TPoint2 &rhs) const {
        return x!=rhs.x || y!=rhs.y;
    }

    T operator[] (int i) const {
        return (&x)[i];
    }

    T& operator[] (int i) {
        return (&x)[i];
    }

    bool isZero() const {
        return x==0 && y==0;
    }
};

template <typename T>
std::ostream& operator«(std::ostream &os, const TPoint2<T> &p) {
    os « p.x « " " « p.y;
    return os;
}

template <typename T>
TPoint2<T> operator*(T t, const TPoint2<T> &v) {
    return TPoint2(t*v.x, t*v.y);
}

template<>
inline TPoint2<int> TPoint2<int>::operator/(int i) const {
    assert(i!=0);
    return TPoint2<int>(x/i, y/i);
}

template<>
inline TPoint2<int>& TPoint2<int>::operator/=(int i) {
    assert(i!=0);
    x/=i, y/=i;
    return *this;
}


template <typename T>
struct TPoint3 {
    /*--- data field ---*/
    T x, y, z;

    /*--- constructor ---*/
    TPoint3() { }

    TPoint3(T _x, T _y, T _z) : x(_x), y(_y), z(_z) { }

    explicit TPoint3(T t) : x(t), y(t), z(t) { }

    /*--- operator overloading ---*/
    TPoint3 operator+(const TVector3<T> &rhs) const {
        return TPoint3(x+rhs.x, y+rhs.y, z+rhs.z);
    }

    TPoint3 operator+(const TPoint3<T>& rhs) const {
        return TPoint3(x + rhs.x, y + rhs.y, z + rhs.z);
    }

    TPoint3& operator+=(const TVector3<T> &rhs) {
        x += rhs.x, y += rhs.y, z += rhs.z;
        return *this;
    }

    TPoint3 operator-(const TVector3<T> &rhs) const {
        return TPoint3(x-rhs.x, y-rhs.y, z-rhs.z);
    }

    TVector3<T> operator-(const TPoint3<T> &rhs) const {
        return TVector3<T>(x-rhs.x, y-rhs.y, z-rhs.z);
    }

    TPoint3& operator-=(const TVector3<T> &rhs) {
        x -= rhs.x, y -= rhs.y, z -= rhs.z;
        return *this;
    }

    TPoint3 operator*(T t) const {
        return TPoint3(x*t, y*t, z*t);
    }

    TPoint3& operator*=(T t) {
        x *= t, y *= t, z *= t;
        return *this;
    }

    TPoint3 operator/(T t) const {
        assert(t != 0);
        T recip = (T) 1 / t;
        return TPoint3(x*recip, y*recip, z*recip);
    }

    TPoint3& operator/=(T t) {
        assert(t != 0);
        T recip = (T) 1 / t;
        x *= recip, y *= recip, z *= recip;
        return *this;
    }

    TPoint3 operator-() const {
        return TPoint3(-x, -y, -z);
    }

    bool operator==(const TPoint3 &rhs) const {
        return x==rhs.x && y==rhs.y && z==rhs.z;
    }

    bool operator!=(const TPoint3 &rhs) const {
        return x!=rhs.x || y!=rhs.y || z!=rhs.z;
    }

    T operator[] (int i) const {
        return (&x)[i];
    }

    T& operator[] (int i) {
        return (&x)[i];
    }

    bool isZero() const {
        return x==0 && y==0 && z==0;
    }
};

template <typename T>
std::ostream& operator«(std::ostream &os, const TPoint3<T> &p) {
    os « p.x « " " « p.y « " " « p.z;
    return os;
}

template <typename T>
TPoint3<T> operator*(T t, const TPoint3<T> &v) {
    return TPoint3(t*v.x, t*v.y, t*v.z);
}

template<>
inline TPoint3<int> TPoint3<int>::operator/(int i) const {
    assert(i!=0);
    return TPoint3<int>(x/i, y/i, z/i);
}

template<>
inline TPoint3<int>& TPoint3<int>::operator/=(int i) {
    assert(i!=0);
    x/=i, y/=i, z/i;
    return *this;
```

```
234 }
```

9.23 /mnt/renderer/Zero/src/CoreLayer/Geometry/Transform3d.cpp File Reference

3d transformation representation impl.

```
#include "Transform3d.h"
```

9.23.1 Detailed Description

3d transformation representation impl.

Author

Zhimin Fan

Version

0.1

Date

2022-05-20

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.24 /mnt/renderer/Zero/src/CoreLayer/Geometry/Transform3d.h File Reference

3d transformation representation. More like an interface: be inherited means that the inheriting class supports 3d transformation.

```
#include "Geometry.h"
#include "Matrix.h"
#include <memory>
```

9.24.1 Detailed Description

3d transformation representation. More like an interface: be inherited means that the inheriting class supports 3d transformation.

Author

orbitchen

Version

0.1

Date

2022-04-30

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.25 Transform3d.h

Go to the documentation of this file.

```
#pragma once

#include "Geometry.h"
#include "Matrix.h"

#include <memory>

class Transform3D
{

    std::shared_ptr<TransformMatrix3D> matrix;

    bool isDone = false;

protected:
    // apply matrix to this object. Cache should be managed locally.
    virtual void apply() = 0;

public:
    Transform3D();

    Transform3D(std::shared_ptr<Transform3D> _matrix);

    void setTranslate(double x, double y, double z);

    void setScale(double x, double y, double z);
    void setScale(double ratio);

    void setRotateEuler(Angle x, Angle y, Angle z, EulerType type = EulerType::EULER_XYZ);

    void setRotateQuaternion(double w, double x, double y, double z);

    // @brief Rotate by axis. Counterclockwise rotate.
    void setRotateAxis(Angle angle, Vec3d axis);

    // @brief inform this object that transform setting is DONE and 'you' can apply all transformation without redundant calculation. apply() should be called within.
    void done();

    Point3d getTranslate();
};
```

9.26 /mnt/renderer/Zero/src/CoreLayer/Geometry/vector.h File Reference

Template class for vector.

```
#include <assert.h>
#include <cmath>
#include <iostream>
```

Functions

- template<typename T >
  TVector3< T > cross (const TVector3< T > &v1, const TVector3< T > &v2)

  Only 3 deminsion vector can do this operation.

9.26.1 Detailed Description

Template class for vector.

Author

zcx

Version

0.1

Date

2022-04-22

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.26.2 Function Documentation

9.26.2.1 cross()

```
template<typename T >
TVector3< T > cross (
            const TVector3< T > & v1,
            const TVector3< T > & v2 )
```

Only 3 deminsion vector can do this operation.

Template Parameters

T	

Parameters

v1	
v2	

Returns

TVector3<T>

9.27 vector.h

Go to the documentation of this file.

```

#pragma once
#include <assert.h>
#include <cmath>
#include <iostream>

template <typename T>
struct TVector2 {

    /*---data field---*/
    T x, y;

    /*----constructor---*/
    TVector2() { }

    TVector2(T _x, T _y) : x(_x), y(_y) { }

    explicit TVector2(T t) : x(t), y(t) { }

    /*---operator overloading---*/
    TVector2 operator+(const TVector2 &rhs) const {
        return TVector2(x + rhs.x, y + rhs.y);
    }

    TVector2& operator+=(const TVector2 &rhs) {
        x += rhs.x, y += rhs.y;
        return *this;
    }

    TVector2 operator-(const TVector2 &rhs) const {
        return TVector2(x - rhs.x, y - rhs.y);
    }

    TVector2& operator-=(const TVector2 &rhs) {
        x -= rhs.x, y-= rhs.y;
        return *this;
    }

    TVector2 operator*(const T t) const {
        return TVector2(x * t, y * t);
    }

    TVector2& operator*=(const T t) {
        x *= t, y *= t;
        return *this;
    }

    TVector2 operator/(T t) const {
        assert(t != 0);
        T recip = (T) 1 / t;
        return TVector2(x * recip, y * recip);
    }

    TVector2& operator/=(T t) {
        assert(t != 0);
        T recip = (T) 1/t;
        x *= recip, y *= recip;
        return *this;
    }

    TVector2 operator-() const {
        return TVector2(-x, -y);
    }

    T operator[](int i) const {
        return (&x)[i];
    }

    T& operator[](int i) {
        return (&x)[i];
    }

    decltype(auto) length2() const {
        return x * x + y * y;
    }

    decltype(auto) length() const {
        return std::sqrt(x*x + y*y);
    }

    bool operator==(const TVector2 &rhs) const {
        return x==rhs.x && y==rhs.y;
    }

    bool operator!=(const TVector2 &rhs) const {
        return x!=rhs.x || y!=rhs.y;
    }

    bool isZero() const {
        return x==0 && y==0;
    }

};

template <typename T>
std::ostream& operator«(std::ostream& os, const TVector2<T> &v) {
    os « v.x « " " « v.y ;
    return os;
}

template <typename T>
TVector2<T> operator*(T t, const TVector2<T> &v) {
    return TVector2<T>(v.x * t, v.y * t);
}

template <typename T>
decltype(auto) dot(const TVector2<T> &v1, const TVector2<T> &v2) {
    return v1.x*v2.x + v1.y * v2.y;
}

template <typename T>
decltype(auto) absDot(const TVector2<T> &v1, const TVector2<T> &v2) {
    return std::abs(v1.x*v2.x + v1.y * v2.y);
}

template <typename T>
TVector2<double> normalize(const TVector2<T> &v) {
    double recip = 1.0f / v.length();
    return TVector2<double> (v.x * recip, v.y * recip);
}

template<>
inline TVector2<int> TVector2<int>::operator/(int i) const {
    assert(i != 0);
    return TVector2<int>(x/i, y/i);
}

template<>
inline TVector2<int>& TVector2<int>::operator/=(int i) {
    assert(i != 0);
    x/=i, y/=i;
    return *this;
}



template <typename T>
struct TVector3 {
    /*--- data field ---*/
    T x, y, z;

    /*--- constructor ---*/
    TVector3 () { }

    TVector3 (T _x, T _y, T _z) : x(_x), y(_y), z(_z) { }

    explicit TVector3 (T t) : x(t), y(t), z(t) { }

    /*--- operator overloading ---*/
    TVector3 operator+(const TVector3 &rhs) const {
        return TVector3(x + rhs.x, y + rhs.y, z + rhs.z);
    }

    TVector3& operator+=(const TVector3 &rhs) {
        x += rhs.x, y += rhs.y, z += rhs.z;
        return *this;
    }

    TVector3 operator-(const TVector3 &rhs) const {
        return TVector3(x-rhs.x, y-rhs.y, z-rhs.z);
    }

    TVector3& operator-=(const TVector3 &rhs) {
        x -= rhs.x, y -= rhs.y, z -= rhs.z;
        return *this;
    }

    TVector3 operator*(const T t) const {
        return TVector3(x*t, y*t, z*t);
    }

    TVector3& operator*=(const T t) {
        x*=t, y*=t, z*=t;
        return *this;
    }

    TVector3 operator/(const T t) const {
        assert(t != 0);
        T recip = (T) 1/t;
        return TVector3(x*recip, y*recip, z*recip);
    }

    TVector3& operator/=(const T t) {
        assert(t != 0);
        T recip = (T) 1/t;
        x*=recip, y*=recip, z*=recip;
        return *this;
    }

    TVector3 operator-() const {
        return TVector3(-x, -y, -z);
    }

    T operator[](const int i) const {
        return (&x)[i];
    }

    T& operator[](const int i) {
        return (&x)[i];
    }

    decltype(auto) length2() const {
        return x*x + y*y + z*z;
    }

    decltype(auto) length() const {
        return std::sqrt(x*x + y*y + z*z);
    }

    bool operator==(const TVector3 &rhs) const {
        return x==rhs.x && y==rhs.y && z==rhs.z;
    }

    bool operator!=(const TVector3 &rhs) const {
        return x!=rhs.x || y!=rhs.y || z!=rhs.z;
    }

    bool isZero() const {
        return x==0 && y==0 && z==0;
    }
};

template <typename T>
std::ostream& operator«(std::ostream &os, const TVector3<T> &v) {
    os « v.x « " " « v.y « " " « v.z;
    return os;
}

template <typename T>
TVector3<T> operator*(T t, const TVector3<T> &v) {
    return TVector3<T>(v.x*t, v.y*t, v.z*t);
}

template <typename T>
decltype(auto) dot(const TVector3<T> &v1, const TVector3<T> &v2) {
    return v1.x * v2.x + v1.y * v2.y + v1.z * v2.z;
}

template <typename T>
decltype(auto) absDot(const TVector3<T> &v1, const TVector3<T> &v2) {
    return std::abs(v1.x * v2.x + v1.y * v2.y + v1.z * v2.z);
}

template <typename T>
TVector3<double> normalize(const TVector3<T> &v) {
    double recip = 1.0f / v.length();
    return TVector3<double> (v.x * recip, v.y * recip, v.z * recip);
}

template <typename T>
TVector3<T> cross(const TVector3<T> &v1, const TVector3<T> &v2) {
    return TVector3<T> (
        v1.y * v2.z - v1.z * v2.y,
        v1.z * v2.x - v1.x * v2.z,
        v1.x * v2.y - v1.y * v2.x
    );
}
```

9.28 /mnt/renderer/Zero/src/CoreLayer/Math/Common.h File Reference

some common but useful functions

```
#include <cmath>
#include <limits>
#include <utility>
```

Functions

- float **clamp** (float value, float min, float max)

  Simple floating point clamping function.
- int **clamp** (int value, int min, int max)

  Simple integer clamping function.
- float **lerp** (float t, float v1, float v2)

  Linearly interpolate between two values.
- int **mod** (int a, int b)

  Always-positive modulo operation.

9.28.1 Detailed Description

some common but useful functions

Author

Junping Yuan

Version

0.1

Date

2022/06/06

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.29 Common.h

Go to the documentation of this file.

```

#pragma  once
#include <cmath>
#include <limits>
#include <utility>
// some constant values
#define M_PI         3.14159265358979323846f
#define INV_PI       0.31830988618379067154f
#define INV_TWOPI    0.15915494309189533577f

constexpr double ONEMINUSEPSILON = 1 - std::numeric_limits<double>::epsilon();

inline float clamp(float value, float min, float max) {
    if (value < min)
        return min;
    else if (value > max)
        return max;
    else return value;
}

inline int clamp(int value, int min, int max) {
    if (value < min)
        return min;
    else if (value > max)
        return max;
    else return value;
}

inline float lerp(float t, float v1, float v2) {
    return ((float) 1 - t) * v1 + t * v2;
}

inline int mod(int a, int b) {
    int r = a % b;
    return (r < 0) ? r+b : r;
}

inline  float fresnel(float cosThetaI, float extIOR, float intIOR) {
    float etaI = extIOR, etaT = intIOR;

    if (extIOR == intIOR)
        return 0.0f;

    /* Swap the indices of refraction if the interaction starts
       at the inside of the object */
    if (cosThetaI < 0.0f) {
        std::swap(etaI, etaT);
        cosThetaI = -cosThetaI;
    }

    /* Using Snell’s law, calculate the squared sine of the
       angle between the normal and the transmitted ray */
    float eta = etaI / etaT,
            sinThetaTSqr = eta*eta * (1-cosThetaI*cosThetaI);

    if (sinThetaTSqr > 1.0f)
        return 1.0f;  /* Total internal reflection! */

    float cosThetaT = std::sqrt(1.0f - sinThetaTSqr);

    float Rs = (etaI * cosThetaI - etaT * cosThetaT)
               / (etaI * cosThetaI + etaT * cosThetaT);
    float Rp = (etaT * cosThetaI - etaI * cosThetaT)
               / (etaT * cosThetaI + etaI * cosThetaT);

    return (Rs * Rs + Rp * Rp) / 2.0f;
}

inline  double fresnel(double cosThetaI, double extIOR, double intIOR) {
    double etaI = extIOR, etaT = intIOR;

    if (extIOR == intIOR)
        return 0.0;

    /* Swap the indices of refraction if the interaction starts
       at the inside of the object */
    if (cosThetaI < 0.0f) {
        std::swap(etaI, etaT);
        cosThetaI = -cosThetaI;
    }

    /* Using Snell's law, calculate the squared sine of the
       angle between the normal and the transmitted ray */
    double eta = etaI / etaT,
            sinThetaTSqr = eta*eta * (1-cosThetaI*cosThetaI);

    if (sinThetaTSqr > 1.0f)
        return 1.0f;  /* Total internal reflection! */

    double cosThetaT = std::sqrt(1.0f - sinThetaTSqr);

    double Rs = (etaI * cosThetaI - etaT * cosThetaT)
               / (etaI * cosThetaI + etaT * cosThetaT);
    double Rp = (etaT * cosThetaI - etaI * cosThetaT)
               / (etaT * cosThetaI + etaI * cosThetaT);

    return (Rs * Rs + Rp * Rp) / 2.0;
}
```

9.30 Warp.h

```
#include "CoreLayer/Geometry/Geometry.h"
#include "Common.h"

static  double TentInverse(double x){
    if(x<=.5f)
        return std::sqrt(2*x)-1;
    return  1- std::sqrt(2-2*x);
}
inline   Point2d SquareToTent(const Point2d &sample) {
    Point2d  res(TentInverse(sample[0]), TentInverse(sample[1]));
    return  res;
//    throw NoriException("SquareToTent() is not yet implemented!");
}

inline  float SquareToTentPdf(const Point2d &p) {
    return (1.0-abs(p[0])) * (1.0-abs(p[1]));
}


//Point2d SquareToUniformDisk(const Point2d &sample) {
//    auto phi=2*sample.x()*M_PI;
//    auto r=sqrt(sample.y());
//    return {r*cos(phi),r*sin(phi)};
//}
//
//float SquareToUniformDiskPdf(const Point2d &p) {
//    return  < 1.f ? INV_PI : .0f;}

inline  Vec3d SquareToUniformSphere(const Point2d &sample) {
    float z = 1 - 2 * sample[0];
    float r = std::sqrt(std::max((float )0, (float)1 - z * z));
    float phi = 2 * M_PI * sample[1];
    return {r * std::cos(phi), r * std::sin(phi), z};
}

```

```
inline  float SquareToUniformSpherePdf(const Vec3d &v) {
    return 0.25f*INV_PI;
}

inline  Vec3d SquareToUniformHemisphere(const Point2d &sample) {
    float z = 1 - 2 * sample[0];
    float r = std::sqrt(std::max((float )0, (float)1 - z * z));
    float phi = 2 * M_PI * sample[1];
    return {r * std::cos(phi), r * std::sin(phi), abs(z)};
}

inline float SquareToUniformHemispherePdf(const Vec3d &v) {
    return v[2] >=0 ? 0.5f * INV_PI : .0f;
}

inline  Vec3d SquareToCosineHemisphere(const Point2d &sample) {
    float z=sqrt(1-sample.x);
    float phi=sample.y*2*M_PI;

    return {sqrt(sample.x)* cos(phi),sqrt(sample.x)*sin(phi),z};
}

inline  float SquareToCosineHemispherePdf(const Vec3d &v) {
    return v[2] >=0 ? v.z * INV_PI : .0f;
}
//
inline  Vec3d SquareToBeckmann(const Point2d &sample,double alpha) {
    auto tan2theta= -alpha*alpha*log( sample.x );
    auto cosTheta=sqrt(1/(1+tan2theta));
    auto sinTheta= sqrt(1-cosTheta*cosTheta);
    auto phi=sample.y * 2 * M_PI;
    Vec3d t1= Vec3d(sinTheta*cos(phi), sinTheta*sin(phi),cosTheta);
    return t1;
}
//
inline  float SquareToBeckmannPdf(const Vec3d &m, double alpha) {
    if(m.z<=0)
        return 0.0f;
    auto cosTheta=m.z;
    auto sinTheta=sqrt(1-cosTheta*cosTheta);
    auto tan2Theta=(sinTheta* sinTheta)/(cosTheta*cosTheta);
    float azimuthal =   INV_PI;
    float longitudinal = exp(-tan2Theta/(alpha*alpha))  / (alpha*alpha*pow(cosTheta,3));
    return azimuthal * longitudinal;
}
```

9.31 /mnt/renderer/Zero/src/CoreLayer/Ray/Ray.cpp File Reference

Ray implemention.

```
#include "Ray.h"
```

9.31.1 Detailed Description

Ray implemention.

Author

Zhimin Fan

Version

0.1

Date

2022-05-09

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.32 /mnt/renderer/Zero/src/CoreLayer/Ray/Ray.h File Reference

Ray representation.

```
#include <memory>
#include "CoreLayer/Geometry/Geometry.h"
#include "FunctionLayer/Medium/Medium.h"
#include <cfloat>
```

9.32.1 Detailed Description

Ray representation.

Author

orbitchen

Version

0.1

Date

2022-04-30

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.33 Ray.h

Go to the documentation of this file.

```

#pragma once

//#include "../core.h"
#include <memory>
#include "CoreLayer/Geometry/Geometry.h"
#include "FunctionLayer/Medium/Medium.h"
#include <cfloat>
struct Ray
{
    Point3d origin;
    Vec3d direction;

    double timeMin;
    double timeMax;

    std::shared_ptr<Medium> medium;

    Point3d at(double t);

    bool withinTime(double time);

    Ray(const Point3d &_origin, const Vec3d &_direction, double _timeMin = .0f, double _timeMax =
    DBL_MAX);
};

struct RayDifferential : public Ray
{
    Point3d origin_x, origin_y;
    Vec3d direction_x, direction_y;
};
```

9.34 /mnt/renderer/Zero/src/CoreLayer/Scene/Scene.cpp File Reference

Scene implemention.

```
#include "Scene.h"
```

9.34.1 Detailed Description

Scene implemention.

Author

Zhimin Fan

Version

0.1

Date

2022-05-31

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.35 /mnt/renderer/Zero/src/CoreLayer/Scene/Scene.h File Reference

Scene representation. Handle ray intersection.

```
#include "CoreLayer/Ray/Ray.h"
#include "FunctionLayer/Shape/Entity.h"
#include "FunctionLayer/Aggregate/Bvh.h"
#include "FunctionLayer/Intersection.h"
#include "FunctionLayer/Light/Light.h"
#include <optional>
```

9.35.1 Detailed Description

Scene representation. Handle ray intersection.

Author

orbitchen

Version

0.1

Date

2022-04-30

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.36 Scene.h

Go to the documentation of this file.

```

#pragma once

#include "CoreLayer/Ray/Ray.h"
#include "FunctionLayer/Shape/Entity.h"
#include "FunctionLayer/Aggregate/Bvh.h"
#include "FunctionLayer/Intersection.h"
#include "FunctionLayer/Light/Light.h"

#include <optional>

class Scene
{
    std::shared_ptr<Bvh> BVH;
    std::shared_ptr<std::vector<std::shared_ptr<Light»> lights;
    std::shared_ptr<std::vector<std::shared_ptr<Entity»> entities;

public:
    Scene();
    void addEntity(std::shared_ptr<Entity> object);
    void addLight(std::shared_ptr<Light> light);

    void build();
    std::optional<Intersection> intersect(const Ray &r) const;

    // @return true if r hits object first (closest), false otherwise.
    bool intersectionTest(const Ray &r, std::shared_ptr<Entity> object) const;

    std::shared_ptr<std::vector<std::shared_ptr<Light»> getLights() const;
};
```

9.37 /mnt/renderer/Zero/src/FunctionLayer/Aggregate/Bvh.h File Reference

BVH implementation.

```
#include "CoreLayer/Geometry/BoundingBox.h"
#include "FunctionLayer/Shape/Entity.h"
#include "FunctionLayer/Intersection.h"
```

9.37.1 Detailed Description

BVH implementation.

BVH declaration.

Author

Pengpei Hong

Version

0.1

Date

2022-06-30

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.38 Bvh.h

Go to the documentation of this file.

```

#pragma once

#include "CoreLayer/Geometry/BoundingBox.h"
#include "FunctionLayer/Shape/Entity.h"
#include "FunctionLayer/Intersection.h"

//@brief Entity information declaration for building BVH
struct EntityInfo {
    EntityInfo(){}
    EntityInfo(int _EntityId, const BoundingBox3f& _bounds): EntityId(_EntityId), bounds(_bounds),
    center(0.5 * (_bounds.pMin + _bounds.pMax)){}
    int EntityId;
    BoundingBox3f bounds;
    Point3d center;
};

struct BvhTreeNode {
    BoundingBox3f bounds;
    std::shared_ptr<BvhTreeNode> children[2] = {nullptr, nullptr};//0: left, 1: right
    int splitAxis;
    int nEntites = 0;//0: interior nodes, otherwise: leaf nodes
    int entityOffset;
};

//@brief Bvh Nodes in Dfs-Order
struct LinearBvhNode {
    BoundingBox3f bounds;
    union
    {
        int firstdEntityOffset;//for leaves to enumerate
        int secondChildOrder;//for interior nodes to traverse
    };
    int nEntites = 0;
    int splitAxis;
};

struct Bvh {
public:
    enum class SplitMethod{
        SAH,
        Middle,
        EqualCounts
    };//not support LBVH yet
    const SplitMethod splitMethod;
    std::vector<std::shared_ptr<Entity» entites;
    std::vector<LinearBvhNode> linearBvhNodes;

    /*
     * @brief Bvh constructor
     * @param <_entites>
     * @param <_SplitMeshod>
     */
    Bvh(std::vector<std::shared_ptr<Entity»& _entites, SplitMethod _splitMethod = SplitMethod::SAH);

    /*
    * @brief recursively build BVH
    * @return the root of BVH
    */
    std::shared_ptr<BvhTreeNode> RecursiveBuild(std::vector<EntityInfo>& entityInfo, int start, int end,
    int& nodeNumber, std::vector<std::shared_ptr<Entity»& orderedEntites);

    //@brief flatten the BVH to Dfs-Order
    void Flatten(std::shared_ptr<BvhTreeNode> node, int& dfsOrder);

    //@brief return the scene intersection
    std::optional<Intersection> Intersect(const Ray& r);
};
```

9.39 /mnt/renderer/Zero/src/FunctionLayer/Camera/Camera.h File Reference

Different cameras. Need to be investigated.

```
#include "CoreLayer/Geometry/Geometry.h"
#include "CoreLayer/Ray/Ray.h"
```

9.39.1 Detailed Description

Different cameras. Need to be investigated.

Author

orbitchen

Version

0.1

Date

2022-04-30

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.40 Camera.h

Go to the documentation of this file.

```

#pragma once

#include "CoreLayer/Geometry/Geometry.h"
#include "CoreLayer/Ray/Ray.h"

struct CameraSample {
    Point2d xy;
    double time;
    Point2d lens;
};

class Camera
{
public:

    virtual Ray generateRay(const Point2i &filmResolution,
                            const Point2i &pixelPosition,
                            const CameraSample &sample) const = 0;
};
```

9.41 /mnt/renderer/Zero/src/FunctionLayer/Camera/Perspective.h File Reference

Abstract base class for all perspective camera.

```
#include "CoreLayer/Geometry/Matrix.h"
#include "Camera.h"
```

9.41.1 Detailed Description

Abstract base class for all perspective camera.

Author

Chenxi Zhou

Version

0.1

Date

2022-06-10

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.42 Perspective.h

Go to the documentation of this file.

```
#pragma once
#include "CoreLayer/Geometry/Matrix.h"
#include "Camera.h"

class PerspectiveCamera : public Camera {
protected:
public:
    // TODO replace martrix with transform3d
    //
    //     [0, 0]---------------> x
    //          |      |
    //          |------+ [x, y]
    //          |
    //          y
    Matrix4x4 cameraToWorld, sampleToFilm;
public:
    PerspectiveCamera() = default;

    PerspectiveCamera(
        const Point3d &lookFrom,
        const Point3d &lookAt,
        const Vec3d &up,
        double xFov,
        double aspectRatio,
        double distToFilm
    )
    {
        cameraToWorld =
            Matrix4x4::lookAt(lookFrom, lookAt - lookFrom, up).inverse();
        // ! near is the distToFilm and far set to MAX_FLOAT
        // ! if far set to MAX_DOUBLE, it will crash when computing the matrix
        Matrix4x4 filmToSample = Matrix4x4::perspective(
            Angle(xFov, Angle::AngleType::ANGLE_DEG),
            aspectRatio,
            distToFilm,
            std::numeric_limits<float>::max()
        );
        sampleToFilm =
            Matrix4x4::scale(0.5, -0.5, 1.0)
            * Matrix4x4::translate(1, -1, 0)
            * filmToSample;
        sampleToFilm = sampleToFilm.inverse();
    }
};
```

9.43 Pinhole.h

```
#pragma once

#include "Perspective.h"

class PinholeCamera : public PerspectiveCamera {
public:
    PinholeCamera() = default;
    PinholeCamera(
        const Point3d &lookFrom,
        const Point3d &lookAt,
        const Vec3d &up,
        double xFov,
        double aspectRatio,
        double distToFilm
    ) : PerspectiveCamera(lookFrom, lookAt, up, xFov, aspectRatio, distToFilm) { }

    virtual Ray generateRay(const Point2i &filmResolution,
                            const Point2i &pixelPosition,
                            const CameraSample &sample) const override;
};
```

9.44 Thinlens.h

```
#pragma once

#include "Perspective.h"

class ThinlensCamera : public PerspectiveCamera {
protected:
    double apertureRadius, focalLen, focalDistance;
public:
    ThinlensCamera() = default;
    ThinlensCamera(
        const Point3d &lookFrom,
        const Point3d &lookAt,
        const Vec3d &up,
        double xFov,
        double aspectRatio,
        double _focaDistance,
        double _apertureRadius = 0.1,
        double _focalLen = 0.05
    ):apertureRadius(_apertureRadius), focalLen(_focalLen), focalDistance(_focaDistance) {
        double distToFilm =
            (_focalLen * _focaDistance) / (_focaDistance - _focalLen);

        cameraToWorld =
            Matrix4x4::lookAt(lookFrom, lookAt - lookFrom, up).inverse();
        // ! near is the distToFilm and far set to MAX_FLOAT
        // ! if far set to MAX_DOUBLE, it will crash when computing the matrix
        Matrix4x4 filmToSample = Matrix4x4::perspective(
            Angle(xFov, Angle::AngleType::ANGLE_DEG),
            aspectRatio,
            distToFilm,
            std::numeric_limits<float>::max()
        );
        sampleToFilm =
            Matrix4x4::scale(0.5, -0.5, 1.0)
            * Matrix4x4::translate(1, -1, 0)
            * filmToSample;
        sampleToFilm = sampleToFilm.inverse();

    }


    virtual Ray generateRay(const Point2i &filmResolution ,
                            const Point2i &pixelPosition,
                            const CameraSample &sample) const override;
};
```

9.45 /mnt/renderer/Zero/src/FunctionLayer/Film/Film.cpp File Reference

The class that records ray sampling results. Just like old camera exposures!

```
#include <fstream>
#include "Film.h"
```

9.45.1 Detailed Description

The class that records ray sampling results. Just like old camera exposures!

Author

Zhimin Fan

Version

0.1

Date

2022-05-31

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.46 /mnt/renderer/Zero/src/FunctionLayer/Film/Film.h File Reference

The class that records ray sampling results. Just like old camera exposures!

```
#include "FunctionLayer/Filter/Filter.h"
#include "ResourceLayer/ResourceManager.h"
#include <memory>
```

9.46.1 Detailed Description

The class that records ray sampling results. Just like old camera exposures!

Author

orbitchen

Version

0.1

Date

2022-04-30

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.47 Film.h

Go to the documentation of this file.

```

#pragma once

#include "FunctionLayer/Filter/Filter.h"
#include "ResourceLayer/ResourceManager.h"
#include <memory>

class Film
{
protected:
    std::unique_ptr<Image> image;
    std::shared_ptr<Filter> filter;
    Point2i resolution;
    int channels;
    std::vector<double> sumWeights; // a temp impl
    std::vector<Spectrum> sumValues; // a temp impl

    void syncWithGui();

public:
    Film(Point2i resolution, int channels);
    Film(Point3i shape);

    Spectrum getSpectrum(const Point2i &p);
    RGB3 getRGB(const Point2i &p);

    void setGamma(double gamma = 2.2);
    void setFilter(std::shared_ptr<Filter> filter);
    Point2i getResolution() const;

    // @brief: 'deposit' a spectrum at p. all deposit will be averaged when get(p).
    void deposit(const Point2i &p, const Spectrum &s);
    void save(const std::string &path);

    int getDepositeCount(const Point2i &p);
};
```

9.48 /mnt/renderer/Zero/src/FunctionLayer/Filter/Filter.h File Reference

Linear filters.

```
#include "CoreLayer/Geometry/Geometry.h"
```

9.48.1 Detailed Description

Linear filters.

Author

Zhimin Fan

Version

0.1

Date

2022-05-01

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.49 Filter.h

Go to the documentation of this file.

```

#pragma once

#include "CoreLayer/Geometry/Geometry.h"

// ! Note that we cannot use any non-linear "filter" here, such as bilteral and etc

// @brief: Base class for linear filter used for film
class Filter
{
public:
    virtual float eval(const Point2d &pos) = 0;

    // @brief: also known as Kernel Size
    virtual Point2d support() = 0;
};

```

9.50 /mnt/renderer/Zero/src/FunctionLayer/Integrator/AbstractPath↩Integrator.cpp File Reference

Path Integrator Abstraction.

```
#include "AbstractPathIntegrator.h"
```

9.50.1 Detailed Description

Path Integrator Abstraction.

Author

Zhimin Fan

Version

0.1

Date

2022-05-16

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.51 /mnt/renderer/Zero/src/FunctionLayer/Integrator/AbstractPath↩Integrator.h File Reference

Path Integrator Abstraction.

```
#include <cmath>
#include "CoreLayer/Ray/Ray.h"
#include "MonteCarloIntegrator.h"
```

9.51.1 Detailed Description

Path Integrator Abstraction.

Author

Zhimin Fan

Version

0.1

Date

2022-05-16

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.52 AbstractPathIntegrator.h

Go to the documentation of this file.

```

#pragma once

#include <cmath>
#include "CoreLayer/Ray/Ray.h"
#include "MonteCarloIntegrator.h"

struct PathIntegratorLocalRecord
{
    Vec3d wi;
    Spectrum f;
    double pdf;
    bool isDelta = false;
};

class AbstractPathIntegrator : public MonteCarloIntegrator
{
protected:
    const int nDirectLightSamples = 1;
    const double misWeightPower = 1.0f;

public:
    AbstractPathIntegrator(
        std::shared_ptr<Camera> _camera,
        std::unique_ptr<Film> _film,
        std::unique_ptr<TileGenerator> _tileGenerator,
        std::shared_ptr<Sampler> _sampler,
        int _spp,
        int _renderThreadNum=4);

    virtual Spectrum Li(const Ray &ray, std::shared_ptr<Scene> scene);
    virtual double MISWeight(double x, double y);

    /*************************************************************
    Functions below need to be implemented in derived classes
    *************************************************************/

    // @brief Return the radiance along given ray, emitted from given intersection.
    // @param scene      Ptr to scene.
    // @param its        Intersection hit by ray.
    // @param ray        Ray to evaluate.
    // @return           Direction of given ray, incident radiance at origin of ray, pdf of direct light sampling.
    virtual PathIntegratorLocalRecord evalEmittance(std::shared_ptr<Scene> scene,
                                                    std::optional<Intersection> its,
                                                    const Ray &ray) = 0;

    // @brief Sample incident direction of direct lighting.
    // @param scene     Ptr to scene.
    // @param its       Reference point.
    // @param ray       Ray, used to specify wo (out direction).
    // @return          Sampled incident direction, incident radiance and pdf per solid angle.
    virtual PathIntegratorLocalRecord sampleDirectLighting(std::shared_ptr<Scene> scene,
                                                           const Intersection &its,
                                                           const Ray &ray) = 0;

    // @brief Return scatter value of BSDF or phase function.
    // @param scene     Ptr to scene.
    // @param its       Reference point.
    // @param ray       Ray, used to specify wo (out direction).
    // @param wi        Incident direction wi.
    // @return          Incident direction, scatter throughput f, pdf per solid angle.
    //                  For surface, f is the product of BSDF value and cosine term.
    //                  For medium, f is the value of phase function.
    virtual PathIntegratorLocalRecord evalScatter(std::shared_ptr<Scene> scene,
                                                  const Intersection &its,
                                                  const Ray &ray,
                                                  const Vec3d &wi) = 0;

    // @brief Sample incident direction by scatter value of BSDF or phase function.
    // @param scene     Ptr to scene.
    // @param its       Reference point.
    // @param ray       Ray, used to specify wo (out direction).
    // @return          Sampled incident direction, scatter throughput f, pdf per solid angle.
    //                  For surface, f is the product of BSDF value and cosine term.
    //                  For medium, f is the value of phase function.
    virtual PathIntegratorLocalRecord sampleScatter(std::shared_ptr<Scene> scene,
                                                    const Intersection &its,
                                                    const Ray &ray) = 0;

    // @brief Return probability of Russian roulette.
    virtual double russianRoulette(std::shared_ptr<Scene> scene,
                                   const Intersection &its,
                                   const Spectrum &T,
                                   int nBounce) = 0;
};
```

9.53 /mnt/renderer/Zero/src/FunctionLayer/Integrator/Integrator.cpp File Reference

Integrator.

```
#include "Integrator.h"
```

9.53.1 Detailed Description

Integrator.

Author

Zhimin Fan

Version

0.1

Date

2022-05-31

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.54 /mnt/renderer/Zero/src/FunctionLayer/Integrator/Integrator.h File Reference

Renderer. The start of everything.

```
#include "CoreLayer/Scene/Scene.h"
#include "FunctionLayer/Camera/Camera.h"
#include "FunctionLayer/Film/Film.h"
#include "FunctionLayer/Sampler/Sampler.h"
#include "FunctionLayer/TileGenerator/TileGenerator.h"
#include <memory>
```

9.54.1 Detailed Description

Renderer. The start of everything.

Author

orbitchen

Version

0.1

Date

2022-04-30

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.55 Integrator.h

Go to the documentation of this file.

```

#pragma once

#include "CoreLayer/Scene/Scene.h"
#include "FunctionLayer/Camera/Camera.h"
#include "FunctionLayer/Film/Film.h"
#include "FunctionLayer/Sampler/Sampler.h"
#include "FunctionLayer/TileGenerator/TileGenerator.h"

#include <memory>

class Integrator
{
protected:
    std::shared_ptr<Camera> camera;
    std::unique_ptr<Film> film;

public:
    Integrator(std::shared_ptr<Camera> _camera, std::unique_ptr<Film> _film);
    virtual void render(std::shared_ptr<Scene> scene) = 0;
    virtual void save(const std::string& path);

};
```

9.56 /mnt/renderer/Zero/src/FunctionLayer/Integrator/MonteCarlo↩ Integrator.h File Reference

Integrators.

```
#include <cmath>
#include "Integrator.h"
```

9.56.1 Detailed Description

Integrators.

Author

Zhimin Fan edited by orbitchen at 2022-7-7: apply multithread acceleration and tile generator.

Version

0.1

Date

2022-05-06

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.57 MonteCarloIntegrator.h

Go to the documentation of this file.

```

#pragma once

#include <cmath>
#include "Integrator.h"

class MonteCarloIntegrator : public Integrator
{
protected:
    std::shared_ptr<Sampler> sampler;
    int spp = 4;

    std::shared_ptr<TileGenerator> tileGenerator;

    // @brief: render process per thread. Should be called in render().
    void renderPerThread(std::shared_ptr<Scene> scene);

    int renderThreadNum=4;

public:
    MonteCarloIntegrator(
        std::shared_ptr<Camera> _camera,
        std::unique_ptr<Film> _film,
        std::unique_ptr<TileGenerator> _tileGenerator,
        std::shared_ptr<Sampler> _sampler,
        int _spp,
        int _renderThreadNum=4
        );

    virtual void render(std::shared_ptr<Scene> scene);

    // @brief Estimate radiance along a given ray
    virtual Spectrum Li(const Ray &ray, std::shared_ptr<Scene> scene) = 0;
    // @brief Get a random number WITHOUT using MonteCarloIntegrator::sampler
    virtual double randFloat();
};
```

9.58 /mnt/renderer/Zero/src/FunctionLayer/Integrator/PathIntegrator.cpp File Reference

Path Integrator.

```
#include "CoreLayer/Geometry/CoordConvertor.h"
#include "PathIntegrator.h"
```

9.58.1 Detailed Description

Path Integrator.

Author

Zhimin Fan

Version

0.1

Date

2022-05-06

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.59 /mnt/renderer/Zero/src/FunctionLayer/Integrator/PathIntegrator.h File Reference

Path Integrator.

```
#include <cmath>
#include "CoreLayer/Ray/Ray.h"
#include "AbstractPathIntegrator.h"
```

9.59.1 Detailed Description

Path Integrator.

Author

Zhimin Fan

Version

0.1

Date

2022-05-06

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.60 PathIntegrator.h

Go to the documentation of this file.

```

#pragma once

#include <cmath>
#include "CoreLayer/Ray/Ray.h"
#include "AbstractPathIntegrator.h"

class PathIntegrator : public AbstractPathIntegrator
{
protected:
    const int nPathLengthLimit = 20;
    const double pRussianRoulette = 0.95;

public:
    PathIntegrator(
        std::shared_ptr<Camera> _camera,
        std::unique_ptr<Film> _film,
        std::unique_ptr<TileGenerator> _tileGenerator,
        std::shared_ptr<Sampler> _sampler,
        int _spp,
        int _renderThreadNum=4);

    // @brief Return the radiance along given ray, emitted from given intersection.
    // @param scene     Ptr to scene.
    // @param its       Intersection hit by ray.
    // @param ray       Ray to evaluate.
    // @return          Direction of given ray, incident radiance at origin of ray, pdf of direct light sampling.
    virtual PathIntegratorLocalRecord evalEmittance(std::shared_ptr<Scene> scene,
                                                    std::optional<Intersection> its,
                                                    const Ray &ray) override;

    // @brief Sample incident direction of direct lighting.
    // @param scene     Ptr to scene.
    // @param its       Reference point.
    // @param ray       Ray, used to specify wo (out direction).
    // @return          Sampled incident direction, incident radiance and pdf per solid angle.
    virtual PathIntegratorLocalRecord sampleDirectLighting(std::shared_ptr<Scene> scene,
                                                           const Intersection &its,
                                                           const Ray &ray) override;

    // @brief Return scatter value of BSDF or phase function.
    // @param scene     Ptr to scene.
    // @param its       Reference point.
    // @param ray       Ray, used to specify wo (out direction).
    // @param wi        Incident direction wi.
    // @return          Incident direction, scatter throughput f, pdf per solid angle.
    //                  For surface, f is the product of BSDF value and cosine term.
    //                  For medium, f is the value of phase function.
    virtual PathIntegratorLocalRecord evalScatter(std::shared_ptr<Scene> scene,
                                                  const Intersection &its,
                                                  const Ray &ray,
                                                  const Vec3d &wi) override;

    // @brief Sample incident direction by scatter value of BSDF or phase function.
    // @param scene     Ptr to scene.
    // @param its       Reference point.
    // @param ray       Ray, used to specify wo (out direction).
    // @return          Sampled incident direction, scatter throughput f, pdf per solid angle.
    //                  For surface, f is the product of BSDF value and cosine term.
    //                  For medium, f is the value of phase function.
    virtual PathIntegratorLocalRecord sampleScatter(std::shared_ptr<Scene> scene,
                                                    const Intersection &its,
                                                    const Ray &ray);

    // @brief Return survive probability of Russian roulette.
    virtual double russianRoulette(std::shared_ptr<Scene> scene,
                                   const Intersection &its,
                                   const Spectrum &T,
                                   int nBounce) override;

    // todo: move light sampling into a new class (called LightDistribution?)
    // @brief Sample a light, by some weight distribution
    virtual std::pair<std::shared_ptr<Light>, double> chooseOneLight(std::shared_ptr<Scene> scene,
                                                                     const Intersection &its,
                                                                     const Ray &ray,
                                                                     double lightSample);

    // @brief Probability of choosing a specified light source
    virtual double chooseOneLightPdf(std::shared_ptr<Scene> scene,
                                     const Intersection &its,
                                     const Ray &ray,
                                     std::shared_ptr<Light> light);

    // @brief Evaluate radiance of env lights
    virtual PathIntegratorLocalRecord evalEnvLights(std::shared_ptr<Scene> scene,
                                                    const Ray &ray);
};
```

9.61 /mnt/renderer/Zero/src/FunctionLayer/Intersection.h File Reference

Intersection information on an object.

```
#include "CoreLayer/Geometry/Geometry.h"
#include "CoreLayer/Geometry/Frame.h"
#include "FunctionLayer/Material/Material.h"
#include "FunctionLayer/Medium/Medium.h"
#include "FunctionLayer/Shape/Entity.h"
#include <memory>
```

9.61.1 Detailed Description

Intersection information on an object.

Author

orbitchen

Version

0.1

Date

2022-04-30

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.62 Intersection.h

Go to the documentation of this file.

```

#pragma once

#include "CoreLayer/Geometry/Geometry.h"
#include "CoreLayer/Geometry/Frame.h"
#include "FunctionLayer/Material/Material.h"
#include "FunctionLayer/Medium/Medium.h"
#include "FunctionLayer/Shape/Entity.h"

#include <memory>

struct Intersection
{
    Point3d position;
    Normal3d geometryNormal;
    Normal3d geometryTangent;
    Normal3d geometryBitangent;
    Point2d uv;

    // shadingFrame
    Frame shFrame;

    Vec3d dpdu, dpdv;
    Normal3d dndu, dndv;

    // std::shared_ptr<Entity> object;
    const Entity *object;

    std::shared_ptr<Material> material;
    std::shared_ptr<Medium> mediumInside;
    std::shared_ptr<Medium> mediumOutside;

    Vec3d toLocal(const Vec3d &d) const
    {
        return shFrame.toLocal(d);
    }

    Vec3d toWorld(const Vec3d &d) const
    {
        return shFrame.toWorld(d);
    }
};
```

9.63 /mnt/renderer/Zero/src/FunctionLayer/Shape/Intersection.h File Reference

Intersection information on an object.

```
#include "CoreLayer/Geometry/Frame.h"
#include "CoreLayer/Geometry/Geometry.h"
#include "FunctionLayer/Material/Material.h"
#include "FunctionLayer/Medium/Medium.h"
#include "Entity.h"
#include <memory>
```

9.63.1 Detailed Description

Intersection information on an object.

Author

orbitchen

Version

0.1

Date

2022-04-30

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.64 Intersection.h

Go to the documentation of this file.

```

#pragma once

#include "CoreLayer/Geometry/Frame.h"
#include "CoreLayer/Geometry/Geometry.h"
#include "FunctionLayer/Material/Material.h"
#include "FunctionLayer/Medium/Medium.h"
#include "Entity.h"

#include <memory>

struct Intersection
{
    Point3d position;
    Normal3d geometryNormal;
    Normal3d geometryTangent;
    Normal3d geometryBitangent;
    Point2d uv;

    // shadingFrame
    Frame shFrame;

    Point3d dpdu, dpdv;
    Normal3d dndu, dndv;

    // std::shared_ptr<Entity> object;
    const Entity *object;

    std::shared_ptr<Material> material;
    std::shared_ptr<Medium> mediumInside;
    std::shared_ptr<Medium> mediumOutside;

    Vec3d toLocal(const Vec3d &d) const
    {
        return shFrame.toLocal(d);
    }

    Vec3d toWorld(const Vec3d &d) const
    {
        return shFrame.toWorld(d);
    }
};
```

9.65 /mnt/renderer/Zero/src/FunctionLayer/Light/AreaLight.cpp File Reference

Area light (abstract)

```
#include "AreaLight.h"
```

9.65.1 Detailed Description

Area light (abstract)

Author

Zhimin Fan

Version

0.1

Date

2022-05-20

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.66 /mnt/renderer/Zero/src/FunctionLayer/Light/AreaLight.h File Reference

Area light (abstract).

```
#include "CoreLayer/Geometry/Transform3d.h"
#include "Light.h"
```

9.66.1 Detailed Description

Area light (abstract).

Author

Zhimin Fan

Version

0.1

Date

2022-05-20

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.67 AreaLight.h

Go to the documentation of this file.

```

#include "CoreLayer/Geometry/Transform3d.h"
#include "Light.h"

class AreaLight : public Light
{
protected:
    std::shared_ptr<Transform3D> transform;

public:
    AreaLight(std::shared_ptr<Transform3D> transform3D);
};
```

9.68 /mnt/renderer/Zero/src/FunctionLayer/Light/DiffuseAreaLight.cpp File Reference

Diffuse Area light impl.

```
#include "DiffuseAreaLight.h"
#include "CoreLayer/Geometry/CoordConvertor.h"
```

9.68.1 Detailed Description

Diffuse Area light impl.

Author

Zhimin Fan

Version

0.1

Date

2022-05-20

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.69 /mnt/renderer/Zero/src/FunctionLayer/Light/DiffuseAreaLight.h File Reference

Diffuse Area light.

```
#include "AreaLight.h"
#include "CoreLayer/Geometry/Transform3d.h"
#include "FunctionLayer/Shape/Entity.h"
```

9.69.1 Detailed Description

Diffuse Area light.

Author

Zhimin Fan

Version

0.1

Date

2022-05-20

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.70 DiffuseAreaLight.h

Go to the documentation of this file.

```

#include "AreaLight.h"
#include "CoreLayer/Geometry/Transform3d.h"
#include "FunctionLayer/Shape/Entity.h"

class DiffuseAreaLight : public AreaLight
{
    std::shared_ptr<Entity> shape;
    Spectrum radiance;

public:
    DiffuseAreaLight(std::shared_ptr<Entity> shape,
                     Spectrum radiance);
    virtual LightSampleResult evalEnvironment(const Ray &ray) override;
    virtual LightSampleResult eval(const Ray& ray, const Intersection &its, const Vec3d &d) override;
    virtual LightSampleResult sampleEmit(const Point2d &positionSample, const Point2d &directionSample, float time) override;
    virtual LightSampleResult sampleDirect(const Intersection &its, const Point2d &sample, float time) override;
};
```

9.71 Light.h

```

#pragma once
#include "CoreLayer/Geometry/Geometry.h"
#include "CoreLayer/ColorSpace/Color.h"
#include "CoreLayer/Ray/Ray.h"
#include "FunctionLayer/Intersection.h"

struct LightSampleResult
{
    // @brief Spectrum from light.
    Spectrum s;

    // @brief Point on an object where receives light if exists.
    Point3d src;

    // @brief Point on a light where emits light if exists.
    Point3d dst;

    // @brief Direction from src to dst.
    Vec3d wi;

    Normal3d dstNormal;
    Point2d uv;

    // @brief PDF of direct light sampling.
    double pdfDirect;
    // @brief Positional emission PDF if exists.
    double pdfEmitPos;
    // @brief Directional emission PDF if exists.
    double pdfEmitDir;

    // @brief FALSE for area and volume light, TRUE for point and etc
    bool isDeltaPos;
    // @brief TRUE for distant light
    bool isDeltaDir;
};

class Light
{
public:
    // @brief Assume that the given ray hits nothing in the scene.
    // Note that this function will not return a direct light sampling PDF.
    virtual LightSampleResult evalEnvironment(const Ray &ray) = 0;
    // @brief Assume that the given intersection is on the emitter.
    // Note that param ray is only used to fill 'src' field.
    virtual LightSampleResult eval(const Ray& ray, const Intersection &its, const Vec3d &d) = 0;
    // @brief Note that this function will not return a direct light sampling PDF.
    virtual LightSampleResult sampleEmit(const Point2d &positionSample, const Point2d &directionSample, float time) = 0;
    virtual LightSampleResult sampleDirect(const Intersection &its, const Point2d &sample, float time) = 0;
};
```

9.72 /mnt/renderer/Zero/src/FunctionLayer/Light/PointLight.cpp File Reference

Point light.

```
#include "CoreLayer/Geometry/CoordConvertor.h"
#include "PointLight.h"
```

9.72.1 Detailed Description

Point light.

Author

Zhimin Fan

Version

0.1

Date

2022-05-15

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.73 /mnt/renderer/Zero/src/FunctionLayer/Light/PointLight.h File Reference

Point light.

```
#include "Light.h"
#include "CoreLayer/Geometry/Transform3d.h"
```

9.73.1 Detailed Description

Point light.

Author

Zhimin Fan

Version

0.1

Date

2022-05-15

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.74 PointLight.h

Go to the documentation of this file.

```
#pragma once

#include "Light.h"
#include "CoreLayer/Geometry/Transform3d.h"

#pragma  once

class PointLight : public Light, public Transform3D
{
protected:
    Spectrum intensity;

public:
    PointLight(const Spectrum &intensity, const Point3d &center);
    virtual void apply() override;
    virtual LightSampleResult evalEnvironment(const Ray &ray) override;
    virtual LightSampleResult eval(const Ray &ray, const Intersection &its, const Vec3d &d) override;
    virtual LightSampleResult sampleEmit(const Point2d &positionSample, const Point2d &directionSample,
    float time) override;
    virtual LightSampleResult sampleDirect(const Intersection &its, const Point2d &sample, float time)
    override;
};
```

9.75 /mnt/renderer/Zero/src/FunctionLayer/Material/BSSRDF/BSSRDF.h File Reference

BSSRDF.

9.75.1 Detailed Description

BSSRDF.

Author

orbitchen

Version

0.1

Date

2022-04-30

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.76 BSSRDF.h

Go to the documentation of this file.

```
#pragma once

class BSSRDF
{
    // TODO
};
```

9.77 /mnt/renderer/Zero/src/FunctionLayer/Material/BxDF/BxDF.h File Reference

BxDF, including BRDF and BTDF.

```
#include "CoreLayer/ColorSpace/Color.h"
#include "CoreLayer/Geometry/Geometry.h"
#include "CoreLayer/Math/Warp.h"
```

9.77.1 Detailed Description

BxDF, including BRDF and BTDF.

Author

orbitchen

Version

0.1

Date

2022-04-30

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.78 BxDF.h

Go to the documentation of this file.

```

#pragma once

#include "CoreLayer/ColorSpace/Color.h"
#include "CoreLayer/Geometry/Geometry.h"
#include "CoreLayer/Math/Warp.h"
struct BxDFSampleResult
{
    Spectrum s;
    Vec3d directionIn;
    double pdf;
    bool isSpecular;
};

// @brief BxDF. out == rays from/to camera, in == rays from/to objects/lights.
class BxDF
{

public:
    virtual Spectrum f(const Vec3d &out, const Vec3d &in) const = 0;

    virtual Vec3d sampleWi(const Vec3d &out, const Point2d& sample) const = 0;

    virtual double pdf(const Vec3d &out, const Vec3d &in) const = 0;

    virtual BxDFSampleResult sample(const Vec3d &out, const Point2d& sample) const = 0;

    virtual bool isSpecular() const = 0;
};
```

9.79 /mnt/renderer/Zero/src/FunctionLayer/Material/BxDF/Dielectric.h File Reference

Dielectric Bxdf.

```
#include "CoreLayer/Geometry/Frame.h"
#include "BxDF.h"
```

9.79.1 Detailed Description

Dielectric Bxdf.

Author

Junping Yuan

Version

0.1

Date

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.80 Dielectric.h

Go to the documentation of this file.

```

#pragma once
#include "CoreLayer/Geometry/Frame.h"
#include "BxDF.h"
class Dielectric : public  BxDF{
public:
    Dielectric(float m_intIOR =1.5046f,float m_extIOR = 1.00277f) : m_intIOR(m_intIOR),
    m_extIOR(m_extIOR){

    }

    virtual Spectrum f(const Vec3d &wo, const Vec3d &wi) const ;

    virtual Vec3d sampleWi(const Vec3d &wo, const Point2d &sample) const ;

    virtual double pdf(const Vec3d &wo, const Vec3d &wi) const ;


    virtual BxDFSampleResult sample(const Vec3d &wo, const Point2d &sample) const ;

    virtual bool isSpecular() const ;


private:
   float m_intIOR;
   float m_extIOR;

    static Vec3d reflect(const Vec3d &wi) {
        return Vec3d {
                -wi[0], -wi[1], wi[2]
        };
    }

    Vec3d refract(const Vec3d &wi) const {
        float cosThetaI = Frame::cosTheta(wi),
                eta = cosThetaI > 0 ? m_extIOR / m_intIOR : m_intIOR / m_extIOR,
                cosThetaT = std::sqrt(
                1 - eta*eta*(1-cosThetaI*cosThetaI)
        );
        return Vec3d {
                - wi[0] * eta,
                - wi[1] * eta,
                cosThetaI > 0 ? -cosThetaT : cosThetaT
        };
    }


};


```

9.81 /mnt/renderer/Zero/src/FunctionLayer/Material/BxDF/Diffuse.cpp File Reference

```
#include "Diffuse.h"
```

9.81.1 Detailed Description

Author

Junping Yuan

Version

0.1

Date

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.82 /mnt/renderer/Zero/src/FunctionLayer/Material/BxDF/Diffuse.h File Reference

diffuse bxdf

```
#include "CoreLayer/Geometry/Frame.h"
#include "BxDF.h"
```

9.82.1 Detailed Description

diffuse bxdf

Author

Junping Yuan

Version

0.1

Date

2022/06/06

Copyright

Copyright (c) 2022

9.83 Diffuse.h

Go to the documentation of this file.

```
#pragma  once
#include "CoreLayer/Geometry/Frame.h"
#include "BxDF.h"



class Diffuse : public  BxDF{
private:

    Spectrum  albedo;

public:

    Diffuse(Spectrum albedo);

    virtual Spectrum f(const Vec3d &wo, const Vec3d &wi) const;

    virtual Vec3d sampleWi(const Vec3d &wo, const Point2d &sample) const;

    virtual double pdf(const Vec3d &wo, const Vec3d &wi) const;

    virtual BxDFSampleResult sample(const Vec3d &wo, const Point2d &sample) const;

    virtual bool isSpecular() const;

};
```

9.84 Microfacet.h

```
#pragma  once

#include "BxDF.h"
#include "CoreLayer/Geometry/Frame.h"

class Mircofacet : public  BxDF {


public:
    Mircofacet(Spectrum kd, double intIOR, double extIOR,double alpha);

    virtual Spectrum f(const Vec3d &wo, const Vec3d &wi) const;

    virtual Vec3d sampleWi(const Vec3d &wo, const Point2d &sample) const;

    virtual double pdf(const Vec3d &wo, const Vec3d &wi) const;

    virtual BxDFSampleResult sample(const Vec3d &wo, const Point2d &sample) const;

    virtual bool isSpecular() const;
private:
    Spectrum kd;
    double ks;
    double intIOR,extIOR,alpha;
};
```

9.85 /mnt/renderer/Zero/src/FunctionLayer/Material/BxDF/Mirror.h File Reference

Mirror Material.

```
#include "CoreLayer/Geometry/Frame.h"
#include "BxDF.h"
```

9.85.1 Detailed Description

Mirror Material.

Author

Junping Yuan

Version

0.1

Date

2022/6/8

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.86 Mirror.h

Go to the documentation of this file.

```

#include "CoreLayer/Geometry/Frame.h"
#include "BxDF.h"

#pragma  once

class Mirror : public  BxDF {

    virtual Spectrum f(const Vec3d &wo, const Vec3d &wi) const;

    virtual Vec3d sampleWi(const Vec3d &wo, const Point2d &sample) const;

    virtual double pdf(const Vec3d &wo, const Vec3d &wi) const;

    virtual BxDFSampleResult sample(const Vec3d &wo, const Point2d &sample) const;

    virtual bool isSpecular() const;
};

```

9.87 TestMirrorBxdf.h

```
#pragma once

#include "BxDF.h"
#include "FunctionLayer/Intersection.h"

class TestMirrorBxdf : public BxDF
{
public:
    virtual Spectrum f(const Vec3d &wo, const Vec3d &wi) const;

    virtual Vec3d sampleWi(const Vec3d &wo, const Point2d &sample) const;

    virtual double pdf(const Vec3d &wo, const Vec3d &wi) const;

    virtual BxDFSampleResult sample(const Vec3d &wo, const Point2d &sample) const;

    virtual bool isSpecular() const;
};
```

9.88 /mnt/renderer/Zero/src/FunctionLayer/Material/DelectricMaterial.h File Reference

```
#include "Material.h"
#include "FunctionLayer/Material/BxDF/Dielectric.h"
#include "FunctionLayer/Intersection.h"
#include "FunctionLayer/Texture/Texture.h"
```

9.88.1 Detailed Description

Author

Junping Yuan

Version

0.1

Date

2022/6/11

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.89 DelectricMaterial.h

Go to the documentation of this file.

```

#pragma once
#include "Material.h"
#include "FunctionLayer/Material/BxDF/Dielectric.h"
#include "FunctionLayer/Intersection.h"
#include "FunctionLayer/Texture/Texture.h"

class DelectricMaterial : public Material{

public:
    virtual std::shared_ptr<BxDF> getBxDF(Intersection intersect) const;

    virtual std::shared_ptr<BSSRDF> getBSSRDF(Intersection intersect) const;

    DelectricMaterial(const std::shared_ptr<Texture<float>> & m_intIDR, const
    std::shared_ptr<Texture<float>> & m_extIDR) : m_intIDR(m_intIDR)
    {

    }

    DelectricMaterial(){

    }
private:
    std::shared_ptr<Texture<float>> m_intIDR;
    std::shared_ptr<Texture<float>> m_extIDR;

};


```

9.90 /mnt/renderer/Zero/src/FunctionLayer/Material/Material.h File Reference

Material of an object. Generate BxDF and/or BSSRDF.

```
#include "FunctionLayer/Material/BxDF/BxDF.h"
#include "FunctionLayer/Material/BSSRDF/BSSRDF.h"
#include <memory>
```

9.90.1 Detailed Description

Material of an object. Generate BxDF and/or BSSRDF.

Author

orbitchen

Version

0.1

Date

2022-04-30

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.91 Material.h

Go to the documentation of this file.

```

#pragma once

#include "FunctionLayer/Material/BxDF/BxDF.h"
#include "FunctionLayer/Material/BSSRDF/BSSRDF.h"
#include <memory>
//#include "Intersection.h"



struct Intersection;

class Material
{
public:

    virtual std::shared_ptr<BxDF> getBxDF(Intersection intersect) const = 0;
    virtual std::shared_ptr<BSSRDF> getBSSRDF(Intersection intersect) const = 0;
};
```

9.92 /mnt/renderer/Zero/src/FunctionLayer/Material/MatteMaterial.h File Reference

```
#include "Material.h"
#include "FunctionLayer/Intersection.h"
#include "FunctionLayer/Texture/Texture.h"
```

9.92.1 Detailed Description

Author

Junping Yuan

Version

0.1

Date

2022/6/7

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.93 MatteMaterial.h

Go to the documentation of this file.

```

#pragma  once
#include "Material.h"
#include "FunctionLayer/Intersection.h"
#include "FunctionLayer/Texture/Texture.h"

class MatteMaterial     : public  Material{
private:
    std::shared_ptr<Texture<Spectrum» kd ;
public:
    MatteMaterial(const std::shared_ptr<Texture<Spectrum» & kd );

    virtual std::shared_ptr<BxDF> getBxDF(Intersection intersect) const;

    virtual std::shared_ptr<BSSRDF> getBSSRDF(Intersection intersect) const;

};


```

9.94 /mnt/renderer/Zero/src/FunctionLayer/Material/MirrorMaterial.h File Reference

```
#include "Material.h"
#include "FunctionLayer/Material/BxDF/Mirror.h"
#include "FunctionLayer/Intersection.h"
```

9.94.1 Detailed Description

Author

Junping Yuan

Version

0.1

Date

2022/6/9

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.95 MirrorMaterial.h

Go to the documentation of this file.

```

#include "Material.h"
#include "FunctionLayer/Material/BxDF/Mirror.h"
#include "FunctionLayer/Intersection.h"

#pragma  once
class MirrorMaterial  : public  Material{


public:
    virtual std::shared_ptr<BxDF> getBxDF(Intersection intersect) const;
    virtual std::shared_ptr<BSSRDF> getBSSRDF(Intersection intersect) const;
};
```

9.96 TestMirror.h

```

#pragma once
#include "Material.h"
#include "FunctionLayer/Material/BxDF/TestMirrorBxdf.h"

class TestMirror : public Material
{
private:
public:
    TestMirror();
    virtual std::shared_ptr<BxDF> getBxDF(Intersection intersect) const;
    virtual std::shared_ptr<BSSRDF> getBSSRDF(Intersection intersect) const;
};
```

9.97 /mnt/renderer/Zero/src/FunctionLayer/Medium/Medium.h File Reference

Medium.

9.97.1 Detailed Description

Medium.

Author

orbitchen

Version

0.1

Date

2022-04-30

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.98 Medium.h

Go to the documentation of this file.

```

#pragma once

class Medium
{
    // TODO
};
```

9.99 DirectSampler.h

```
#pragma once

#include "Sampler.h"

#pragma  once
class DirectSampler : public Sampler
{
public:
    virtual double sample() const;
    virtual std::vector<double> sample(int num) const;
};
```

9.100 Independent.h

```
#pragma once
#include "Sampler.h"

class IndependentSampler : public Sampler {
public:
    IndependentSampler() = default;

    virtual ~IndependentSampler() = default;

    virtual void startPixel(const Point2i &pixelPosition) override {
        // do nothing
    }

    virtual void nextSample() override {
        // do nothing
    }

    virtual double sample1D() override {
        return rng();
    }

    virtual Point2d sample2D() override {
        return Point2d{rng(), rng()};
    }
};
```

9.101 /mnt/renderer/Zero/src/FunctionLayer/Sampler/Sampler.h File Reference

generate random numbers between (0,1).

```
#include <vector>
#include "CoreLayer/Adapter/random.h"
#include "CoreLayer/Geometry/Geometry.h"
#include "CoreLayer/Math/Common.h"
```

Classes

- class PixelSampler

  Base class for pixel sampler, which generate specific samples before rendering each pixel need the dimensions and samplesPerPixels when construct.

Functions

- template<typename T >
  void shuffle (std::vector< T > &samples, RandomNumberGenerator &rng)

  Shuffle the vector samples.

9.101.1 Detailed Description

generate random numbers between (0,1).

Author

orbitchen

Version

0.1

Date

2022-04-30

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.101.2 Function Documentation

9.101.2.1 shuffle()

```
template<typename T >
void shuffle (
            std::vector< T > & samples,
            RandomNumberGenerator & rng )
```

Shuffle the vector samples.

Template Parameters

T	

Parameters

samples	

9.102 Sampler.h

Go to the documentation of this file.

```

#pragma once
```

```

#include <vector>
#include "CoreLayer/Adapter/random.h"
#include "CoreLayer/Geometry/Geometry.h"
#include "CoreLayer/Math/Common.h"

struct CameraSample;

class Sampler
{
protected:
    RandomNumberGenerator rng;
public:
    Sampler() = default;

    virtual ~Sampler() = default;

    virtual void startPixel(const Point2i &pixelPosition) = 0;

    virtual void nextSample() = 0;

    virtual double sample1D() = 0;

    virtual Point2d sample2D() = 0;

    CameraSample getCameraSample();

};

template<typename T>
void shuffle(std::vector<T> &samples, RandomNumberGenerator &rng) {
    int count = samples.size();
    for (int i = 0; i < count; ++i) {
        int other = rng(i, count);
        std::swap(samples[i], samples[other]);
    }
}

class PixelSampler : public Sampler {
protected:
    Point2i pixelPosition;
    std::vector<std::vector<double»  samples1D;
    std::vector<std::vector<Point2d» samples2D;
    int sppSqrt;
    int samplesPerPixel;
    int nDimensions;
    int curSamplePixelIndex;
    int curDimensionIndex1D;
    int curDimensionIndex2D;

    virtual void generateSamples1D(std::vector<double> &samples) = 0;
    virtual void generateSamples2D(std::vector<Point2d> &samples) = 0;

public:
    PixelSampler() = delete;

    PixelSampler(int _sppSqrt)
        : sppSqrt(_sppSqrt), samplesPerPixel(_sppSqrt * _sppSqrt), nDimensions(4), curSamplePixelIndex(0), curDimensionIndex1D(0), curDimensionIndex2D(0) {
        for (int i = 0; i < nDimensions; ++i) {
            samples1D.emplace_back(std::vector<double>(samplesPerPixel));
            samples2D.emplace_back(std::vector<Point2d>(samplesPerPixel));
        }
    }

    PixelSampler(int _sppSqrt, int _nDimensions)
        : sppSqrt(_sppSqrt) ,samplesPerPixel(_sppSqrt * _sppSqrt), nDimensions(_nDimensions), curSamplePixelIndex(0), curDimensionIndex1D(0), curDimensionIndex2D(0) {
        for (int i = 0; i < nDimensions; ++i) {
            samples1D.emplace_back(std::vector<double>(samplesPerPixel));
            samples2D.emplace_back(std::vector<Point2d>(samplesPerPixel));
        }
    }

    virtual ~PixelSampler() = default;

     // @brief The sampler may change the sampling stragety at different pixel location ,record the current pixel location
     // @param _pixelPositon location of current pixel
     virtual void startPixel(const Point2i &_pixelPositon) override {
         // record the position
         pixelPosition = _pixelPositon;
         // reset the sample index
         curDimensionIndex1D = curDimensionIndex2D = curSamplePixelIndex = 0;
         // generate the samples will be used in current pixel
         for (int i = 0; i < nDimensions; ++i) {
             generateSamples1D(samples1D[i]);
             shuffle(samples1D[i], rng);
             generateSamples2D(samples2D[i]);
             shuffle(samples2D[i], rng);
         }
     }

     // @brief Start the next sample at a specific pixel
     virtual void nextSample() override {
         ++curSamplePixelIndex;
         curDimensionIndex1D = curDimensionIndex2D = 0;
     }

     // @brief Sample a double in [0, 1], if curDimension exceeds the nDimensions, just return rng()
     // @return A double sample
     virtual double sample1D() override {
         if (curDimensionIndex1D < nDimensions)
             return samples1D[curDimensionIndex1D++][curSamplePixelIndex];
         else
             return rng();
     }

     // @brief Sample a point2d in [0, 1]^2, if curDimension exceeds the nDimensions, just return {rng(),
    rng()}
     // @return A point2d sample
     virtual Point2d sample2D() override {
         if (curDimensionIndex2D < nDimensions)
             return samples2D[curDimensionIndex2D++][curSamplePixelIndex];
         else
             return Point2d{rng(), rng()};
     }
 };
```

9.103 /mnt/renderer/Zero/src/FunctionLayer/Sampler/Stratified.h File Reference

Stratified sampler.

```
#include "Sampler.h"
```

9.103.1 Detailed Description

Stratified sampler.

Author

Chenxi Zhou

Version

0.1

Date

2022-06-24

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.104 Stratified.h

Go to the documentation of this file.

```

#pragma once
#include "Sampler.h"

class StratifiedSampler : public PixelSampler {
    // fill the samples1D
    virtual void generateSamples1D(std::vector<double> &samples) override;
    // fill the samples2D
    virtual void generateSamples2D(std::vector<Point2d> &samples) override;
public:
    StratifiedSampler() = delete;

    StratifiedSampler(int sppSqrt) : PixelSampler(sppSqrt) { }

    StratifiedSampler(int sppSqrt, int _nDimensions) : PixelSampler(sppSqrt, _nDimensions) { }

    ~StratifiedSampler() = default;

};
```

9.105 /mnt/renderer/Zero/src/FunctionLayer/Shape/Entity.h File Reference

Objects that can be intersected with ray.

```
#include "CoreLayer/Geometry/Transform3d.h"
#include "FunctionLayer/Material/Material.h"
#include "CoreLayer/Geometry/BoundingBox.h"
#include "CoreLayer/Ray/Ray.h"
#include <optional>
#include <memory>
```

9.105.1 Detailed Description

Objects that can be intersected with ray.

Author

orbitchen

Version

0.1

Date

2022-04-30

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.106 Entity.h

Go to the documentation of this file.

```

#pragma once

#include "CoreLayer/Geometry/Transform3d.h"
#include "FunctionLayer/Material/Material.h"
#include "CoreLayer/Geometry/BoundingBox.h"
#include "CoreLayer/Ray/Ray.h"

#include <optional>
#include <memory>

struct Intersection;
class Light;

class Entity : public Transform3D
{
public:
    std::shared_ptr<Light> lightPtr;
    std::shared_ptr<Material> material;
    //@brief Returns the intersection of the entity and the ray
    virtual std::optional<Intersection> intersect(const Ray &r) const = 0;
    //@brief Return ptr to light when primitive is a emitter. Otherwise, return nullptr.
    virtual std::shared_ptr<Light> getLight() const = 0;
    virtual void setLight(std::shared_ptr<Light> light) = 0;
    virtual double area() const = 0;
    virtual Intersection sample(const Point2d &positionSample) const = 0;
    //@brief Return the bounding box of the entity
    virtual BoundingBox3f WorldBound() const = 0;
};
```

9.107 /mnt/renderer/Zero/src/FunctionLayer/Shape/Sphere.cpp File Reference

Sphere. Only for test. Ignore transform.

```
#include "Sphere.h"
#include "FunctionLayer/Intersection.h"
#include "CoreLayer/Geometry/CoordConvertor.h"
```

9.107.1 Detailed Description

Sphere. Only for test. Ignore transform.

Author

Zhimin Fan

Version

0.1

Date

2022-05-30

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.108 /mnt/renderer/Zero/src/FunctionLayer/Shape/Sphere.h File Reference

Sphere. Only for test. Ignore transform.

```
#include "Entity.h"
#include <optional>
```

9.108.1 Detailed Description

Sphere. Only for test. Ignore transform.

Author

Zhimin Fan

Version

0.1

Date

2022-05-30

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.109 Sphere.h

Go to the documentation of this file.

```

#pragma once

#include "Entity.h"
#include <optional>

class Sphere : public Entity
{
protected:
    double radius;
    Point3d center;
    virtual void apply() override;

public:
    Sphere(Point3d _center, double _radius, std::shared_ptr<Material> _material);
    virtual std::optional<Intersection> intersect(const Ray &r) const;
    virtual double area() const;
    virtual Intersection sample(const Point2d &positionSample) const;
    virtual std::shared_ptr<Light> getLight() const;
    virtual void setLight(std::shared_ptr<Light> light);
    virtual BoundingBox3f WorldBound() const;
};
```

9.110 /mnt/renderer/Zero/src/FunctionLayer/Shape/Triangle.cpp File Reference

Triangle implementation, transform not implemented yet.

```
#include "Triangle.h"
#include "FunctionLayer/Intersection.h"
```

9.110.1 Detailed Description

Triangle implementation, transform not implemented yet.

Author

Pengpei Hong

Version

0.1

Date

2022-06-26

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.111 /mnt/renderer/Zero/src/FunctionLayer/Shape/Triangle.h File Reference

Triangle implementation, transform not implemented yet.

```
#include "Entity.h"
#include <optional>
```

9.111.1 Detailed Description

Triangle implementation, transform not implemented yet.

Author

Pengpei Hong

Version

0.1

Date

2022-06-26

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.112 Triangle.h

Go to the documentation of this file.

```

#pragma once

#include "Entity.h"
#include <optional>

class TriangleMesh{
public:
    const int nTriangles;
    const int nVertices;
    std::shared_ptr<std::vector<int» vertexIndices;//the i-th triangle maps vertex[i * 3], vertex[i * 3 + 1], vertex[i * 3 + 2]
    std::shared_ptr<std::vector<Point3d» p; //geometry position
    std::shared_ptr<std::vector<Normal3d» n; //shading normal
    std::shared_ptr<std::vector<Vec3d» s; //shading tangent(optional)
    std::shared_ptr<std::vector<Point2d» uv; //uv coordinates
    std::shared_ptr<Material> material;
    /*
    @brief constructor of TriangleMesh, parameter pointers(except material) will point to nullptr after construction
    @param <_nTriangles> number of triangle
    @param <_nVertices> number of vertices(equal to the sizeof of _p)
    @param <_p> position of vertices
    @param <_n> shading normal of vertices(optional)
    @param <_s> shading tangent of vertices(optional)
    @param <_uv> uv coordinates of vertices(optional)
    @param <_material> material of triangle mesh
    */
    TriangleMesh(const int& _nTriangles, const int& _nVertices, const std::shared_ptr<std::vector<int»& _vertexIndices, const std::shared_ptr<std::vector<Point3d»& _p, const std::shared_ptr<std::vector<Normal3d»& _n, const std::shared_ptr<std::vector<Vec3d»& _s, const std::shared_ptr<std::vector<Point2d»& _uv, const std::shared_ptr<Material>& _material);
};

//vertices in counter-clockwise order
class Triangle: public Entity{
protected:
    int vertexId[3];
    int faceId;
    std::shared_ptr<TriangleMesh> mesh;
public:
    virtual void apply() override;
    Triangle(const std::shared_ptr<TriangleMesh>& _mesh, const int& _faceId);
    Triangle(const std::vector<Point3d>& points, const std::shared_ptr<Material>& _material);
    virtual std::optional<Intersection> intersect(const Ray& r) const;
    virtual double area() const;
    virtual Intersection sample(const Point2d& positionSample) const;
    virtual std::shared_ptr<Light> getLight() const;
    virtual void setLight(std::shared_ptr<Light> light);
    virtual BoundingBox3f WorldBound() const;
};
```

9.113 /mnt/renderer/Zero/src/FunctionLayer/Texture/ImageTexture.cpp File Reference

Image texture.

```
#include "ImageTexture.h"
```

9.113.1 Detailed Description

Image texture.

Author

Zhimin Fan

Version

0.1

Date

2022-05-10

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.114 /mnt/renderer/Zero/src/FunctionLayer/Texture/ImageTexture.h File Reference

Image texture.

```
#include <cmath>
#include "CoreLayer/ColorSpace/Color.h"
#include "ResourceLayer/File/Image.h"
#include "FunctionLayer/Intersection.h"
#include "Texture.h"
#include "TextureMapping.h"
```

9.114.1 Detailed Description

Image texture.

Author

Zhimin Fan

Version

0.1

Date

2022-05-03

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.115 ImageTexture.h

Go to the documentation of this file.

```cpp

#pragma once

#include <cmath>
#include "CoreLayer/ColorSpace/Color.h"
#include "ResourceLayer/File/Image.h"
#include "FunctionLayer/Intersection.h"
#include "Texture.h"
#include "TextureMapping.h"

enum class WrapMode
{
    // todo
};

template <typename T>
class PrefilteredImage
{
protected:
    WrapMode wrapMode;

public:
    virtual void setWrapMode(enum WrapMode _wrapMode);
    virtual WrapMode getWrapMode();
    // todo: other common parameters

    virtual T eval(const TextureCoord2D &coord) = 0;
    virtual T texel(const Point2i &coord) = 0;
    virtual void loadImage(const std::string &filename) = 0;
};

template <typename T>
class DirectImage : public PrefilteredImage<T>
{
protected:
    std::shared_ptr<Image> image;

public:
    virtual void loadImage(const std::string &filename);
    virtual T eval(const TextureCoord2D &coord);
    virtual T texel(const Point2i &coord);
};

template <typename T>
class LinearMIPMap : public PrefilteredImage<T>
{
protected:
    std::shared_ptr<Image> image;

public:
    virtual void loadImage(const std::string &filename);
    virtual T eval(const TextureCoord2D &coord);
    virtual T texel(const Point2i &coord);
};

template <typename Treturn, typename Tmemory>
class ImageTexture : public StdTexture<Treturn, TextureCoord2D>
{
protected:
    std::shared_ptr<PrefilteredImage<Tmemory» imageSampler;

public:
    ImageTexture(const std::string &filename,
                 std::shared_ptr<TextureMapping2D> mapping = std::make_shared<UVTextureMapping2D>()); // using default sampler
    ImageTexture(const std::string &filename,
                 std::shared_ptr<PrefilteredImage<Tmemory» imageSampler,
                 std::shared_ptr<TextureMapping2D> mapping = std::make_shared<UVTextureMapping2D>());
    virtual Treturn eval(const TextureCoord2D &coord) const override;
};

// * Example: Creating a Image-based Color Texture using UV coordinates from mesh
// * >  ImageTexture<RGB3>("1.jpg");
// * >  ImageTexture<RGB3>("1.jpg", std::make_shared<DirectSampler>());
// * >  ImageTexture<RGB3>("1.jpg", std::make_shared<UVTextureMapping2D>());
// * >  ImageTexture<RGB3>("1.jpg", std::make_shared<DirectSampler>(), std::make_shared<UVTextureMapping2D>());

// * Example: Create a Image-based Normal Map (wip)
// * since normal cannot be directly interpolated, you need to provide T with some compact NDF type
// * The NDF must provide convertor from RGB3 and some accessors for shadings (depends on implementation of material)
```

```

template <typename T>
void PrefilteredImage<T>::setWrapMode(enum WrapMode _wrapMode)
{
    wrapMode = _wrapMode;
}

template <typename T>
WrapMode PrefilteredImage<T>::getWrapMode()
{
    return wrapMode;
}

template <typename T>
void DirectImage<T>::loadImage(const std::string &filename)
{
    image = std::make_shared<Image>(filename);
}

template <typename T>
T DirectImage<T>::texel(const Point2i &coord)
{
    // todo
    return 0.0;
}

template <typename Treturn, typename Tmemory>
ImageTexture<Treturn, Tmemory>::ImageTexture(const std::string &filename,
                                             std::shared_ptr<TextureMapping2D> mapping) :
   StdTexture<Treturn, TextureCoord2D>(mapping)
{
    imageSampler = std::make_shared<DirectImage<Tmemory»();
    imageSampler->loadImage(filename);
}



template <typename Treturn, typename Tmemory>
ImageTexture<Treturn, Tmemory>::ImageTexture(const std::string &filename,
                                             std::shared_ptr<PrefilteredImage<Tmemory» imageSampler,
                                             std::shared_ptr<TextureMapping2D> mapping) :
   imageSampler(imageSampler), StdTexture<Treturn, TextureCoord2D>(mapping)
{
    this->imageSampler->loadImage(filename);
}
```

9.116 /mnt/renderer/Zero/src/FunctionLayer/Texture/Procedural← Texture.cpp File Reference

Procedural texture.

```
#include "ProceduralTexture.h"
#include "TextureMapping.h"
```

9.116.1 Detailed Description

Procedural texture.

Author

Zhimin Fan

Version

0.1

Date

2022-05-13

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.117 /mnt/renderer/Zero/src/FunctionLayer/Texture/ProceduralTexture.h File Reference

Procedural texture.

`#include "Texture.h"`

9.117.1 Detailed Description

Procedural texture.

Author

Zhimin Fan

Version

0.1

Date

2022-05-13

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.118 ProceduralTexture.h

Go to the documentation of this file.

```
#include "Texture.h"

class Checkerboard2D : public StdTexture<double, TextureCoord2D>
{
protected:
public:
    Checkerboard2D();
    using StdTexture::eval;
    virtual double eval(const TextureCoord2D &coord) const override;
};

class Checkerboard3D : public StdTexture<double, TextureCoord3D>
{
protected:
public:
    Checkerboard3D();
    using StdTexture::eval;
    virtual double eval(const TextureCoord3D &coord) const override;
};
```

9.119 /mnt/renderer/Zero/src/FunctionLayer/Texture/Texture.h File Reference

Texture of different types.

```
#include <cmath>
#include "FunctionLayer/Intersection.h"
#include "CoreLayer/Geometry/Geometry.h"
```

9.119.1 Detailed Description

Texture of different types.

Author

Zhimin Fan

Version

0.1

Date

2022-05-03

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.120 Texture.h

Go to the documentation of this file.

```

#pragma once

#include <cmath>
#include "FunctionLayer/Intersection.h"
#include "CoreLayer/Geometry/Geometry.h"

// Tvalue can be float or any specturm type
template <typename Tvalue>
class Texture
{
public:
    virtual Tvalue eval(const Intersection &intersection) const = 0;
};


// @brief Struct for texture coord C and dC/dx, dC/dy.
template <typename Tpos, typename Tvec>
struct TextureCoord
{
    Tpos coord;
    Tvec dcdx;
    Tvec dcdy;
};

typedef TextureCoord<Point2d, Vec2d> TextureCoord2D;
typedef TextureCoord<Point3d, Vec3d> TextureCoord3D;

// @brief TextureMapping maps an intersection to texture coordinates of type Tcoord.
template <typename Tcoord>
class TextureMapping
{
protected:
public:
    virtual Tcoord mapping(const Intersection &intersection) const = 0;
};

typedef TextureMapping<TextureCoord2D> TextureMapping2D;
typedef TextureMapping<TextureCoord3D> TextureMapping3D;

template <typename Tvalue>
class ConstantTexture : public Texture<Tvalue>
{
protected:
    Tvalue value;

public:
    ConstantTexture(const Tvalue &value);
    virtual Tvalue eval(const Intersection &intersection) const;
};

template <typename Tvalue>
class MixTexture : public Texture<Tvalue>
{
protected:
    std::shared_ptr<Texture<Tvalue>» srcA;
    std::shared_ptr<Texture<Tvalue>» srcB;
    std::shared_ptr<Texture<double>» factor;

public:
    MixTexture(std::shared_ptr<Texture<Tvalue>» srcA,
               std::shared_ptr<Texture<Tvalue>» srcB,
               std::shared_ptr<Texture<double>» factor);

    virtual Tvalue eval(const Intersection &intersection) const;
};

// @brief StdTexture refers to textures that needs a texture mapping to generate Tcoord from Intersection
template <typename Tvalue, typename Tcoord>
class StdTexture : public Texture<Tvalue>
{
protected:
    std::shared_ptr<TextureMapping<Tcoord» mapping;

public:
    StdTexture();
    StdTexture(std::shared_ptr<TextureMapping<Tcoord» mapping);

    // @brief This function just redirects the query to eval(coord) using member TextureMapping. Derived should NOT overwrite this.
    virtual Tvalue eval(const Intersection &intersection) const final;

    // @brief Eval texture value at given texture coord. (Derived needs to implement this)
    virtual Tvalue eval(const Tcoord &coord) const = 0;
};

template <typename Tvalue>
ConstantTexture<Tvalue>::ConstantTexture(const Tvalue &value) : value(value)
{
}

template <typename Tvalue>
Tvalue ConstantTexture<Tvalue>::eval(const Intersection &intersection) const
{
    return value;
}

template <typename Tvalue>
MixTexture<Tvalue>::MixTexture(std::shared_ptr<Texture<Tvalue>» srcA,
                               std::shared_ptr<Texture<Tvalue>» srcB,
                               std::shared_ptr<Texture<double>» factor) : srcA(srcA), srcB(srcB), factor(factor)
{
}

template <typename Tvalue>
Tvalue MixTexture<Tvalue>::eval(const Intersection &intersection) const
{
    double alpha = factor->eval(intersection);
    return srcA->eval(intersection) * alpha + srcB->eval(intersection) * (1 - alpha);
}

template <typename Tvalue, typename Tcoord>
```

```
StdTexture<Tvalue, Tcoord>::StdTexture(std::shared_ptr<TextureMapping<Tcoord» mapping): mapping(mapping)
{
}

template <typename Tvalue, typename Tcoord>
Tvalue StdTexture<Tvalue, Tcoord>::eval(const Intersection &intersection) const
{
    return eval(mapping->mapping(intersection));
}
```

9.121 /mnt/renderer/Zero/src/FunctionLayer/Texture/TextureMapping.cpp File Reference

Texture mapping.

```
#include "TextureMapping.h"
```

9.121.1 Detailed Description

Texture mapping.

Author

Zhimin Fan

Version

0.1

Date

2022-05-10

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.122 /mnt/renderer/Zero/src/FunctionLayer/Texture/TextureMapping.h File Reference

Texture mapping.

```
#include <cmath>
#include "CoreLayer/ColorSpace/Color.h"
#include "Texture.h"
#include "FunctionLayer/Intersection.h"
```

9.122.1 Detailed Description

Texture mapping.

Author

Zhimin Fan

Version

0.1

Date

2022-05-03

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.123 TextureMapping.h

Go to the documentation of this file.

```

#pragma once

#include <cmath>
#include "CoreLayer/ColorSpace/Color.h"
#include "Texture.h"
#include "FunctionLayer/Intersection.h"
// Various kinds of TextureMapping

class UVTextureMapping2D : public TextureMapping<TextureCoord2D>
{
protected:
    // todo: add a affine transform
public:
    virtual TextureCoord2D mapping(const Intersection &intersection) const override;
};

class NaturalTextureMapping3D : public TextureMapping<TextureCoord3D>
{
protected:
    // todo: add a affine transform
public:
    virtual TextureCoord3D mapping(const Intersection &intersection) const override;
};
```

9.124 /mnt/renderer/Zero/src/FunctionLayer/TileGenerator/SequenceTileGenerator.cpp File Reference

Implemention of SequenceTileGenerator.

```
#include "SequenceTileGenerator.h"
```

9.124.1 Detailed Description

Implemention of SequenceTileGenerator.

Author

orbitchen

Version

0.1

Date

2022-7-7

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.125 /mnt/renderer/Zero/src/FunctionLayer/TileGenerator/SequenceTileGenerator.h File Reference

TileGenerator thats generate tiles sequentially, from top-left to down-right.

```
#include "TileGenerator.h"
```

9.125.1 Detailed Description

TileGenerator thats generate tiles sequentially, from top-left to down-right.

Author

orbitchen

Version

0.1

Date

2022-7-7

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.126 SequenceTileGenerator.h

Go to the documentation of this file.

```

#pragma once

#include "TileGenerator.h"

class SequenceTileGenerator : public TileGenerator
{

private:

    std::vector<int> xList;
    std::vector<int> yList;

    Point2i currentBeginIndex;
    // currentEndIndex=[currentBeginIndex.x+1,currentBeginIndex.y+1]
    bool reachedEnd=false;

protected:

    //@brief the size of a tile. The default size of a tile is 16x16.
    int size;

public:

    SequenceTileGenerator(const Point2i& _resolution,int _size=16);

    virtual std::vector<std::shared_ptr<Tile» generateTiles();

    virtual std::optional<std::shared_ptr<Tile» generateNextTile();

};
```

9.127 /mnt/renderer/Zero/src/FunctionLayer/TileGenerator/Tile↩Generator.cpp File Reference

Implemention of SquareTile and pointIterator.

```
#include "TileGenerator.h"
```

9.127.1 Detailed Description

Implemention of SquareTile and pointIterator.

Author

orbitchen

Version

0.1

Date

2022-7-7

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.128 /mnt/renderer/Zero/src/FunctionLayer/TileGenerator/TileGenerator.h File Reference

Generate tiles for different threads in renderer.

```
#include "CoreLayer/Geometry/Geometry.h"
#include <vector>
#include <mutex>
#include <optional>
```

9.128.1 Detailed Description

Generate tiles for different threads in renderer.

Author

orbitchen

Version

0.1

Date

2022-04-30

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.129 TileGenerator.h

Go to the documentation of this file.

```

#pragma once

#include "CoreLayer/Geometry/Geometry.h"

#include <vector>
#include <mutex>
#include <optional>

/*@brief Iterator for Point2i.
* It will generate points between pBegin and pEnd but pEnd is not included.
* e.g, pBegin=[0,0], pEnd=[2,2], and the generated points will be
* [0,0], [1,0], [0,1] and [1,1].
* */
class PointIterator
{

protected:

    // @brief the first point of a tile.
    Point2i pBegin;

    // @brief the last point of a tile.
    Point2i pEnd;

    // @brief current point in tile.
    Point2i currentP;

    int xMin, xMax;
    int yMin, yMax;

public:

    PointIterator(const Point2i& _pBegin, const Point2i& _pEnd,const Point2i& p);

    Point2i operator*() const;

    PointIterator& operator++();

    bool operator!=(const PointIterator& anotherIt);

};

/*
* @brief Tile representation, and generate points within the tile.
* Check PointIterator for more information.
*/
class Tile
{

protected:

    Point2i pBegin;
    Point2i pEnd;

public:

    Tile(const Point2i& _pBegin, const Point2i& _pEnd);

    virtual PointIterator begin() const=0;
    virtual PointIterator end() const=0;
};

/*
* @brief Basic Tile implemention. Generate points in a square.
*/
class SquareTile : public Tile
{
private:

    PointIterator beginIte;
    PointIterator endIte;
public:

    SquareTile(const Point2i& _pBegin, const Point2i& _pEnd);

    virtual PointIterator begin() const;
    virtual PointIterator end() const;
};

class TileGenerator
{

protected:

    // @brief the resolution of render image.
    Point2i resolution;

    // @brief lock for generateNextTile.
    std::mutex mute;

public:

    TileGenerator(const Point2i& _resolution);

    // @brief generate all tiles. No mutex.
    virtual std::vector<std::shared_ptr<Tile» generateTiles() = 0;

    /*
    * @brief generate next tile.
    * Use mutex to make sure that threads will have different tiles.
    */
    virtual std::optional<std::shared_ptr<Tile» generateNextTile() = 0;

};
```

9.130 /mnt/renderer/Zero/src/ResourceLayer/File/Image.h File Reference

Simple Image representation.

```
#include "CoreLayer/Geometry/Geometry.h"
#include "CoreLayer/ColorSpace/Color.h"
#include <string>
```

9.130.1 Detailed Description

Simple Image representation.

Author

orbitchen

Version

0.1

Date

2022-04-30

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.131 Image.h

Go to the documentation of this file.

```

#pragma once

#include "CoreLayer/Geometry/Geometry.h"
#include "CoreLayer/ColorSpace/Color.h"

#include <string>

// todo: support various data type

class Image
{
    unsigned char *imageRawData;
    Point2i resolution;
    int channels;

public:
    Image();
    ~Image();

    enum class ImageLoadMode
    {
        IMAGE_LOAD_BW,
        IMAGE_LOAD_COLOR
    };
    // todo: support alpha reading
    // todo: do gamma correction
    Image(const std::string &path, ImageLoadMode = ImageLoadMode::IMAGE_LOAD_COLOR);

    // @brief generate one black image with resolution [width,height] and channels.
    Image(const Point2i &resolution, int channels);
    Image(const Point3i &shape);

    friend class ImageManager;

    Point2i getResolution() const;
    int getChannels() const;
    int getWidth() const;
    int getHeight() const;

    void setColorAt(const Point2i &p, const Spectrum &s);
    void setColorAt(const Point2i &p, const RGB3 &rgb);
    RGB3 getRGBColorAt(const Point2i &p);
    Spectrum getSpectrumColorAt(const Point2i &p);

    bool saveTo(const std::string &path);
};
```

9.132 /mnt/renderer/Zero/src/ResourceLayer/File/MeshData.h File Reference

Mesh data for both real time renderer and ray tracing renderer.

```
#include "CoreLayer/Geometry/Geometry.h"
#include <string>
```

9.132.1 Detailed Description

Mesh data for both real time renderer and ray tracing renderer.

Author

orbitchen

Version

0.1

Date

2022-04-30

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.133 MeshData.h

Go to the documentation of this file.

```

#pragma once

#include "CoreLayer/Geometry/Geometry.h"

#include <string>

class MeshData
{
    double *vertexRaw;
    double *normalRaw;
    double *uvRaw;
    double *tangentRaw;
    double *bitangentRaw;
    int *indiceRaw;

    // @brief init MeshData from raw data pointer. MeshData can not be initialized from file path cause one single file may cantain multiple MeshData.
    MeshData(double *_v, double *_n, double *_uv, double *_tan, double *_bi, int *_indice);

public:
    friend class MeshDataManager;

    Point3d getVertexAt(int i) const;
    Normal3d getNormalAt(int i) const;
    Point2d getUvAt(int i) const;
    Vec3d getTangentAt(int i) const;
    Vec3d getBitangentAt(int i) const;

    // @brief get 3 indices for ist triangle mesh. In order.
    Point3i getTriangleIndiceAt(int i) const;

    int getTriangleNum() const;
};
```

9.134 /mnt/renderer/Zero/src/ResourceLayer/ResourceManager.h File Reference

Simple Memory Allocator.

```
#include "ResourceLayer/File/Image.h"
#include "ResourceLayer/File/MeshData.h"
#include <map>
#include <string>
#include <memory>
```

9.134.1 Detailed Description

Simple Memory Allocator.

Author

orbitchen

Version

0.1

Date

2022-04-30

Copyright

NJUMeta (c) 2022 www.njumeta.com

9.135 ResourceManager.h

Go to the documentation of this file.

```

#pragma once

#include "ResourceLayer/File/Image.h"
#include "ResourceLayer/File/MeshData.h"

#include <map>
#include <string>
#include <memory>

template <typename BaseType>
class ResourceManager
{
protected:
    std::map<std::string, std::shared_ptr<BaseType» hash;

public:
    ResourceManager();
};

class ImageManager : public ResourceManager<Image>
{
    ImageManager();
    std::shared_ptr<ImageManager> instance;

public:
    // @brief singleton pattern get.
    static std::shared_ptr<ImageManager> getInstance();

    std::shared_ptr<Image> getImage(const std::string &path, Image::ImageLoadMode mode);
};

class MeshDataManager : public ResourceManager<MeshData>
{
    MeshDataManager();
    std::shared_ptr<MeshDataManager> instance;

public:
    // @brief singleton pattern get.
    static std::shared_ptr<MeshDataManager> getInstance();

    std::vector<std::shared_ptr<MeshData» getMeshData(const std::string &path);
};
```