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.

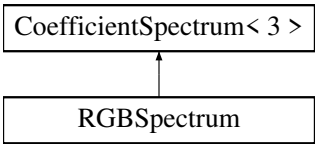
The documentation for this class was generated from the following file:

- /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:

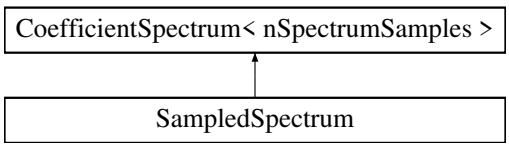
- /mnt/renderer/Zero/src/CoreLayer/ColorSpace/Color.h

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

```
1
11 #pragma once
12 #include "rapidjson/document.h"
13 #include <vector>
14 #include <string>
15 #include <unordered_map>
16 #include <iostream>
17 #include <memory>
18
19 struct Value {
20     Value() = default;
21     virtual ~Value() = default;
22
23     virtual void print() const = 0;
24 };
25
26 class Attribute {
27 public:
28     Attribute() : m_type(EValueType::EInvalid), m_value(nullptr) { }
29
30     Attribute(const rapidjson::Document &document);
31
32     Attribute(const std::string &key, bool b);
33
34     Attribute(const std::string &key, int i);
35
36     Attribute(const std::string &key, double d);
37
38     Attribute(const std::string &key, const std::string &str);
39
40     Attribute(const std::string &key,
41               const rapidjson::GenericValue<rapidjson::UTF8<»::ConstArray &array);
42
43     Attribute(const std::string &key,
44               const rapidjson::GenericValue<rapidjson::UTF8<»::ConstObject &object);
45
46
47     void print() const {
48         std::cout « "Key = " « m_key « ", ";
49         m_value->print();
50     }
51
52     Attribute operator[](const std::string &key) const;
53
54     Attribute operator[](int i) const;
55
56     bool getBool() const;
57
58     int getInt() const;
59
60     double getDouble() const;
61
62     std::string getString() const;
63
64 private:
65
66     enum class EValueType {
67         EInvalid = 0,
```

```
68          EBool,
69          EInt,
70          EDouble,
71          EString,
72          EArray,
73          EObject
74      } m_type;
75
76      std::string m_key;
77
78      std::shared_ptr<Value> m_value;
79
80  };
81
82
83  struct BoolValue : public Value {
84      BoolValue() = default;
85
86      BoolValue(bool b) : data(b) { };
87
88      virtual ~BoolValue() = default;
89
90      virtual void print() const override {
91          std::cout « "BoolValue : " « (data ? "true" : "false") « std::endl;
92      }
93
94      bool data;
95
96  };
97
98  struct IntValue : public Value {
99      IntValue() = default;
100
101      IntValue(int i) : data(i) { }
102
103      virtual ~IntValue() = default;
104
105      int data;
106
107      virtual void print() const override {
108          std::cout « "IntValue : " « data « std::endl;
109      }
110  };
111
112
113  struct DoubleValue : public Value {
114      DoubleValue() = default;
115
116      DoubleValue(double d) : data(d) { }
117
118      virtual ~DoubleValue() = default;
119
120      double data;
121
122      virtual void print() const override {
123          std::cout « "DoubleValue : " « data « std::endl;
124      }
125  };
126
127
128  struct ArrayValue : public Value {
129      ArrayValue() = default;
130
131      virtual ~ArrayValue() {
132          for (int i = 0; i < data.size(); ++i) {
133              delete data[i];
134          }
135      }
136
137      std::vector<Attribute *> data;
138
139      virtual void print() const override {
140          std::cout « "ArrayValue : \n";
141          for (int i = 0; i < data.size(); ++i) {
142              std::cout « "\t";
143              data[i]->print();
144          }
145      }
146  };
147
148  struct StringValue : public Value {
149      StringValue() = default;
150
151      StringValue(const std::string &str) : data(str) { }
152
153      virtual ~StringValue() = default;
154
```

```
155     virtual void print() const override {
156         std::cout « "StringValue : " « data « std::endl;
157     }
158 
159     std::string data;
160 
161 };
162 
163 struct ObjectValue : public Value {
164     ObjectValue() = default;
165 
166     virtual ~ObjectValue() {
167         for (auto itr = data.begin(); itr != data.end(); ++itr) {
168             delete itr->second;
169         }
170     }
171 
172     virtual void print() const override {
173         std::cout « "ObjectValue : {\n";
174         for (auto itr = data.begin(); itr!=data.end(); ++itr) {
175             std::cout « "\t";
176             itr->second->print();
177         }
178         std::cout «"}"«std::endl;
179     }
180 
181     std::unordered_map<std::string, Attribute *> data;
182 };
183 
184 
185 inline Attribute::Attribute(const rapidjson::Document &document) {
186     m_type = EValueType::EObject;
187     m_key = "root";
188     m_value = std::make_shared<ObjectValue>();
189 
190     ObjectValue* object_value = static_cast<ObjectValue *>(m_value.get());
191 
192     for (auto itr = document.MemberBegin(); itr != document.MemberEnd(); ++itr) {
193         Attribute *attribute = nullptr;
194         const std::string &member_key = itr->name.GetString();
195         const auto &member_value = itr->value;
196         if (member_value.IsBool()) {
197             attribute = new Attribute(member_key, member_value.GetBool());
198         } else if (member_value.IsInt()) {
199             attribute = new Attribute(member_key, member_value.GetInt());
200         } else if (member_value.IsDouble()) {
201             attribute = new Attribute(member_key, member_value.GetDouble());
202         } else if (member_value.IsString()) {
203             attribute = new Attribute(member_key, std::string(member_value.GetString()));
204         } else if (member_value.IsArray()) {
205             attribute = new Attribute(member_key, member_value.GetArray());
206         } else if (member_value.IsObject()) {
207             attribute = new Attribute(member_key, member_value.GetObject());
208         }
209         if (attribute == nullptr){
210             std::cout « "Error type\n";
211             std::exit(1);
212         }
213         object_value->data[member_key] = attribute;
214     }
215 }
216 
217 inline Attribute::Attribute(const std::string &key, bool b) :
218     m_type(EValueType::EBool),
219     m_key(key),
220     m_value(new BoolValue {b}) { }
221 
222 
223 inline Attribute::Attribute(const std::string &key, int i) :
224     m_type(EValueType::EInt),
225     m_key(key),
226     m_value(new IntValue {i}) { }
227 
228 inline Attribute::Attribute(const std::string &key, double d) :
229     m_type(EValueType::EDouble),
230     m_key(key),
231     m_value(new DoubleValue {d}) { }
232 
233 inline Attribute::Attribute(const std::string &key, const std::string &str) :
234     m_type(EValueType::EString),
235     m_key(key),
236     m_value(new StringValue {str}) { }
237 
238 
239 inline Attribute::Attribute(
240     const std::string &key,
241     const rapidjson::GenericValue<rapidjson::UTF8<»::ConstArray &array) :
```

```
242     m_type(EValueType::EArray),
243     m_key(key) {
244 
245     m_value = std::make_shared<ArrayValue>();
246     ArrayValue *array_value = static_cast<ArrayValue *>(m_value.get());
247     for (auto itr = array.Begin(); itr != array.End(); ++itr) {
248         Attribute *attribute = nullptr;
249         if (itr->IsBool()) {
250             attribute = new Attribute("", itr->GetBool());
251         } else if (itr->IsInt()) {
252             attribute = new Attribute("", itr->GetInt());
253         } else if (itr->IsDouble()) {
254             attribute = new Attribute("", itr->GetDouble());
255         } else if (itr->IsString()) {
256             attribute = new Attribute("", std::string(itr->GetString()));
257         } else if (itr->IsArray()) {
258             attribute = new Attribute("", itr->GetArray());
259         } else if (itr->IsObject()) {
260             attribute = new Attribute("", itr->GetObject());
261         }
262         if (attribute == nullptr) {
263             std::cout « "Error Type\n";
264             std::exit(1);
265         }
266         array_value->data.emplace_back(attribute);
267     }
268 }
269 
270 inline Attribute::Attribute(
271     const std::string &key,
272     const rapidjson::GenericValue<rapidjson::UTF8<»::ConstObject &object) :
273     m_type(EValueType::EObject),
274     m_key(key) {
275 
276     m_value = std::make_shared<ObjectValue>();
277     ObjectValue *object_value = static_cast<ObjectValue *>(m_value.get());
278     for (auto itr = object.MemberBegin(); itr != object.MemberEnd(); ++itr) {
279         Attribute *attribute = nullptr;
280         const std::string &member_key = itr->name.GetString();
281         const auto &member_value = itr->value;
282         if (member_value.IsBool()) {
283             attribute = new Attribute(member_key, member_value.GetBool());
284         } else if (member_value.IsInt()) {
285             attribute = new Attribute(member_key, member_value.GetInt());
286         } else if (member_value.IsDouble()) {
287             attribute = new Attribute(member_key, member_value.GetDouble());
288         } else if (member_value.IsString()) {
289             attribute = new Attribute(member_key, std::string(member_value.GetString()));
290         } else if (member_value.IsArray()) {
291             attribute = new Attribute(member_key, member_value.GetArray());
292         } else if (member_value.IsObject()) {
293             attribute = new Attribute(member_key, member_value.GetObject());
294         }
295         if (attribute == nullptr){
296             std::cout « "Error type\n";
297             std::exit(1);
298         }
299         object_value->data[member_key] = attribute;
300     }
301 }
302 
303 inline Attribute Attribute::operator[](const std::string &key) const {
304     assert(m_type == EValueType::EObject);
305     ObjectValue *object_value = static_cast<ObjectValue *>(m_value.get());
306     const auto attributes = object_value->data;
307     if (auto itr = attributes.find(key); itr == attributes.end()) {
308         std::cout « "No such a key \"" « key «  "\"\n";
309         std::exit(1);
310     } else {
311         return *(itr->second);
312     }
313 }
314 
315 inline Attribute Attribute::operator[](int i) const {
316     assert(m_type == EValueType::EArray);
317     ArrayValue *array_value = static_cast<ArrayValue *>(m_value.get());
318     const auto attribute_list = array_value->data;
319     assert(i < attribute_list.size());
320     return *attribute_list[i];
321 }
322 
323 inline bool Attribute::getBool() const {
324     assert(m_type == EValueType::EBool);
325     BoolValue *bool_value = static_cast<BoolValue *>(m_value.get());
326     return bool_value->data;
327 }
328 
```

```
329 inline int Attribute::getInt() const {
330     assert(m_type == EValueType::EInt);
331     IntValue *int_value = static_cast<IntValue *>(m_value.get());
332     return int_value->data;
333 }
334
335 inline double Attribute::getDouble() const {
336     assert(m_type == EValueType::EDouble);
337     DoubleValue *double_value = static_cast<DoubleValue *>(m_value.get());
338     return double_value->data;
339 }
340
341 inline std::string Attribute::getString() const {
342     assert(m_type == EValueType::EString);
343     StringValue *string_value = static_cast<StringValue *>(m_value.get());
344     return string_value->data;
345 }
346
```

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

9.3 Random.h

Go to the documentation of this file.

```
1
13 #pragma once
14 #include <random>
15 #include "pcg/pcg_random.hpp"
16 #include "CoreLayer/Math/Common.h"
17
22 class RandomNumberGenerator {
23     std::uniform_real_distribution<> dist;
24     pcg_extras::seed_seq_from<std::random_device> seedSource;
25     pcg32 rng;
26 public:
27     RandomNumberGenerator():dist(0, ONEMINUSEPSILON), rng(seedSource) { }
28
30     double operator()() {
31         return dist(rng);
32     }
33
35     int operator()(int begin, int end) {
36         int sample = begin + (end - begin) * dist(rng);
37         return std::min(end - 1, sample);
38     }
39
40 };
```

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

9.5 Color.h

Go to the documentation of this file.

```
1
12 #pragma once
13
14 #include <vector>
15 #include <cmath>
16 #include <cfloat>
17
18 class RGB3;
19 class XYZ3;
20 class SampledSpectrum;
21
22 // TODO: should be defined by cmake marco.
23 using Spectrum = SampledSpectrum;
24
25 //using Spectrum = RGB3;
26
27 static const double sampledLambdaStart = 400.0;
28 static const double sampledLambdaEnd = 700.0;
29
31 static const int nSpectrumSamples = 60;
32
34 double mathClamp(double source, double low=0.0, double high=DBL_MAX);
35
37 double mathLerp(double ratio, double source0, double source1);
38
40 enum class SpectrumType { REFLECTANCE, ILLUMINANT };
41
```

```
42 class RGB3
43 {
44     double rgbData[3];
45 
46 public:
47     RGB3();
48 
49     RGB3(double r, double g, double b);
50 
51     // @brief initialize all rgb value as val.
52     RGB3(double val);
53 
54     double operator[](int i) const;
55     double &operator[](int i);
56 
57     RGB3 operator+(const RGB3 &rgb);
58     RGB3 operator-(const RGB3 &rgb);
59     RGB3 operator*(const RGB3 &rgb);
60     RGB3 operator/(const RGB3 &rgb);
61 
62     RGB3 &operator+=(const RGB3 &rgb);
63     RGB3 &operator-=(const RGB3 &rgb);
64     RGB3 &operator*=(const RGB3 &rgb);
65     RGB3 &operator/=(const RGB3 &rgb);
66 
67     RGB3 operator*(double v);
68     RGB3 operator/(double v);
69 
70     RGB3 &operator*=(double v);
71     RGB3 &operator/=(double v);
72 
73     friend RGB3 operator*(double v, const RGB3 &rgb);
74 
75     XYZ3 toXYZ3() const;
76 
77     // @brief convert RGB3 to SampledSpectrum.
78     Spectrum toSpectrum(SpectrumType type=SpectrumType::REFLECTANCE) const;
79 };
80 
81 class XYZ3
82 {
83     double xyzData[3];
84 
85 public:
86     XYZ3();
87 
88     XYZ3(double x, double y, double z);
89 
90     // @brief initialize all xyz value as val.
91     XYZ3(double val);
92 
93     double operator[](int i) const;
94     double &operator[](int i);
95 
96     XYZ3 operator+(const XYZ3 &xyz);
97     XYZ3 operator-(const XYZ3 &xyz);
98     XYZ3 operator*(const XYZ3 &xyz);
99     XYZ3 operator/(const XYZ3 &xyz);
100 
101     XYZ3 &operator+=(const XYZ3 &xyz);
102     XYZ3 &operator-=(const XYZ3 &xyz);
103     XYZ3 &operator*=(const XYZ3 &xyz);
104     XYZ3 &operator/=(const XYZ3 &xyz);
105 
106     XYZ3 operator*(double v);
107     XYZ3 operator/(double v);
108 
109     XYZ3 &operator*=(double v);
110     XYZ3 &operator/=(double v);
111 
112     friend XYZ3 operator*(double v, const XYZ3 &xyz);
113 
114     RGB3 toRGB3() const;
115 };
116 
118 template <int nSamples>
119 class CoefficientSpectrum
120 {
121 protected:
122     double coefficients[nSamples];
123 
124 public:
126     CoefficientSpectrum() {
127         for (int i = 0; i < nSamples; i++) {
128             coefficients[i] = 0.0;
129         }
130     }
```

```
131
133     CoefficientSpectrum(double val) {
134         for (int i = 0; i < nSamples; i++) {
135             coefficients[i] = val;
136         }
137     }
138
140     double operator[](int i) const {
141         return coefficients[i];
142     }
143
145     double &operator[](int i) {
146         return coefficients[i];
147     }
148
149     CoefficientSpectrum operator+(const CoefficientSpectrum&s) const {
150         CoefficientSpectrum retVal = *this;
151         for (int i = 0; i < nSamples; i++) {
152             retVal.coefficients[i] += s.coefficients[i];
153         }
154         return retVal;
155     }
156
157     CoefficientSpectrum operator-(const CoefficientSpectrum&s) const {
158         CoefficientSpectrum retVal = *this;
159         for (int i = 0; i < nSamples; i++) {
160             retVal.coefficients[i] -= s.coefficients[i];
161         }
162         return retVal;
163     }
164
165     CoefficientSpectrum operator*(const CoefficientSpectrum&s) const {
166         CoefficientSpectrum retVal = *this;
167         for (int i = 0; i < nSamples; i++) {
168             retVal.coefficients[i] *= s.coefficients[i];
169         }
170         return retVal;
171     }
172
174     CoefficientSpectrum operator/(const CoefficientSpectrum&s) const {
175         CoefficientSpectrum retVal = *this;
176         for (int i = 0; i < nSamples; i++) {
177             retVal.coefficients[i] /= s.coefficients[i];    // NaN
178         }
179         return retVal;
180     }
181
182     CoefficientSpectrum& operator+=(const CoefficientSpectrum&s) {
183         for (int i = 0; i < nSamples; i++) {
184             this->coefficients[i] += s.coefficients[i];
185         }
186         return *this;
187     }
188
189     CoefficientSpectrum& operator-=(const CoefficientSpectrum&s) {
190         for (int i = 0; i < nSamples; i++) {
191             this->coefficients[i] -= s.coefficients[i];
192         }
193         return *this;
194     }
195
196     CoefficientSpectrum& operator*=(const CoefficientSpectrum&s) {
197         for (int i = 0; i < nSamples; i++) {
198             this->coefficients[i] *= s.coefficients[i];
199         }
200         return *this;
201     }
202
204     CoefficientSpectrum& operator/=(const CoefficientSpectrum&s) {
205         for (int i = 0; i < nSamples; i++) {
206             this->coefficients[i] /= s.coefficients[i]; // NaN
207         }
208         return *this;
209     }
210
211     CoefficientSpectrum operator*(double v) const {
212         CoefficientSpectrum retVal = *this;
213         for (int i = 0; i < nSamples; i++) {
214             retVal.coefficients[i] *= v;
215         }
216         return retVal;
217     }
218
220     CoefficientSpectrum operator/(double v) const {
221         CoefficientSpectrum retVal = *this;
222         for (int i = 0; i < nSamples; i++) {
223             retVal.coefficients[i] /= v;    // NaN
```

```
224         }
225         return retVal;
226     }
227 
228     CoefficientSpectrum& operator*=(double v) {
229         for (int i = 0; i < nSamples; i++) {
230             this->coefficients[i] *= v;
231         }
232         return *this;
233     }
234 
236     CoefficientSpectrum& operator/=(double v) {
237         for (int i = 0; i < nSamples; i++) {
238             this->coefficients[i] /= v; // NaN
239         }
240         return *this;
241     }
242 
243     friend CoefficientSpectrum operator*(double v, const CoefficientSpectrum& s) {
244         return s * v;
245     }
246 
248     friend CoefficientSpectrum sqrt(const CoefficientSpectrum& s) {
249         CoefficientSpectrum ret;
250         for (int i = 0; i < nSamples; i++)
251             ret[i] = std::sqrt(s[i]);
252         return ret;
253     }
254 
255     friend CoefficientSpectrum pow(const CoefficientSpectrum&s, double e) {
256         CoefficientSpectrum ret;
257         for (int i = 0; i < nSamples; i++)
258             ret[i] = std::pow(s[i], e);
259         return ret;
260     }
261 
262     friend CoefficientSpectrum exp(const CoefficientSpectrum&s) {
263         CoefficientSpectrum ret;
264         for (int i = 0; i < nSamples; i++)
265             ret[i] = std::exp(s[i]);
266         return ret;
267     }
268 
269     bool isBlack() const {
270         for (int i = 0; i < nSamples; i++) {
271             if (coefficients[i] != 0.0)
272                 return false;
273         }
274     }
275 
276     bool hasNaN() const {
277         for (int i = 0; i <nSamples; i++)
278             if (std::isnan(coefficients[i]))
279                 return true;
280         return false;
281     }
282 
283     inline CoefficientSpectrum clamp(double low = 0.0, double high = DBL_MAX) const {
284         CoefficientSpectrum retVal;
285         for (int i = 0; i < nSamples; i++) {
286             retVal[i] = mathClamp(coefficients[i], low, high);
287         }
288         return retVal;
289     }
290 
291     double sum() const {
292         double sum = 0;
293         for (int i = 0; i < nSamples; i++) {
294             sum += coefficients[i];
295         }
296         return sum;
297     }
298 
299     double average() const {
300         return sum() / nSamples;
301     }
302 
304     virtual XYZ3 toXYZ3() const {
305         //DEBUG this function should never be called.
306         return XYZ3(0.0);
307     }
308 };
309 
311 struct SpectrumSample
312 {
313     double lambda;
314     double value;
```

```
315
316     SpectrumSample(double _lambda, double _value) {
317         lambda = _lambda;
318         value = _value;
319     }
320
322     bool operator>(const SpectrumSample& s) const {
323         return lambda > s.lambda;
324     }
325
327     bool operator<(const SpectrumSample& s) const {
328         return lambda < s.lambda;
329     }
330 };
331
333 class SampledSpectrum
334     : public CoefficientSpectrum<nSpectrumSamples>
335 {
336     // these spectrums should be calculated at compile time.
337     static SampledSpectrum X;
338     static SampledSpectrum Y;
339     static SampledSpectrum Z;
340
341     static SampledSpectrum rgbRefl2SpectWhite;
342     static SampledSpectrum rgbRefl2SpectCyan;
343     static SampledSpectrum rgbRefl2SpectMagenta;
344     static SampledSpectrum rgbRefl2SpectYellow;
345     static SampledSpectrum rgbRefl2SpectRed;
346     static SampledSpectrum rgbRefl2SpectGreen;
347     static SampledSpectrum rgbRefl2SpectBlue;
348
349     static SampledSpectrum rgbIllum2SpectWhite;
350     static SampledSpectrum rgbIllum2SpectCyan;
351     static SampledSpectrum rgbIllum2SpectMagenta;
352     static SampledSpectrum rgbIllum2SpectYellow;
353     static SampledSpectrum rgbIllum2SpectRed;
354     static SampledSpectrum rgbIllum2SpectGreen;
355     static SampledSpectrum rgbIllum2SpectBlue;
356
357 public:
358     friend class RGB3;
359     friend class XYZ3;
360
362     static void init();
363
364     SampledSpectrum();
365
366     SampledSpectrum(double val);
367
368     SampledSpectrum(const CoefficientSpectrum& s);
369
371     static SampledSpectrum fromSampled(std::vector<SpectrumSample> v);
372
373     virtual XYZ3 toXYZ3() const override;
374 };
375
377 class RGBSpectrum : public CoefficientSpectrum<3>
378 {
379     // TODO RGBSpectrum
380 };
```

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

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.

```
1
12 #pragma once
13
14 #include "Geometry.h"
15 #include "CoreLayer/Ray/Ray.h"
16 #include <optional>
17 #include <algorithm>
18 #include <limits>
19
25 template <typename BaseType>
26 class BoundingBox3
27 {
28 public:
29     TPoint3<BaseType> pMin;
30     TPoint3<BaseType> pMax;
31
32     BoundingBox3() {
33         pMin[0] = pMin[1] = pMin[2] = std::numeric_limits<BaseType>::max();
34         pMax[0] = pMax[1] = pMax[2] = std::numeric_limits<BaseType>::min();
35     };
36
37     BoundingBox3(TPoint3<BaseType> _p)
38     {
39         pMin = pMax = _p;
40     }
41     BoundingBox3(TPoint3<BaseType> _pMin, TPoint3<BaseType> _pMax)
42     {
43         pMin = _pMin;
44         pMax = _pMax;
45     }
46
51     std::optional<Point2d> Intersection(const Ray& r) {
52         double t0 = 0, t1 = DBL_MAX;
53         for (int i = 0; i < 3; i++) {
54             double inv = 1.0 / r.direction[i];
55             double tNear = (pMin[i] - r.origin[i]) * inv;
56             double tFar = (pMax[i] - r.origin[i]) * inv;
57             if (tNear > tFar) std::swap(tNear, tFar);
58             if (t0 < tNear) t0 = tNear;
59             if (t1 > tFar) t1 = tFar;
60         }
61         if (t0 > t1) return std::nullopt;
62         return std::make_optional(Point2d(t0, t1));
63     }
64
69     double SurfaceArea() {
70         double x = pMax[0] - pMin[0];
71         double y = pMax[1] - pMin[1];
72         double z = pMax[2] - pMin[2];
73         return 2 * (x * y + x * z + y * z);
74     }
75 };
76
81 template <typename BaseType>
82 static BoundingBox3<BaseType> BoundingBoxUnion(const BoundingBox3<BaseType>& b1,
83                                                const BoundingBox3<BaseType>& b2)
84 {
85     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]));
86     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]));
87     return BoundingBox3(_pMin, _pMax);
88 }
94 template <typename BaseType>
95 static BoundingBox3<BaseType> BoundingBoxUnionIntersect(const BoundingBox3<BaseType>& b1,
96                                                         const BoundingBox3<BaseType>& b2)
97 {
98     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]));
99     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]));
100     return BoundingBox3(_pMin, _pMax);
101 }
102
103 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

9.11 CoordConvertor.h

Go to the documentation of this file.

```
1
13 #include "Geometry.h"
14
18 class CoordConvertor
19 {
20 public:
25     static Point3d cartesian2Spherical(const Point2d &p);
26
28     static Vec3d cartesian2SphericalVec(const Point2d &p);
29
30     static Vec3d world2TBN(const Vec3d &v, const Vec3d &t, const Vec3d &b, const Vec3d &n);
31
32     static Vec3d TBN2World(const Vec3d &v, const Vec3d &t, const Vec3d &b, const Vec3d &n);
33 };
```

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

9.13 Frame.h

Go to the documentation of this file.

```
1 //
2 // Created by yjp on 2022/6/6.
3 //
4
16 #pragma once
17
18 #include "Geometry.h"
19 #include "CoreLayer/Math/Common.h"
20
21
22 static void coordinateSystem(const Normal3d &a, Vec3d  &b, Vec3d  &c) {
23     if (std::abs(a.x) > std::abs(a.y)) {
24         float invLen = 1.0f / std::sqrt(a.x * a.x + a.z * a.z);
25         c = Vec3d (a.z * invLen, 0.0f, -a.x * invLen);
26     } else {
27         float invLen = 1.0f / std::sqrt(a.y * a.y + a.z * a.z);
28         c = Vec3d(0.0f, a.z * invLen, -a.y * invLen);
29     }
30     Vec3d  _a(a.x,a.y,a.z);
31     b = cross(_a,c);
32 }
33
34 struct Frame {
35     Vec3d s, t;
36     Normal3d  n;
37
39     Frame() { }
40
42     Frame(const Vec3d  &s, const Vec3d &t, const Normal3d  &n)
43             : s(s), t(t), n(n) { }
44
46     Frame(const Vec3d &x, const Vec3d &y, const Vec3d &z)
47             : s(x), t(y), n(z.x,z.y,z.z) { }
48
50     Frame(const Normal3d n) : n(n){
51         coordinateSystem(n, s, t);
52     }
53
55     Vec3d toLocal(const Vec3d &v) const {
56         return Vec3d(
57                 dot(v,s), dot(v,t), dot(v,n)
58         );
59     }
60
62     Vec3d toWorld(const Vec3d &v) const {
63         return s * v.x + t * v.y   + n * v.z ;
64     }
65
68     static float cosTheta(const Vec3d &v) {
69         return v.z;
70     }
71
74     static float sinTheta(const Vec3d &v) {
75         float temp = sinTheta2(v);
76         if (temp <= 0.0f)
77             return 0.0f;
78         return std::sqrt(temp);
79     }
80
83     static float tanTheta(const Vec3d &v) {
84         float temp = 1 - v.z*v.z;
85         if (temp <= 0.0f)
86             return 0.0f;
87         return std::sqrt(temp) / v.z;
88     }
89
92     static float sinTheta2(const Vec3d &v) {
93         return 1.0f - v.z * v.z;
94     }
95
98     static float sinPhi(const Vec3d &v) {
99         float sinTheta = Frame::sinTheta(v);
100         if (sinTheta == 0.0f)
101             return 1.0f;
102         return clamp(v.y / sinTheta, -1.0f, 1.0f);
103     }
104
107     static float cosPhi(const Vec3d &v) {
108         float sinTheta = Frame::sinTheta(v);
109         if (sinTheta == 0.0f)
110             return 1.0f;
111         return clamp(v.x / sinTheta, -1.0f, 1.0f);
```

```
112     }
113
117     static float sinPhi2(const Vec3d &v) {
118         return clamp(v.y * v.y / sinTheta2(v), 0.0f, 1.0f);
119     }
120
124     static float cosPhi2(const Vec3d &v) {
125         return clamp(v.x * v.x / sinTheta2(v), 0.0f, 1.0f);
126     }
127
129     bool operator==(const Frame &frame) const {
130         return frame.s == s && frame.t == t && frame.n == n;
131     }
132
134     bool operator!=(const Frame &frame) const {
135         return !operator==(frame);
136     }
137
138
139 };
```

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.

```
1
12 #pragma once
13
14 #include "normal.h"
15 #include "point.h"
16 #include "vector.h"
17
19
20 // vector.h
21 template <typename T>
22 struct TVector2;
23 template <typename T>
24 struct TVector3;
25 using Vec2f = TVector2<float>;
26 using Vec2d = TVector2<double>;
27 using Vec2i = TVector2<int>;
28 using Vec3f = TVector3<float>;
29 using Vec3d = TVector3<double>;
30 using Vec3i = TVector3<int>;
31 // point.h
32 template <typename T>
33 struct TPoint2;
34 template <typename T>
35 struct TPoint3;
36 using Point2f = TPoint2<float>;
37 using Point2d = TPoint2<double>;
38 using Point2i = TPoint2<int>;
39 using Point3f = TPoint3<float>;
40 using Point3d = TPoint3<double>;
41 using Point3i = TPoint3<int>;
42 // normal.h
43 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.

```
1
12 #pragma once
13
14 #include "Geometry.h"
15
16 #include "Eigen/Dense"
17
18  /*
19   * @brief Angle type. Providing convenient angle transformation between deg and rad.
20   */
21 class Angle
22 {
23     double deg;
24     double rad;
25
26 public:
27     enum class AngleType
28     {
29         ANGLE_DEG,
30         ANGLE_RAD
31     };
32     /*
33      * @brief init an angle.
34      * @param v value of angle, deg or rad.
35      * @param type the type of v.
36      */
37     Angle(double v, AngleType type);
38
39     double getDeg() const;
40
```

```
41     double getRad() const;
42 };
43
44 enum class EulerType
45 {
46     EULER_XYZ,
47     EULER_ZYX
48 };
49
50 // @brief a simple wrap of Eigen Matrix 4x4.
51 class Matrix4x4
52 {
53 private:
54
55     // @brief Eigen data. inaccessible from outside.
56     Eigen::Matrix4d matrix=Eigen::Matrix4d::Identity();
57
58     // @brief init from Eigen data. inaccessible from outside.
59     Matrix4x4(const Eigen::Matrix4d& _matrix);
60
61 public:
62
63     Matrix4x4();
64
65     Matrix4x4 operator*(const Matrix4x4& mat) const;
66
67     Vec3d operator*(const Vec3d& v) const;
68     Point3d operator*(const Point3d& p) const;
69     Normal3d operator*(const Normal3d& n) const;
70
71     // model
72     static Matrix4x4 translate(double x, double y, double z);
73     static Matrix4x4 scale(double x, double y, double z);
74     static Matrix4x4 scale(double ratio);
75     static Matrix4x4 rotateEuler(const Angle& x, const Angle& y, const Angle& z, EulerType type =
       EulerType::EULER_XYZ);
76     static Matrix4x4 rotateQuaternion(double w, double x, double y, double z);
77     static Matrix4x4 rotateAxis(const Vec3d& axis, const Angle& angle);
78
79     // view & projection
80     static Matrix4x4 lookAt(const Point3d& lookFrom, const Vec3d& vecLookAt, const Vec3d& up);
81     static Matrix4x4 orthographic(double left,double right,double up,double down,double near,double far);
82     static Matrix4x4 perspective(const Angle& fov,double aspect, double near, double far);
83
84     Matrix4x4 inverse();
85     Matrix4x4 transpose();
86
87     friend void printMatrix(const Matrix4x4 &mat);
88 };
89
90 /*
91  * @brief Encapsulated transform matrix. By default, it will be initialized as identity matrix.
92  */
93 class TransformMatrix3D
94 {
95     // @brief the matrix that applies rotate, scale and translate.
96     Matrix4x4 matrixAll;
97
98     Matrix4x4 matrixRotate;
99     Matrix4x4 matrixScale;
100     Matrix4x4 matrixTranslate;
101
102     // @brief true iff matrixAll!=matrixTranslate*matrixScale*matrixRotate.
103     bool dirty;
104
105     void update();
106
107 public:
108     TransformMatrix3D();
109
110     void setTranslate(double x, double y, double z);
111
112     void setScale(double x, double y, double z);
113     void setScale(double ratio);
114
115     void setRotateEuler(const Angle& x, const Angle& y, const Angle& z, EulerType type =
       EulerType::EULER_XYZ);
116
117     void setRotateQuaternion(double w, double x, double y, double z);
118
119     // @brief Rotate by axis. Counterclockwise rotate.
120     void setRotateAxis(const Vec3d& axis, const Angle& angle);
121
122     Vec3d operator*(const Vec3d &v);
123     Point3d operator*(const Point3d &p);
124     Normal3d operator*(const Normal3d &n);
125 };
```

```
126
127 // 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.

```
1
13 #pragma once
14 #include "vector.h"
15
16 struct Normal3d : public TVector3<double> {
17
18     Normal3d() { }
19
20     Normal3d (const TVector3<double> &v) : TVector3<double>(normalize(v)) { }
21
22     Normal3d (double _x, double _y, double _z) : Normal3d(TVector3<double> {_x, _y, _z})  { }
23
24
25 };
```

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

9.22 point.h

Go to the documentation of this file.

```
1
12 #pragma once
13 #include "vector.h"
14
15 template <typename T>
16 struct TPoint2 {
17     /*--- data field ---*/
18     T x, y;
19
20     /*--- constructor ---*/
21     TPoint2() { }
22
23     TPoint2(T _x, T _y) : x(_x), y(_y) { }
24
25     explicit TPoint2(T t) : x(t), y(t) { }
26
27     /*--- operator overloading ---*/
28     TPoint2 operator+(const TVector2<T> &rhs) const {
29         return TPoint2(x+rhs.x, y+rhs.y);
30     }
31
32     TPoint2 operator+(const TPoint2<T>& rhs) const {
33         return TPoint2(x + rhs.x, y + rhs.y);
34     }
35
36     TPoint2& operator+=(const TVector2<T> &rhs) {
37         x += rhs.x, y += rhs.y;
38         return *this;
39     }
40
41     TPoint2 operator-(const TVector2<T> &rhs) const {
42         return TPoint2(x-rhs.x, y-rhs.y);
43     }
44
45     TVector2<T> operator-(const TPoint2<T>& rhs) const {
46         return TVector2(x - rhs.x, y - rhs.y);
47     }
48
49     TPoint2& operator-=(const TVector2<T> &rhs) {
50         x -= rhs.x, y -= rhs.y;
51         return *this;
52     }
53
54     TPoint2 operator*(T t) const {
55         return TPoint2(x*t, y*t);
56     }
57
58     TPoint2& operator*=(T t) {
59         x *= t, y *= t;
```

```
60          return *this;
61      }
62
63      TPoint2 operator/(T t) const {
64          assert(t != 0);
65          T recip = (T) 1 / t;
66          return TPoint2(x*recip, y*recip);
67      }
68
69      TPoint2& operator/=(T t) {
70          assert(t != 0);
71          T recip = (T) 1 / t;
72          x *= recip, y *= recip;
73          return *this;
74      }
75
76      TPoint2 operator-() const {
77          return TPoint2(-x, -y);
78      }
79
80      bool operator==(const TPoint2 &rhs) const {
81          return x==rhs.x && y==rhs.y;
82      }
83
84      bool operator!=(const TPoint2 &rhs) const {
85          return x!=rhs.x || y!=rhs.y;
86      }
87
88      T operator[] (int i) const {
89          return (&x)[i];
90      }
91
92      T& operator[] (int i) {
93          return (&x)[i];
94      }
95
96      bool isZero() const {
97          return x==0 && y==0;
98      }
99  };
100
101 template <typename T>
102 std::ostream& operator«(std::ostream &os, const TPoint2<T> &p) {
103     os « p.x « " " « p.y;
104     return os;
105 }
106
107 template <typename T>
108 TPoint2<T> operator*(T t, const TPoint2<T> &v) {
109     return TPoint2(t*v.x, t*v.y);
110 }
111
112 template<>
113 inline TPoint2<int> TPoint2<int>::operator/(int i) const {
114     assert(i!=0);
115     return TPoint2<int>(x/i, y/i);
116 }
117
118 template<>
119 inline TPoint2<int>& TPoint2<int>::operator/=(int i) {
120     assert(i!=0);
121     x/=i, y/=i;
122     return *this;
123 }
124
125
126 template <typename T>
127 struct TPoint3 {
128     /*--- data field ---*/
129     T x, y, z;
130
131     /*--- constructor ---*/
132     TPoint3() { }
133
134     TPoint3(T _x, T _y, T _z) : x(_x), y(_y), z(_z) { }
135
136     explicit TPoint3(T t) : x(t), y(t), z(t) { }
137
138     /*--- operator overloading ---*/
139     TPoint3 operator+(const TVector3<T> &rhs) const {
140         return TPoint3(x+rhs.x, y+rhs.y, z+rhs.z);
141     }
142
143     TPoint3 operator+(const TPoint3<T>& rhs) const {
144         return TPoint3(x + rhs.x, y + rhs.y, z + rhs.z);
145     }
146
```

```
147     TPoint3& operator+=(const TVector3<T> &rhs) {
148         x += rhs.x, y += rhs.y, z += rhs.z;
149         return *this;
150     }
151 
152     TPoint3 operator-(const TVector3<T> &rhs) const {
153         return TPoint3(x-rhs.x, y-rhs.y, z-rhs.z);
154     }
155 
156     TVector3<T> operator-(const TPoint3<T> &rhs) const {
157         return TVector3<T>(x-rhs.x, y-rhs.y, z-rhs.z);
158     }
159 
160     TPoint3& operator-=(const TVector3<T> &rhs) {
161         x -= rhs.x, y -= rhs.y, z -= rhs.z;
162         return *this;
163     }
164 
165     TPoint3 operator*(T t) const {
166         return TPoint3(x*t, y*t, z*t);
167     }
168 
169     TPoint3& operator*=(T t) {
170         x *= t, y *= t, z *= t;
171         return *this;
172     }
173 
174     TPoint3 operator/(T t) const {
175         assert(t != 0);
176         T recip = (T) 1 / t;
177         return TPoint3(x*recip, y*recip, z*recip);
178     }
179 
180     TPoint3& operator/=(T t) {
181         assert(t != 0);
182         T recip = (T) 1 / t;
183         x *= recip, y *= recip, z *= recip;
184         return *this;
185     }
186 
187     TPoint3 operator-() const {
188         return TPoint3(-x, -y, -z);
189     }
190 
191     bool operator==(const TPoint3 &rhs) const {
192         return x==rhs.x && y==rhs.y && z==rhs.z;
193     }
194 
195     bool operator!=(const TPoint3 &rhs) const {
196         return x!=rhs.x || y!=rhs.y || z!=rhs.z;
197     }
198 
199     T operator[] (int i) const {
200         return (&x)[i];
201     }
202 
203     T& operator[] (int i) {
204         return (&x)[i];
205     }
206 
207     bool isZero() const {
208         return x==0 && y==0 && z==0;
209     }
210 };
211 
212 template <typename T>
213 std::ostream& operator«(std::ostream &os, const TPoint3<T> &p) {
214     os « p.x « " " « p.y « " " « p.z;
215     return os;
216 }
217 
218 template <typename T>
219 TPoint3<T> operator*(T t, const TPoint3<T> &v) {
220     return TPoint3(t*v.x, t*v.y, t*v.z);
221 }
222 
223 template<>
224 inline TPoint3<int> TPoint3<int>::operator/(int i) const {
225     assert(i!=0);
226     return TPoint3<int>(x/i, y/i, z/i);
227 }
228 
229 template<>
230 inline TPoint3<int>& TPoint3<int>::operator/=(int i) {
231     assert(i!=0);
232     x/=i, y/=i, z/i;
233     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

9.25 Transform3d.h

Go to the documentation of this file.

```
1
12 #pragma once
13
14 #include "Geometry.h"
15 #include "Matrix.h"
16
17 #include <memory>
18
19 class Transform3D
20 {
21
22     std::shared_ptr<TransformMatrix3D> matrix;
23
24     bool isDone = false;
25
26 protected:
27     // apply matrix to this object. Cache should be managed locally.
28     virtual void apply() = 0;
29
30 public:
31     Transform3D();
32
33     Transform3D(std::shared_ptr<Transform3D> _matrix);
34
35     void setTranslate(double x, double y, double z);
36
37     void setScale(double x, double y, double z);
38     void setScale(double ratio);
39
40     void setRotateEuler(Angle x, Angle y, Angle z, EulerType type = EulerType::EULER_XYZ);
41
42     void setRotateQuaternion(double w, double x, double y, double z);
43
44     // @brief Rotate by axis. Counterclockwise rotate.
45     void setRotateAxis(Angle angle, Vec3d axis);
46
47     // @brief inform this object that transform setting is DONE and 'you' can apply all transformation
       without redundant calculation. apply() should be called within.
48     void done();
49
50     Point3d getTranslate();
51 };
```

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.

```
1
12 #pragma once
13 #include <assert.h>
14 #include <cmath>
15 #include <iostream>
16
17 template <typename T>
18 struct TVector2 {
19
20     /*---data field---*/
21     T x, y;
22
23     /*----constructor---*/
24     TVector2() { }
25
26     TVector2(T _x, T _y) : x(_x), y(_y) { }
27
28     explicit TVector2(T t) : x(t), y(t) { }
29
30     /*---operator overloading---*/
31     TVector2 operator+(const TVector2 &rhs) const {
32         return TVector2(x + rhs.x, y + rhs.y);
33     }
34
35     TVector2& operator+=(const TVector2 &rhs) {
36         x += rhs.x, y += rhs.y;
37         return *this;
38     }
39
40     TVector2 operator-(const TVector2 &rhs) const {
41         return TVector2(x - rhs.x, y - rhs.y);
42     }
43
44     TVector2& operator-=(const TVector2 &rhs) {
45         x -= rhs.x, y-= rhs.y;
46         return *this;
47     }
48
49     TVector2 operator*(const T t) const {
50         return TVector2(x * t, y * t);
51     }
52
53     TVector2& operator*=(const T t) {
54         x *= t, y *= t;
55         return *this;
56     }
57
58     TVector2 operator/(T t) const {
59         assert(t != 0);
60         T recip = (T) 1 / t;
61         return TVector2(x * recip, y * recip);
62     }
63
64     TVector2& operator/=(T t) {
65         assert(t != 0);
66         T recip = (T) 1/t;
```

```
67          x *= recip, y *= recip;
68          return *this;
69      }
70
71      TVector2 operator-() const {
72          return TVector2(-x, -y);
73      }
74
75      T operator[](int i) const {
76          return (&x)[i];
77      }
78
79      T& operator[](int i) {
80          return (&x)[i];
81      }
82
83      decltype(auto) length2() const {
84          return x * x + y * y;
85      }
86
87      decltype(auto) length() const {
88          return std::sqrt(x*x + y*y);
89      }
90
91      bool operator==(const TVector2 &rhs) const {
92          return x==rhs.x && y==rhs.y;
93      }
94
95      bool operator!=(const TVector2 &rhs) const {
96          return x!=rhs.x || y!=rhs.y;
97      }
98
99      bool isZero() const {
100          return x==0 && y==0;
101      }
102
103 };
104
105 template <typename T>
106 std::ostream& operator«(std::ostream& os, const TVector2<T> &v) {
107     os « v.x « " " « v.y ;
108     return os;
109 }
110
111 template <typename T>
112 TVector2<T> operator*(T t, const TVector2<T> &v) {
113     return TVector2<T>(v.x * t, v.y * t);
114 }
115
116 template <typename T>
117 decltype(auto) dot(const TVector2<T> &v1, const TVector2<T> &v2) {
118     return v1.x*v2.x + v1.y * v2.y;
119 }
120
121 template <typename T>
122 decltype(auto) absDot(const TVector2<T> &v1, const TVector2<T> &v2) {
123     return std::abs(v1.x*v2.x + v1.y * v2.y);
124 }
125
126 template <typename T>
127 TVector2<double> normalize(const TVector2<T> &v) {
128     double recip = 1.0f / v.length();
129     return TVector2<double> (v.x * recip, v.y * recip);
130 }
131
132 template<>
133 inline TVector2<int> TVector2<int>::operator/(int i) const {
134     assert(i != 0);
135     return TVector2<int>(x/i, y/i);
136 }
137
138 template<>
139 inline TVector2<int>& TVector2<int>::operator/=(int i) {
140     assert(i != 0);
141     x/=i, y/=i;
142     return *this;
143 }
144
145
146
147 template <typename T>
148 struct TVector3 {
149     /*--- data field ---*/
150     T x, y, z;
151
152     /*--- constructor ---*/
153     TVector3 () { }
```

```
154
155     TVector3 (T _x, T _y, T _z) : x(_x), y(_y), z(_z) { }
156
157     explicit TVector3 (T t) : x(t), y(t), z(t) { }
158
159     /*--- operator overloading ---*/
160     TVector3 operator+(const TVector3 &rhs) const {
161         return TVector3(x + rhs.x, y + rhs.y, z + rhs.z);
162     }
163
164     TVector3& operator+=(const TVector3 &rhs) {
165         x += rhs.x, y += rhs.y, z += rhs.z;
166         return *this;
167     }
168
169     TVector3 operator-(const TVector3 &rhs) const {
170         return TVector3(x-rhs.x, y-rhs.y, z-rhs.z);
171     }
172
173     TVector3& operator-=(const TVector3 &rhs) {
174         x -= rhs.x, y -= rhs.y, z -= rhs.z;
175         return *this;
176     }
177
178     TVector3 operator*(const T t) const {
179         return TVector3(x*t, y*t, z*t);
180     }
181
182     TVector3& operator*=(const T t) {
183         x*=t, y*=t, z*=t;
184         return *this;
185     }
186
187     TVector3 operator/(const T t) const {
188         assert(t != 0);
189         T recip = (T) 1/t;
190         return TVector3(x*recip, y*recip, z*recip);
191     }
192
193     TVector3& operator/=(const T t) {
194         assert(t != 0);
195         T recip = (T) 1/t;
196         x*=recip, y*=recip, z*=recip;
197         return *this;
198     }
199
200     TVector3 operator-() const {
201         return TVector3(-x, -y, -z);
202     }
203
204     T operator[](const int i) const {
205         return (&x)[i];
206     }
207
208     T& operator[](const int i) {
209         return (&x)[i];
210     }
211
212     decltype(auto) length2() const {
213         return x*x + y*y + z*z;
214     }
215
216     decltype(auto) length() const {
217         return std::sqrt(x*x + y*y + z*z);
218     }
219
220     bool operator==(const TVector3 &rhs) const {
221         return x==rhs.x && y==rhs.y && z==rhs.z;
222     }
223
224     bool operator!=(const TVector3 &rhs) const {
225         return x!=rhs.x || y!=rhs.y || z!=rhs.z;
226     }
227
228     bool isZero() const {
229         return x==0 && y==0 && z==0;
230     }
231 };
232
233 template <typename T>
234 std::ostream& operator«(std::ostream &os, const TVector3<T> &v) {
235     os « v.x « " " « v.y « " " « v.z;
236     return os;
237 }
238
239 template <typename T>
240 TVector3<T> operator*(T t, const TVector3<T> &v) {
```

```
241     return TVector3<T>(v.x*t, v.y*t, v.z*t);
242 }
243
244 template <typename T>
245 decltype(auto) dot(const TVector3<T> &v1, const TVector3<T> &v2) {
246     return v1.x * v2.x + v1.y * v2.y + v1.z * v2.z;
247 }
248
249 template <typename T>
250 decltype(auto) absDot(const TVector3<T> &v1, const TVector3<T> &v2) {
251     return std::abs(v1.x * v2.x + v1.y * v2.y + v1.z * v2.z);
252 }
253
254 template <typename T>
255 TVector3<double> normalize(const TVector3<T> &v) {
256     double recip = 1.0f / v.length();
257     return TVector3<double> (v.x * recip, v.y * recip, v.z * recip);
258 }
259
268 template <typename T>
269 TVector3<T> cross(const TVector3<T> &v1, const TVector3<T> &v2) {
270     return TVector3<T> (
271         v1.y * v2.z - v1.z * v2.y,
272         v1.z * v2.x - v1.x * v2.z,
273         v1.x * v2.y - v1.y * v2.x
274     );
275 }
276
277
278
279
```

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

9.29 Common.h

Go to the documentation of this file.

```
1
13 #pragma  once
14 #include <cmath>
15 #include <limits>
16 #include <utility>
17 // some constant values
18 #define M_PI         3.14159265358979323846f
19 #define INV_PI       0.31830988618379067154f
20 #define INV_TWOPI    0.15915494309189533577f
21
22 constexpr double ONEMINUSEPSILON = 1 - std::numeric_limits<double>::epsilon();
23
25 inline float clamp(float value, float min, float max) {
26     if (value < min)
27         return min;
28     else if (value > max)
29         return max;
30     else return value;
31 }
32
34 inline int clamp(int value, int min, int max) {
35     if (value < min)
36         return min;
37     else if (value > max)
38         return max;
39     else return value;
40 }
41
43 inline float lerp(float t, float v1, float v2) {
44     return ((float) 1 - t) * v1 + t * v2;
45 }
46
48 inline int mod(int a, int b) {
49     int r = a % b;
50     return (r < 0) ? r+b : r;
51 }
52
53 inline  float fresnel(float cosThetaI, float extIOR, float intIOR) {
54     float etaI = extIOR, etaT = intIOR;
55
56     if (extIOR == intIOR)
57         return 0.0f;
58
59     /* Swap the indices of refraction if the interaction starts
60        at the inside of the object */
61     if (cosThetaI < 0.0f) {
62         std::swap(etaI, etaT);
63         cosThetaI = -cosThetaI;
64     }
65
66     /* Using Snell's law, calculate the squared sine of the
67        angle between the normal and the transmitted ray */
68     float eta = etaI / etaT,
69             sinThetaTSqr = eta*eta * (1-cosThetaI*cosThetaI);
70
71     if (sinThetaTSqr > 1.0f)
72         return 1.0f;  /* Total internal reflection! */
73
74     float cosThetaT = std::sqrt(1.0f - sinThetaTSqr);
75
```

```
76     float Rs = (etaI * cosThetaI - etaT * cosThetaT)
77                / (etaI * cosThetaI + etaT * cosThetaT);
78     float Rp = (etaT * cosThetaI - etaI * cosThetaT)
79                / (etaT * cosThetaI + etaI * cosThetaT);
80 
81     return (Rs * Rs + Rp * Rp) / 2.0f;
82 }
83 
84 inline  double fresnel(double cosThetaI, double extIOR, double intIOR) {
85     double etaI = extIOR, etaT = intIOR;
86 
87     if (extIOR == intIOR)
88         return 0.0;
89 
90     /* Swap the indices of refraction if the interaction starts
91        at the inside of the object */
92     if (cosThetaI < 0.0f) {
93         std::swap(etaI, etaT);
94         cosThetaI = -cosThetaI;
95     }
96 
97     /* Using Snell's law, calculate the squared sine of the
98        angle between the normal and the transmitted ray */
99     double eta = etaI / etaT,
100            sinThetaTSqr = eta*eta * (1-cosThetaI*cosThetaI);
101 
102     if (sinThetaTSqr > 1.0f)
103         return 1.0f;  /* Total internal reflection! */
104 
105     double cosThetaT = std::sqrt(1.0f - sinThetaTSqr);
106 
107     double Rs = (etaI * cosThetaI - etaT * cosThetaT)
108                 / (etaI * cosThetaI + etaT * cosThetaT);
109     double Rp = (etaT * cosThetaI - etaI * cosThetaT)
110                 / (etaT * cosThetaI + etaI * cosThetaT);
111 
112     return (Rs * Rs + Rp * Rp) / 2.0;
113 }
114 
115 
116 
117 
118 
119 
120 
```

9.30 Warp.h

```
1 #include "CoreLayer/Geometry/Geometry.h"
2 #include "Common.h"
3 
4 static  double TentInverse(double x){
5     if(x<=.5f)
6         return std::sqrt(2*x)-1;
7     return  1- std::sqrt(2-2*x);
8 }
9 inline   Point2d SquareToTent(const Point2d &sample) {
10     Point2d  res(TentInverse(sample[0]), TentInverse(sample[1]));
11     return  res;
12 //    throw NoriException("SquareToTent() is not yet implemented!");
13 }
14 
15 inline  float SquareToTentPdf(const Point2d &p) {
16     return (1.0-abs(p[0])) * (1.0-abs(p[1]));
17 }
18 
19 
20 //Point2d SquareToUniformDisk(const Point2d &sample) {
21 //    auto phi=2*sample.x()*M_PI;
22 //    auto r=sqrt(sample.y());
23 //    return {r*cos(phi),r*sin(phi)};
24 //}
25 //
26 //float SquareToUniformDiskPdf(const Point2d &p) {
27 //    return  < 1.f ? INV_PI : .0f;}
28 
29 inline  Vec3d SquareToUniformSphere(const Point2d &sample) {
30     float z = 1 - 2 * sample[0];
31     float r = std::sqrt(std::max((float )0, (float)1 - z * z));
32     float phi = 2 * M_PI * sample[1];
33     return {r * std::cos(phi), r * std::sin(phi), z};
34 }
35 
```

```
36 inline  float SquareToUniformSpherePdf(const Vec3d &v) {
37     return 0.25f*INV_PI;
38 }
39
40 inline  Vec3d SquareToUniformHemisphere(const Point2d &sample) {
41     float z = 1 - 2 * sample[0];
42     float r = std::sqrt(std::max((float )0, (float)1 - z * z));
43     float phi = 2 * M_PI * sample[1];
44     return {r * std::cos(phi), r * std::sin(phi), abs(z)};
45 }
46
47 inline float SquareToUniformHemispherePdf(const Vec3d &v) {
48     return v[2] >=0 ? 0.5f * INV_PI : .0f;
49 }
50
51 inline  Vec3d SquareToCosineHemisphere(const Point2d &sample) {
52     float z=sqrt(1-sample.x);
53     float phi=sample.y*2*M_PI;
54
55     return {sqrt(sample.x)* cos(phi),sqrt(sample.x)*sin(phi),z};
56 }
57
58 inline  float SquareToCosineHemispherePdf(const Vec3d &v) {
59     return v[2] >=0 ? v.z * INV_PI : .0f;
60 }
61 //
62 inline  Vec3d SquareToBeckmann(const Point2d &sample,double alpha) {
63     auto tan2theta= -alpha*alpha*log( sample.x );
64     auto cosTheta=sqrt(1/(1+tan2theta));
65     auto sinTheta= sqrt(1-cosTheta*cosTheta);
66     auto phi=sample.y * 2 * M_PI;
67     Vec3d t1= Vec3d(sinTheta*cos(phi), sinTheta*sin(phi),cosTheta);
68     return t1;
69 }
70 //
71 inline  float SquareToBeckmannPdf(const Vec3d &m, double alpha) {
72     if(m.z<=0)
73         return 0.0f;
74     auto cosTheta=m.z;
75     auto sinTheta=sqrt(1-cosTheta*cosTheta);
76     auto tan2Theta=(sinTheta* sinTheta)/(cosTheta*cosTheta);
77     float azimuthal =   INV_PI;
78     float longitudinal = exp(-tan2Theta/(alpha*alpha))  / (alpha*alpha*pow(cosTheta,3));
79     return azimuthal * longitudinal;
80 }
```

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.

```
1
12 #pragma once
13
14 //#include "../core.h"
15 #include <memory>
16 #include "CoreLayer/Geometry/Geometry.h"
17 #include "FunctionLayer/Medium/Medium.h"
18 #include <cfloat>
19 struct Ray
20 {
21     Point3d origin;
22     Vec3d direction;
23
24     double timeMin;
25     double timeMax;
26
27     std::shared_ptr<Medium> medium;
28
29     Point3d at(double t);
30
31     bool withinTime(double time);
32
33     Ray(const Point3d &_origin, const Vec3d &_direction, double _timeMin = .0f, double _timeMax =
       DBL_MAX);
34 };
35
36 struct RayDifferential : public Ray
37 {
38     Point3d origin_x, origin_y;
39     Vec3d direction_x, direction_y;
40 };
```

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.

```
1
12 #pragma once
13
14 #include "CoreLayer/Ray/Ray.h"
15 #include "FunctionLayer/Shape/Entity.h"
16 #include "FunctionLayer/Aggregate/Bvh.h"
17 #include "FunctionLayer/Intersection.h"
18 #include "FunctionLayer/Light/Light.h"
19
20 #include <optional>
21
22 class Scene
23 {
24     std::shared_ptr<Bvh> BVH;
25     std::shared_ptr<std::vector<std::shared_ptr<Light>>> lights;
26     std::shared_ptr<std::vector<std::shared_ptr<Entity>>> entities;
27
28 public:
29     Scene();
30     void addEntity(std::shared_ptr<Entity> object);
31     void addLight(std::shared_ptr<Light> light);
32
33     void build();
34     std::optional<Intersection> intersect(const Ray &r) const;
35
36     // @return true if r hits object first (closest), false otherwise.
37     bool intersectionTest(const Ray &r, std::shared_ptr<Entity> object) const;
38
39     std::shared_ptr<std::vector<std::shared_ptr<Light>>> getLights() const;
40 };
```

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

9.38 Bvh.h

Go to the documentation of this file.

```
1
12 #pragma once
13
14 #include "CoreLayer/Geometry/BoundingBox.h"
15 #include "FunctionLayer/Shape/Entity.h"
16 #include "FunctionLayer/Intersection.h"
17
18 //@brief Entity information declaration for building BVH
19 struct EntityInfo {
20     EntityInfo(){}
21     EntityInfo(int _EntityId, const BoundingBox3f& _bounds): EntityId(_EntityId), bounds(_bounds),
       center(0.5 * (_bounds.pMin + _bounds.pMax)){}
22     int EntityId;
23     BoundingBox3f bounds;
24     Point3d center;
25 };
26
27 struct BvhTreeNode {
28     BoundingBox3f bounds;
29     std::shared_ptr<BvhTreeNode> children[2] = {nullptr, nullptr};//0: left, 1: right
30     int splitAxis;
31     int nEntites = 0;//0: interior nodes, otherwise: leaf nodes
32     int entityOffset;
33 };
34
35 //@brief Bvh Nodes in Dfs-Order
36 struct LinearBvhNode {
37     BoundingBox3f bounds;
38     union
39     {
40         int firstdEntityOffset;//for leaves to enumerate
41         int secondChildOrder;//for interior nodes to traverse
42     };
43     int nEntites = 0;
44     int splitAxis;
45 };
46
47 struct Bvh {
48 public:
49     enum class SplitMethod{
50         SAH,
51         Middle,
52         EqualCounts
53     };//not support LBVH yet
54     const SplitMethod splitMethod;
55     std::vector<std::shared_ptr<Entity» entites;
56     std::vector<LinearBvhNode> linearBvhNodes;
57
58     /*
59      * @brief Bvh constructor
60      * @param <_entites>
61      * @param <_SplitMeshod>
62      */
63     Bvh(std::vector<std::shared_ptr<Entity»& _entites, SplitMethod _splitMethod = SplitMethod::SAH);
64
65     /*
66     * @brief recursively build BVH
67     * @return the root of BVH
68     */
69     std::shared_ptr<BvhTreeNode> RecursiveBuild(std::vector<EntityInfo>& entityInfo, int start, int end,
       int& nodeNumber, std::vector<std::shared_ptr<Entity»& orderedEntites);
70
71     //@brief flatten the BVH to Dfs-Order
72     void Flatten(std::shared_ptr<BvhTreeNode> node, int& dfsOrder);
73
74     //@brief return the scene intersection
75     std::optional<Intersection> Intersect(const Ray& r);
76 };
```

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.

```
1
12 #pragma once
13
14 #include "CoreLayer/Geometry/Geometry.h"
15 #include "CoreLayer/Ray/Ray.h"
16
17 struct CameraSample {
18     Point2d xy;
19     double time;
20     Point2d lens;
21 };
22
23 class Camera
24 {
25 public:
26
27     virtual Ray generateRay(const Point2i &filmResolution,
28                             const Point2i &pixelPosition,
29                             const CameraSample &sample) const = 0;
30 };
```

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.

```
1
12 #pragma once
13 #include "CoreLayer/Geometry/Matrix.h"
14 #include "Camera.h"
15
16 class PerspectiveCamera : public Camera {
17 protected:
18 public:
19     // TODO replace martrix with transform3d
22     //
23     //     [0, 0]---------------> x
24     //          |      |
25     //          |------+ [x, y]
26     //          |
27     //          y
28     Matrix4x4 cameraToWorld, sampleToFilm;
29 public:
30     PerspectiveCamera() = default;
31
42     PerspectiveCamera(
43         const Point3d &lookFrom,
44         const Point3d &lookAt,
45         const Vec3d &up,
46         double xFov,
47         double aspectRatio,
48         double distToFilm
49     )
50     {
51         cameraToWorld =
52             Matrix4x4::lookAt(lookFrom, lookAt - lookFrom, up).inverse();
53         // ! near is the distToFilm and far set to MAX_FLOAT
54         // ! if far set to MAX_DOUBLE, it will crash when computing the matrix
55         Matrix4x4 filmToSample = Matrix4x4::perspective(
56             Angle(xFov, Angle::AngleType::ANGLE_DEG),
57             aspectRatio,
58             distToFilm,
59             std::numeric_limits<float>::max()
60         );
61         sampleToFilm =
62             Matrix4x4::scale(0.5, -0.5, 1.0)
63             * Matrix4x4::translate(1, -1, 0)
64             * filmToSample;
65         sampleToFilm = sampleToFilm.inverse();
66     }
67 };
```

9.43 Pinhole.h

```
1 #pragma once
2
3 #include "Perspective.h"
4
5 class PinholeCamera : public PerspectiveCamera {
6 public:
7     PinholeCamera() = default;
8     PinholeCamera(
9         const Point3d &lookFrom,
10         const Point3d &lookAt,
11         const Vec3d &up,
12         double xFov,
13         double aspectRatio,
14         double distToFilm
15     ) : PerspectiveCamera(lookFrom, lookAt, up, xFov, aspectRatio, distToFilm) { }
16
24     virtual Ray generateRay(const Point2i &filmResolution,
25                             const Point2i &pixelPosition,
26                             const CameraSample &sample) const override;
27 };
```

9.44 Thinlens.h

```
1 #pragma once
2
3 #include "Perspective.h"
4
5 class ThinlensCamera : public PerspectiveCamera {
6 protected:
7     double apertureRadius, focalLen, focalDistance;
8 public:
9     ThinlensCamera() = default;
10     ThinlensCamera(
11         const Point3d &lookFrom,
12         const Point3d &lookAt,
13         const Vec3d &up,
14         double xFov,
15         double aspectRatio,
16         double _focaDistance,
17         double _apertureRadius = 0.1,
18         double _focalLen = 0.05
19     ):apertureRadius(_apertureRadius), focalLen(_focalLen), focalDistance(_focaDistance) {
20         double distToFilm =
21             (_focalLen * _focaDistance) / (_focaDistance - _focalLen);
22
23         cameraToWorld =
24             Matrix4x4::lookAt(lookFrom, lookAt - lookFrom, up).inverse();
25         // ! near is the distToFilm and far set to MAX_FLOAT
26         // ! if far set to MAX_DOUBLE, it will crash when computing the matrix
27         Matrix4x4 filmToSample = Matrix4x4::perspective(
28             Angle(xFov, Angle::AngleType::ANGLE_DEG),
29             aspectRatio,
30             distToFilm,
31             std::numeric_limits<float>::max()
32         );
33         sampleToFilm =
34             Matrix4x4::scale(0.5, -0.5, 1.0)
35             * Matrix4x4::translate(1, -1, 0)
36             * filmToSample;
37         sampleToFilm = sampleToFilm.inverse();
38
39     }
40
41
42     virtual Ray generateRay(const Point2i &filmResolution ,
43                             const Point2i &pixelPosition,
44                             const CameraSample &sample) const override;
45 };
```

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

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

9.47 Film.h

Go to the documentation of this file.

```
1
12 #pragma once
13
14 #include "FunctionLayer/Filter/Filter.h"
15 #include "ResourceLayer/ResourceManager.h"
16 #include <memory>
17
18 class Film
19 {
20 protected:
21     std::unique_ptr<Image> image;
22     std::shared_ptr<Filter> filter;
23     Point2i resolution;
24     int channels;
25     std::vector<double> sumWeights; // a temp impl
26     std::vector<Spectrum> sumValues; // a temp impl
27
28     void syncWithGui();
29
30 public:
31     Film(Point2i resolution, int channels);
32     Film(Point3i shape);
33
34     Spectrum getSpectrum(const Point2i &p);
35     RGB3 getRGB(const Point2i &p);
36
37     void setGamma(double gamma = 2.2);
38     void setFilter(std::shared_ptr<Filter> filter);
39     Point2i getResolution() const;
40
41     // @brief: 'deposit' a spectrum at p. all deposit will be averaged when get(p).
42     void deposit(const Point2i &p, const Spectrum &s);
43     void save(const std::string &path);
44
45     int getDepositeCount(const Point2i &p);
46 };
```

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

9.49 Filter.h

Go to the documentation of this file.

```
1
12 #pragma once
13
14 #include "CoreLayer/Geometry/Geometry.h"
15
16 // ! Note that we cannot use any non-linear "filter" here, such as bilteral and etc
17
18 // @brief: Base class for linear filter used for film
19 class Filter
20 {
21 public:
22     virtual float eval(const Point2d &pos) = 0;
23
24     // @brief: also known as Kernel Size
25     virtual Point2d support() = 0;
26 };
27
```

9.50 /mnt/renderer/Zero/src/FunctionLayer/Integrator/AbstractPathIntegrator.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/AbstractPathIntegrator.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

9.52 AbstractPathIntegrator.h

Go to the documentation of this file.

```
1
12 #pragma once
13
14 #include <cmath>
15 #include "CoreLayer/Ray/Ray.h"
16 #include "MonteCarloIntegrator.h"
17
18 struct PathIntegratorLocalRecord
19 {
20     Vec3d wi;
21     Spectrum f;
22     double pdf;
23     bool isDelta = false;
24 };
25
26 class AbstractPathIntegrator : public MonteCarloIntegrator
27 {
28 protected:
29     const int nDirectLightSamples = 1;
30     const double misWeightPower = 1.0f;
31
32 public:
33     AbstractPathIntegrator(
34         std::shared_ptr<Camera> _camera,
35         std::unique_ptr<Film> _film,
36         std::unique_ptr<TileGenerator> _tileGenerator,
37         std::shared_ptr<Sampler> _sampler,
38         int _spp,
39         int _renderThreadNum=4);
40
41     virtual Spectrum Li(const Ray &ray, std::shared_ptr<Scene> scene);
42     virtual double MISWeight(double x, double y);
43
44     /*************************************************************
45     Functions below need to be implemented in derived classes
46     *************************************************************/
47
48     // @brief Return the radiance along given ray, emitted from given intersection.
49     // @param scene     Ptr to scene.
50     // @param its       Intersection hit by ray.
51     // @param ray       Ray to evaluate.
52     // @return          Direction of given ray, incident radiance at origin of ray, pdf of direct light
       sampling.
53     virtual PathIntegratorLocalRecord evalEmittance(std::shared_ptr<Scene> scene,
54                                                     std::optional<Intersection> its,
```

```
55                                                  const Ray &ray) = 0;
56
57     // @brief Sample incident direction of direct lighting.
58     // @param scene     Ptr to scene.
59     // @param its       Reference point.
60     // @param ray       Ray, used to specify wo (out direction).
61     // @return          Sampled incident direction, incident radiance and pdf per solid angle.
62     virtual PathIntegratorLocalRecord sampleDirectLighting(std::shared_ptr<Scene> scene,
63                                                            const Intersection &its,
64                                                            const Ray &ray) = 0;
65
66     // @brief Return scatter value of BSDF or phase function.
67     // @param scene     Ptr to scene.
68     // @param its       Reference point.
69     // @param ray       Ray, used to specify wo (out direction).
70     // @param wi        Incident direction wi.
71     // @return          Incident direction, scatter throughput f, pdf per solid angle.
72     //                  For surface, f is the product of BSDF value and cosine term.
73     //                  For medium, f is the value of phase function.
74     virtual PathIntegratorLocalRecord evalScatter(std::shared_ptr<Scene> scene,
75                                                   const Intersection &its,
76                                                   const Ray &ray,
77                                                   const Vec3d &wi) = 0;
78
79     // @brief Sample incident direction by scatter value of BSDF or phase function.
80     // @param scene     Ptr to scene.
81     // @param its       Reference point.
82     // @param ray       Ray, used to specify wo (out direction).
83     // @return          Sampled incident direction, scatter throughput f, pdf per solid angle.
84     //                  For surface, f is the product of BSDF value and cosine term.
85     //                  For medium, f is the value of phase function.
86     virtual PathIntegratorLocalRecord sampleScatter(std::shared_ptr<Scene> scene,
87                                                     const Intersection &its,
88                                                     const Ray &ray) = 0;
89
90     // @brief Return probability of Russian roulette.
91     virtual double russianRoulette(std::shared_ptr<Scene> scene,
92                                    const Intersection &its,
93                                    const Spectrum &T,
94                                    int nBounce) = 0;
95 };
```

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

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

9.55 Integrator.h

Go to the documentation of this file.

```
1
12 #pragma once
13
14 #include "CoreLayer/Scene/Scene.h"
15 #include "FunctionLayer/Camera/Camera.h"
16 #include "FunctionLayer/Film/Film.h"
17 #include "FunctionLayer/Sampler/Sampler.h"
18 #include "FunctionLayer/TileGenerator/TileGenerator.h"
19
20 #include <memory>
21
22 class Integrator
23 {
24 protected:
25     std::shared_ptr<Camera> camera;
26     std::unique_ptr<Film> film;
27
28 public:
29     Integrator(std::shared_ptr<Camera> _camera, std::unique_ptr<Film> _film);
30     virtual void render(std::shared_ptr<Scene> scene) = 0;
31     virtual void save(const std::string& path);
32
33 };
```

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.

```
1
13 #pragma once
14
15 #include <cmath>
16 #include "Integrator.h"
17
18 class MonteCarloIntegrator : public Integrator
19 {
20 protected:
21     std::shared_ptr<Sampler> sampler;
22     int spp = 4;
23
24     std::shared_ptr<TileGenerator> tileGenerator;
25
26     // @brief: render process per thread. Should be called in render().
27     void renderPerThread(std::shared_ptr<Scene> scene);
28
29     int renderThreadNum=4;
30
31 public:
32     MonteCarloIntegrator(
33         std::shared_ptr<Camera> _camera,
34         std::unique_ptr<Film> _film,
35         std::unique_ptr<TileGenerator> _tileGenerator,
36         std::shared_ptr<Sampler> _sampler,
37         int _spp,
38         int _renderThreadNum=4
39         );
40
41     virtual void render(std::shared_ptr<Scene> scene);
42
43     // @brief Estimate radiance along a given ray
44     virtual Spectrum Li(const Ray &ray, std::shared_ptr<Scene> scene) = 0;
45     // @brief Get a random number WITHOUT using MonteCarloIntegrator::sampler
46     virtual double randFloat();
47 };
```

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.

```
1
12 #pragma once
13
14 #include <cmath>
15 #include "CoreLayer/Ray/Ray.h"
16 #include "AbstractPathIntegrator.h"
17
18 class PathIntegrator : public AbstractPathIntegrator
19 {
20 protected:
21     const int nPathLengthLimit = 20;
22     const double pRussianRoulette = 0.95;
23
24 public:
25     PathIntegrator(
26         std::shared_ptr<Camera> _camera,
27         std::unique_ptr<Film> _film,
28         std::unique_ptr<TileGenerator> _tileGenerator,
29         std::shared_ptr<Sampler> _sampler,
30         int _spp,
31         int _renderThreadNum=4);
32
33     // @brief Return the radiance along given ray, emitted from given intersection.
34     // @param scene     Ptr to scene.
35     // @param its       Intersection hit by ray.
36     // @param ray       Ray to evaluate.
37     // @return          Direction of given ray, incident radiance at origin of ray, pdf of direct light
       sampling.
38     virtual PathIntegratorLocalRecord evalEmittance(std::shared_ptr<Scene> scene,
39                                                     std::optional<Intersection> its,
40                                                     const Ray &ray) override;
41
42     // @brief Sample incident direction of direct lighting.
43     // @param scene     Ptr to scene.
44     // @param its       Reference point.
45     // @param ray       Ray, used to specify wo (out direction).
46     // @return          Sampled incident direction, incident radiance and pdf per solid angle.
47     virtual PathIntegratorLocalRecord sampleDirectLighting(std::shared_ptr<Scene> scene,
48                                                            const Intersection &its,
49                                                            const Ray &ray) override;
50
51     // @brief Return scatter value of BSDF or phase function.
52     // @param scene     Ptr to scene.
53     // @param its       Reference point.
54     // @param ray       Ray, used to specify wo (out direction).
55     // @param wi        Incident direction wi.
56     // @return          Incident direction, scatter throughput f, pdf per solid angle.
57     //                  For surface, f is the product of BSDF value and cosine term.
58     //                  For medium, f is the value of phase function.
59     virtual PathIntegratorLocalRecord evalScatter(std::shared_ptr<Scene> scene,
60                                                   const Intersection &its,
61                                                   const Ray &ray,
62                                                   const Vec3d &wi) override;
63
64     // @brief Sample incident direction by scatter value of BSDF or phase function.
65     // @param scene     Ptr to scene.
66     // @param its       Reference point.
67     // @param ray       Ray, used to specify wo (out direction).
68     // @return          Sampled incident direction, scatter throughput f, pdf per solid angle.
69     //                  For surface, f is the product of BSDF value and cosine term.
70     //                  For medium, f is the value of phase function.
71     virtual PathIntegratorLocalRecord sampleScatter(std::shared_ptr<Scene> scene,
72                                                     const Intersection &its,
73                                                     const Ray &ray);
74
75     // @brief Return survive probability of Russian roulette.
76     virtual double russianRoulette(std::shared_ptr<Scene> scene,
77                                    const Intersection &its,
78                                    const Spectrum &T,
79                                    int nBounce) override;
80
81     // todo: move light sampling into a new class (called LightDistribution?)
82     // @brief Sample a light, by some weight distribution
83     virtual std::pair<std::shared_ptr<Light>, double> chooseOneLight(std::shared_ptr<Scene> scene,
84                                                                      const Intersection &its,
85                                                                      const Ray &ray,
86                                                                      double lightSample);
87
88     // @brief Probability of choosing a specified light source
89     virtual double chooseOneLightPdf(std::shared_ptr<Scene> scene,
90                                      const Intersection &its,
91                                      const Ray &ray,
```

```
92                                      std::shared_ptr<Light> light);
93
94     // @brief Evaluate radiance of env lights
95     virtual PathIntegratorLocalRecord evalEnvLights(std::shared_ptr<Scene> scene,
96                                                     const Ray &ray);
97 };
```

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

9.62 Intersection.h

Go to the documentation of this file.

```
1
12 #pragma once
13
14 #include "CoreLayer/Geometry/Geometry.h"
15 #include "CoreLayer/Geometry/Frame.h"
16 #include "FunctionLayer/Material/Material.h"
17 #include "FunctionLayer/Medium/Medium.h"
18 #include "FunctionLayer/Shape/Entity.h"
19
20 #include <memory>
21
22 struct Intersection
23 {
```

```
24     Point3d position;
25     Normal3d geometryNormal;
26     Normal3d geometryTangent;
27     Normal3d geometryBitangent;
28     Point2d uv;
29
30     // shadingFrame
31     Frame shFrame;
32
33     Vec3d dpdu, dpdv;
34     Normal3d dndu, dndv;
35
36     // std::shared_ptr<Entity> object;
37     const Entity *object;
38
39     std::shared_ptr<Material> material;
40     std::shared_ptr<Medium> mediumInside;
41     std::shared_ptr<Medium> mediumOutside;
42
43     Vec3d toLocal(const Vec3d &d) const
44     {
45         return shFrame.toLocal(d);
46     }
47
48     Vec3d toWorld(const Vec3d &d) const
49     {
50         return shFrame.toWorld(d);
51     }
52 };
```

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

9.64 Intersection.h

Go to the documentation of this file.

```
1
12 #pragma once
13
14 #include "CoreLayer/Geometry/Frame.h"
15 #include "CoreLayer/Geometry/Geometry.h"
16 #include "FunctionLayer/Material/Material.h"
17 #include "FunctionLayer/Medium/Medium.h"
18 #include "Entity.h"
19
20 #include <memory>
21
22 struct Intersection
23 {
24     Point3d position;
25     Normal3d geometryNormal;
26     Normal3d geometryTangent;
27     Normal3d geometryBitangent;
28     Point2d uv;
29
30     // shadingFrame
31     Frame shFrame;
32
33     Point3d dpdu, dpdv;
34     Normal3d dndu, dndv;
35
36     // std::shared_ptr<Entity> object;
37     const Entity *object;
38
39     std::shared_ptr<Material> material;
40     std::shared_ptr<Medium> mediumInside;
41     std::shared_ptr<Medium> mediumOutside;
42
43     Vec3d toLocal(const Vec3d &d) const
44     {
45         return shFrame.toLocal(d);
46     }
47
48     Vec3d toWorld(const Vec3d &d) const
49     {
50         return shFrame.toWorld(d);
51     }
52 };
```

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

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

9.67 AreaLight.h

Go to the documentation of this file.

```
1
13 #include "CoreLayer/Geometry/Transform3d.h"
14 #include "Light.h"
15
16 class AreaLight : public Light
17 {
18 protected:
19     std::shared_ptr<Transform3D> transform;
20
21 public:
22     AreaLight(std::shared_ptr<Transform3D> transform3D);
23 };
```

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.

```
1
13 #include "AreaLight.h"
14 #include "CoreLayer/Geometry/Transform3d.h"
15 #include "FunctionLayer/Shape/Entity.h"
16
17 class DiffuseAreaLight : public AreaLight
18 {
19     std::shared_ptr<Entity> shape;
20     Spectrum radiance;
21
22 public:
23     DiffuseAreaLight(std::shared_ptr<Entity> shape,
24                      Spectrum radiance);
25     virtual LightSampleResult evalEnvironment(const Ray &ray) override;
26     virtual LightSampleResult eval(const Ray& ray, const Intersection &its, const Vec3d &d) override;
27     virtual LightSampleResult sampleEmit(const Point2d &positionSample, const Point2d &directionSample,
       float time) override;
28     virtual LightSampleResult sampleDirect(const Intersection &its, const Point2d &sample, float time)
       override;
29 };
```

9.71 Light.h

```
1
12 #pragma once
13 #include "CoreLayer/Geometry/Geometry.h"
14 #include "CoreLayer/ColorSpace/Color.h"
15 #include "CoreLayer/Ray/Ray.h"
16 #include "FunctionLayer/Intersection.h"
17
18 struct LightSampleResult
19 {
20     // @brief Spectrum from light.
21     Spectrum s;
22
23     // @brief Point on an object where receives light if exists.
24     Point3d src;
25
26     // @brief Point on a light where emits light if exists.
27     Point3d dst;
28
29     // @brief Direction from src to dst.
30     Vec3d wi;
31
32     Normal3d dstNormal;
33     Point2d uv;
34
35     // @brief PDF of direct light sampling.
36     double pdfDirect;
37     // @brief Positional emission PDF if exists.
38     double pdfEmitPos;
39     // @brief Directional emission PDF if exists.
40     double pdfEmitDir;
41
42     // @brief FALSE for area and volume light, TRUE for point and etc
43     bool isDeltaPos;
44     // @brief TRUE for distant light
45     bool isDeltaDir;
46 };
47
48 class Light
49 {
50 public:
51     // @brief Assume that the given ray hits nothing in the scene.
52     // Note that this function will not return a direct light sampling PDF.
53     virtual LightSampleResult evalEnvironment(const Ray &ray) = 0;
54     // @brief Assume that the given intersection is on the emitter.
55     // Note that param ray is only used to fill 'src' field.
56     virtual LightSampleResult eval(const Ray& ray, const Intersection &its, const Vec3d &d) = 0;
57     // @brief Note that this function will not return a direct light sampling PDF.
58     virtual LightSampleResult sampleEmit(const Point2d &positionSample, const Point2d &directionSample,
       float time) = 0;
59     virtual LightSampleResult sampleDirect(const Intersection &its, const Point2d &sample, float time) =
       0;
60 };
```

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.

```
1
13 #pragma once
14
15 #include "Light.h"
16 #include "CoreLayer/Geometry/Transform3d.h"
17
18 #pragma  once
19
20 class PointLight : public Light, public Transform3D
21 {
22 protected:
23     Spectrum intensity;
24
25 public:
26     PointLight(const Spectrum &intensity, const Point3d &center);
27     virtual void apply() override;
28     virtual LightSampleResult evalEnvironment(const Ray &ray) override;
29     virtual LightSampleResult eval(const Ray &ray, const Intersection &its, const Vec3d &d) override;
30     virtual LightSampleResult sampleEmit(const Point2d &positionSample, const Point2d &directionSample,
       float time) override;
31     virtual LightSampleResult sampleDirect(const Intersection &its, const Point2d &sample, float time)
       override;
32 };
```

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.

```
1
12 #pragma once
13
14 class BSSRDF
15 {
16     // TODO
17 };
```

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.

```
1
12 #pragma once
13
14 #include "CoreLayer/ColorSpace/Color.h"
15 #include "CoreLayer/Geometry/Geometry.h"
16 #include "CoreLayer/Math/Warp.h"
17 struct BxDFSampleResult
18 {
19     Spectrum s;
20     Vec3d directionIn;
21     double pdf;
22     bool isSpecular;
23 };
24
25 // @brief BxDF. out == rays from/to camera, in == rays from/to objects/lights.
26 class BxDF
27 {
28
29 public:
30     virtual Spectrum f(const Vec3d &out, const Vec3d &in) const = 0;
31
32     virtual Vec3d sampleWi(const Vec3d &out, const Point2d& sample) const = 0;
33
34     virtual double pdf(const Vec3d &out, const Vec3d &in) const = 0;
35
36     virtual BxDFSampleResult sample(const Vec3d &out, const Point2d& sample) const = 0;
37
38     virtual bool isSpecular() const = 0;
39 };
```

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.

```
1
12 #pragma  once
13 #include "CoreLayer/Geometry/Frame.h"
14 #include "BxDF.h"
15 class Dielectric : public  BxDF{
16 public:
17     Dielectric(float m_intIOR =1.5046f,float m_extIOR = 1.00277f) : m_intIOR(m_intIOR),
       m_extIOR(m_extIOR){
18
19     }
20
21     virtual Spectrum f(const Vec3d &wo, const Vec3d &wi) const ;
22
23     virtual Vec3d sampleWi(const Vec3d &wo, const Point2d &sample) const ;
24
25     virtual double pdf(const Vec3d &wo, const Vec3d &wi) const ;
26
27
28     virtual BxDFSampleResult sample(const Vec3d &wo, const Point2d &sample) const ;
29
30     virtual bool isSpecular() const ;
31
32
33 private:
34    float m_intIOR;
35    float m_extIOR;
36
37     static Vec3d reflect(const Vec3d &wi) {
```

```
38          return Vec3d {
39                  -wi[0], -wi[1], wi[2]
40          };
41      }
42
43      Vec3d refract(const Vec3d &wi) const {
44          float cosThetaI = Frame::cosTheta(wi),
45                  eta = cosThetaI > 0 ? m_extIOR / m_intIOR : m_intIOR / m_extIOR,
46                  cosThetaT = std::sqrt(
47                  1 - eta*eta*(1-cosThetaI*cosThetaI)
48          );
49          return Vec3d {
50                  - wi[0] * eta,
51                  - wi[1] * eta,
52                  cosThetaI > 0 ? -cosThetaT : cosThetaT
53          };
54      }
55
56
57 };
58
59
```

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

9.83 Diffuse.h

Go to the documentation of this file.

```
1
12 #pragma  once
13 #include "CoreLayer/Geometry/Frame.h"
14 #include "BxDF.h"
15
16
17
18 class Diffuse : public  BxDF{
19 private:
20
21     Spectrum  albedo;
22
23 public:
24
25     Diffuse(Spectrum albedo);
26
27     virtual Spectrum f(const Vec3d &wo, const Vec3d &wi) const;
28
29     virtual Vec3d sampleWi(const Vec3d &wo, const Point2d &sample) const;
30
31     virtual double pdf(const Vec3d &wo, const Vec3d &wi) const;
32
33     virtual BxDFSampleResult sample(const Vec3d &wo, const Point2d &sample) const;
34
35     virtual bool isSpecular() const;
36
37 };
38
39
40
```

9.84 Microfacet.h

```
1
13 #pragma  once
14
15 #include "BxDF.h"
16 #include "CoreLayer/Geometry/Frame.h"
17
18 class Mircofacet : public  BxDF {
19
20
21 public:
22     Mircofacet(Spectrum kd, double intIOR, double extIOR,double alpha);
23
24     virtual Spectrum f(const Vec3d &wo, const Vec3d &wi) const;
25
26     virtual Vec3d sampleWi(const Vec3d &wo, const Point2d &sample) const;
27
28     virtual double pdf(const Vec3d &wo, const Vec3d &wi) const;
29
30     virtual BxDFSampleResult sample(const Vec3d &wo, const Point2d &sample) const;
31
32     virtual bool isSpecular() const;
33 private:
34     Spectrum kd;
35     double ks;
36     double intIOR,extIOR,alpha;
37 };
38
39
```

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.

```
1
13 #include "CoreLayer/Geometry/Frame.h"
14 #include "BxDF.h"
15
16 #pragma  once
17
18 class Mirror : public  BxDF {
19
20     virtual Spectrum f(const Vec3d &wo, const Vec3d &wi) const;
21
22     virtual Vec3d sampleWi(const Vec3d &wo, const Point2d &sample) const;
23
24     virtual double pdf(const Vec3d &wo, const Vec3d &wi) const;
25
26     virtual BxDFSampleResult sample(const Vec3d &wo, const Point2d &sample) const;
27
28     virtual bool isSpecular() const;
29 };
30
```

9.87 TestMirrorBxdf.h

```
1 #pragma once
2
3 #include "BxDF.h"
4 #include "FunctionLayer/Intersection.h"
5
6 class TestMirrorBxdf : public BxDF
7 {
8 public:
9     virtual Spectrum f(const Vec3d &wo, const Vec3d &wi) const;
10
11     virtual Vec3d sampleWi(const Vec3d &wo, const Point2d &sample) const;
12
13     virtual double pdf(const Vec3d &wo, const Vec3d &wi) const;
14
15     virtual BxDFSampleResult sample(const Vec3d &wo, const Point2d &sample) const;
16
17     virtual bool isSpecular() const;
18 };
```

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

9.89 DelectricMaterial.h

Go to the documentation of this file.

```
1
13 #pragma  once
14 #include "Material.h"
15 #include "FunctionLayer/Material/BxDF/Dielectric.h"
16 #include "FunctionLayer/Intersection.h"
17 #include "FunctionLayer/Texture/Texture.h"
18
19 class DelectricMaterial : public  Material{
20
21 public:
22     virtual std::shared_ptr<BxDF> getBxDF(Intersection intersect) const;
23
24     virtual std::shared_ptr<BSSRDF> getBSSRDF(Intersection intersect) const;
25
26     DelectricMaterial(const  std::shared_ptr<Texture<float» & m_intIDR, const
       std::shared_ptr<Texture<float» & m_extIDR) : m_intIDR(m_intIDR)
27     {
28
29     }
30
31     DelectricMaterial(){
32
33     }
34 private:
35     std::shared_ptr<Texture<float» m_intIDR;
36     std::shared_ptr<Texture<float» m_extIDR;
37
38 };
39
40
```

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

9.91 Material.h

Go to the documentation of this file.

```
1
12 #pragma once
13
14 #include "FunctionLayer/Material/BxDF/BxDF.h"
15 #include "FunctionLayer/Material/BSSRDF/BSSRDF.h"
16 #include <memory>
17 //#include "Intersection.h"
18
19
20
21 struct Intersection;
22
23 class Material
24 {
25 public:
26
27     virtual std::shared_ptr<BxDF> getBxDF(Intersection intersect) const = 0;
28     virtual std::shared_ptr<BSSRDF> getBSSRDF(Intersection intersect) const = 0;
29 };
```

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

9.93 MatteMaterial.h

Go to the documentation of this file.

```
1
12 #pragma  once
13 #include "Material.h"
14 #include "FunctionLayer/Intersection.h"
15 #include "FunctionLayer/Texture/Texture.h"
16
17 class MatteMaterial     : public  Material{
18 private:
19     std::shared_ptr<Texture<Spectrum» kd ;
20 public:
21     MatteMaterial(const std::shared_ptr<Texture<Spectrum» & kd );
22
23     virtual std::shared_ptr<BxDF> getBxDF(Intersection intersect) const;
24
25     virtual std::shared_ptr<BSSRDF> getBSSRDF(Intersection intersect) const;
26
27 };
28
29
```

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.

```
1
13 #include "Material.h"
14 #include "FunctionLayer/Material/BxDF/Mirror.h"
15 #include "FunctionLayer/Intersection.h"
16
17 #pragma  once
18 class MirrorMaterial  : public  Material{
19
20
21 public:
22     virtual std::shared_ptr<BxDF> getBxDF(Intersection intersect) const;
23     virtual std::shared_ptr<BSSRDF> getBSSRDF(Intersection intersect) const;
24 };
```

9.96 TestMirror.h

```
1
2 #pragma once
3 #include "Material.h"
4 #include "FunctionLayer/Material/BxDF/TestMirrorBxdf.h"
5
6 class TestMirror : public Material
7 {
8 private:
9 public:
10     TestMirror();
11     virtual std::shared_ptr<BxDF> getBxDF(Intersection intersect) const;
12     virtual std::shared_ptr<BSSRDF> getBSSRDF(Intersection intersect) const;
13 };
```

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.

```
1
12 #pragma once
13
14 class Medium
15 {
16     // TODO
17 };
```

9.99 DirectSampler.h

```
1 #pragma once
2
3 #include "Sampler.h"
4
5 #pragma  once
6 class DirectSampler : public Sampler
7 {
8 public:
9     virtual double sample() const;
10     virtual std::vector<double> sample(int num) const;
11 };
```

9.100 Independent.h

```
1 #pragma once
2 #include "Sampler.h"
3
4 class IndependentSampler : public Sampler {
5 public:
6     IndependentSampler() = default;
7
8     virtual ~IndependentSampler() = default;
9
10     virtual void startPixel(const Point2i &pixelPosition) override {
11         // do nothing
12     }
13
14     virtual void nextSample() override {
15         // do nothing
16     }
17
18     virtual double sample1D() override {
19         return rng();
20     }
21
22     virtual Point2d sample2D() override {
23         return Point2d{rng(), rng()};
24     }
25 };
```

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.

```
1
12 #pragma once
```

```
13
14 #include <vector>
15 #include "CoreLayer/Adapter/random.h"
16 #include "CoreLayer/Geometry/Geometry.h"
17 #include "CoreLayer/Math/Common.h"
18
23 struct CameraSample;
24
25 class Sampler
26 {
27 protected:
28     RandomNumberGenerator rng;
29 public:
30     Sampler() = default;
31
32     virtual ~Sampler() = default;
33
34     virtual void startPixel(const Point2i &pixelPosition) = 0;
35
36     virtual void nextSample() = 0;
37
38     virtual double sample1D() = 0;
39
40     virtual Point2d sample2D() = 0;
41
42     CameraSample getCameraSample();
43
44 };
45
52 template<typename T>
53 void shuffle(std::vector<T> &samples, RandomNumberGenerator &rng) {
54     int count = samples.size();
55     for (int i = 0; i < count; ++i) {
56         int other = rng(i, count);
57         std::swap(samples[i], samples[other]);
58     }
59 }
60
65 class PixelSampler : public Sampler {
66 protected:
67     Point2i pixelPosition;
68     std::vector<std::vector<double>>  samples1D;
69     std::vector<std::vector<Point2d>> samples2D;
70     int sppSqrt;
71     int samplesPerPixel;
72     int nDimensions;
73     int curSamplePixelIndex;
74     int curDimensionIndex1D;
75     int curDimensionIndex2D;
76
77     virtual void generateSamples1D(std::vector<double> &samples) = 0;
78     virtual void generateSamples2D(std::vector<Point2d> &samples) = 0;
79
80 public:
81     PixelSampler() = delete;
82
83     PixelSampler(int _sppSqrt)
84         : sppSqrt(_sppSqrt), samplesPerPixel(_sppSqrt * _sppSqrt), nDimensions(4),
    curSamplePixelIndex(0), curDimensionIndex1D(0), curDimensionIndex2D(0) {
85         for (int i = 0; i < nDimensions; ++i) {
86             samples1D.emplace_back(std::vector<double>(samplesPerPixel));
87             samples2D.emplace_back(std::vector<Point2d>(samplesPerPixel));
88         }
89     }
90
91     PixelSampler(int _sppSqrt, int _nDimensions)
92         : sppSqrt(_sppSqrt) ,samplesPerPixel(_sppSqrt * _sppSqrt), nDimensions(_nDimensions),
    curSamplePixelIndex(0), curDimensionIndex1D(0), curDimensionIndex2D(0) {
93         for (int i = 0; i < nDimensions; ++i) {
94             samples1D.emplace_back(std::vector<double>(samplesPerPixel));
95             samples2D.emplace_back(std::vector<Point2d>(samplesPerPixel));
96         }
97     }
98
99     virtual ~PixelSampler() = default;
100
101     // @brief The sampler may change the sampling stragety at different pixel location ,record the
    current pixel location
102     // @param _pixelPositon location of current pixel
103     virtual void startPixel(const Point2i &_pixelPositon) override {
104         // record the position
105         pixelPosition = _pixelPositon;
106         // reset the sample index
107         curDimensionIndex1D = curDimensionIndex2D = curSamplePixelIndex = 0;
108         // generate the samples will be used in current pixel
109         for (int i = 0; i < nDimensions; ++i) {
110             generateSamples1D(samples1D[i]);
```

```
111             shuffle(samples1D[i], rng);
112             generateSamples2D(samples2D[i]);
113             shuffle(samples2D[i], rng);
114         }
115     }
116
117     // @brief Start the next sample at a specific pixel
118     virtual void nextSample() override {
119         ++curSamplePixelIndex;
120         curDimensionIndex1D = curDimensionIndex2D = 0;
121     }
122
123     // @brief Sample a double in [0, 1], if curDimension exceeds the nDimensions, just return rng()
124     // @return A double sample
125     virtual double sample1D() override {
126         if (curDimensionIndex1D < nDimensions)
127             return samples1D[curDimensionIndex1D++][curSamplePixelIndex];
128         else
129             return rng();
130     }
131
132     // @brief Sample a point2d in [0, 1]^2, if curDimension exceeds the nDimensions, just return {rng(),
      rng()}
133     // @return A point2d sample
134     virtual Point2d sample2D() override {
135         if (curDimensionIndex2D < nDimensions)
136             return samples2D[curDimensionIndex2D++][curSamplePixelIndex];
137         else
138             return Point2d{rng(), rng()};
139     }
140 };
```

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.

```
1
13 #pragma once
14 #include "Sampler.h"
15
16 class StratifiedSampler : public PixelSampler {
17     // fill the samples1D
18     virtual void generateSamples1D(std::vector<double> &samples) override;
19     // fill the samples2D
20     virtual void generateSamples2D(std::vector<Point2d> &samples) override;
21 public:
22     StratifiedSampler() = delete;
23
24     StratifiedSampler(int sppSqrt) : PixelSampler(sppSqrt) { }
25
26     StratifiedSampler(int sppSqrt, int _nDimensions) : PixelSampler(sppSqrt, _nDimensions) { }
27
28     ~StratifiedSampler() = default;
29
30 };
```

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

9.106 Entity.h

Go to the documentation of this file.

```
1
12 #pragma once
13
14 #include "CoreLayer/Geometry/Transform3d.h"
15 #include "FunctionLayer/Material/Material.h"
16 #include "CoreLayer/Geometry/BoundingBox.h"
17 #include "CoreLayer/Ray/Ray.h"
18
19 #include <optional>
20 #include <memory>
21
22 struct Intersection;
23 class Light;
24
25 class Entity : public Transform3D
26 {
27 public:
28     std::shared_ptr<Light> lightPtr;
29     std::shared_ptr<Material> material;
30     //@brief Returns the intersection of the entity and the ray
31     virtual std::optional<Intersection> intersect(const Ray &r) const = 0;
32     //@brief Return ptr to light when primitive is a emitter. Otherwise, return nullptr.
33     virtual std::shared_ptr<Light> getLight() const = 0;
34     virtual void setLight(std::shared_ptr<Light> light) = 0;
35     virtual double area() const = 0;
36     virtual Intersection sample(const Point2d &positionSample) const = 0;
37     //@brief Return the bounding box of the entity
38     virtual BoundingBox3f WorldBound() const = 0;
39 };
```

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.

```
1
12 #pragma once
13
14 #include "Entity.h"
15 #include <optional>
16
17 class Sphere : public Entity
18 {
19 protected:
20     double radius;
21     Point3d center;
22     virtual void apply() override;
23
24 public:
25     Sphere(Point3d _center, double _radius, std::shared_ptr<Material> _material);
26     virtual std::optional<Intersection> intersect(const Ray &r) const;
27     virtual double area() const;
28     virtual Intersection sample(const Point2d &positionSample) const;
29     virtual std::shared_ptr<Light> getLight() const;
30     virtual void setLight(std::shared_ptr<Light> light);
31     virtual BoundingBox3f WorldBound() const;
32 };
```

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.

```
1
13 #pragma once
14
15 #include "Entity.h"
16 #include <optional>
17
18 class TriangleMesh{
19 public:
20     const int nTriangles;
21     const int nVertices;
22     std::shared_ptr<std::vector<int» vertexIndices;//the i-th triangle maps vertex[i * 3], vertex[i * 3 +
       1], vertex[i * 3 + 2]
23     std::shared_ptr<std::vector<Point3d» p; //geometry position
24     std::shared_ptr<std::vector<Normal3d» n; //shading normal
25     std::shared_ptr<std::vector<Vec3d» s; //shading tangent(optional)
26     std::shared_ptr<std::vector<Point2d» uv; //uv coordinates
27     std::shared_ptr<Material> material;
28     /*
29     @brief constructor of TriangleMesh, parameter pointers(except material) will point to nullptr after
       construction
30     @param <_nTriangles> number of triangle
31     @param <_nVertices> number of vertices(equal to the sizeof of _p)
32     @param <_p> position of vertices
33     @param <_n> shading normal of vertices(optional)
34     @param <_s> shading tangent of vertices(optional)
35     @param <_uv> uv coordinates of vertices(optional)
36     @param <_material> material of triangle mesh
37     */
38     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);
39 };
40
41 //vertices in counter-clockwise order
42 class Triangle: public Entity{
43 protected:
44     int vertexId[3];
45     int faceId;
46     std::shared_ptr<TriangleMesh> mesh;
47 public:
48     virtual void apply() override;
49     Triangle(const std::shared_ptr<TriangleMesh>& _mesh, const int& _faceId);
50     Triangle(const std::vector<Point3d>& points, const std::shared_ptr<Material>& _material);
51     virtual std::optional<Intersection> intersect(const Ray& r) const;
52     virtual double area() const;
53     virtual Intersection sample(const Point2d& positionSample) const;
54     virtual std::shared_ptr<Light> getLight() const;
55     virtual void setLight(std::shared_ptr<Light> light);
56     virtual BoundingBox3f WorldBound() const;
57 };
```

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

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

9.115 ImageTexture.h

Go to the documentation of this file.

```
1
12 #pragma once
13
14 #include <cmath>
15 #include "CoreLayer/ColorSpace/Color.h"
16 #include "ResourceLayer/File/Image.h"
17 #include "FunctionLayer/Intersection.h"
18 #include "Texture.h"
19 #include "TextureMapping.h"
20
21 enum class WrapMode
22 {
23     // todo
24 };
25
26 template <typename T>
27 class PrefilteredImage
28 {
29 protected:
30     WrapMode wrapMode;
31
32 public:
33     virtual void setWrapMode(enum WrapMode _wrapMode);
34     virtual WrapMode getWrapMode();
35     // todo: other common parameters
36
37     virtual T eval(const TextureCoord2D &coord) = 0;
38     virtual T texel(const Point2i &coord) = 0;
39     virtual void loadImage(const std::string &filename) = 0;
40 };
41
42 template <typename T>
43 class DirectImage : public PrefilteredImage<T>
44 {
45 protected:
46     std::shared_ptr<Image> image;
47
48 public:
49     virtual void loadImage(const std::string &filename);
50     virtual T eval(const TextureCoord2D &coord);
51     virtual T texel(const Point2i &coord);
52 };
53
54 template <typename T>
55 class LinearMIPMap : public PrefilteredImage<T>
56 {
57 protected:
58     std::shared_ptr<Image> image;
59
60 public:
61     virtual void loadImage(const std::string &filename);
62     virtual T eval(const TextureCoord2D &coord);
63     virtual T texel(const Point2i &coord);
64 };
65
66 template <typename Treturn, typename Tmemory>
67 class ImageTexture : public StdTexture<Treturn, TextureCoord2D>
68 {
69 protected:
70     std::shared_ptr<PrefilteredImage<Tmemory» imageSampler;
71
72 public:
73     ImageTexture(const std::string &filename,
74                  std::shared_ptr<TextureMapping2D> mapping = std::make_shared<UVTextureMapping2D>()); //
      using default sampler
75     ImageTexture(const std::string &filename,
76                  std::shared_ptr<PrefilteredImage<Tmemory» imageSampler,
77                  std::shared_ptr<TextureMapping2D> mapping = std::make_shared<UVTextureMapping2D>());
78     virtual Treturn eval(const TextureCoord2D &coord) const override;
79 };
80
81 // * Example: Creating a Image-based Color Texture using UV coordinates from mesh
82 // * >  ImageTexture<RGB3>("1.jpg");
83 // * >  ImageTexture<RGB3>("1.jpg", std::make_shared<DirectSampler>());
84 // * >  ImageTexture<RGB3>("1.jpg", std::make_shared<UVTextureMapping2D>());
85 // * >  ImageTexture<RGB3>("1.jpg", std::make_shared<DirectSampler>(),
      std::make_shared<UVTextureMapping2D>());
86
87 // * Example: Create a Image-based Normal Map (wip)
88 // * since normal cannot be directly interpolated, you need to provide T with some compact NDF type
89 // * The NDF must provide convertor from RGB3 and some accessors for shadings (depends on implementation
      of material)
```

```
90
91 template <typename T>
92 void PrefilteredImage<T>::setWrapMode(enum WrapMode _wrapMode)
93 {
94     wrapMode = _wrapMode;
95 }
96
97 template <typename T>
98 WrapMode PrefilteredImage<T>::getWrapMode()
99 {
100     return wrapMode;
101 }
102
103 template <typename T>
104 void DirectImage<T>::loadImage(const std::string &filename)
105 {
106     image = std::make_shared<Image>(filename);
107 }
108
109 template <typename T>
110 T DirectImage<T>::texel(const Point2i &coord)
111 {
112     // todo
113     return 0.0;
114 }
115
116 template <typename Treturn, typename Tmemory>
117 ImageTexture<Treturn, Tmemory>::ImageTexture(const std::string &filename,
118                                              std::shared_ptr<TextureMapping2D> mapping) :
    StdTexture<Treturn, TextureCoord2D>(mapping)
119 {
120     imageSampler = std::make_shared<DirectImage<Tmemory>>();
121     imageSampler->loadImage(filename);
122 }
123
124
125
126 template <typename Treturn, typename Tmemory>
127 ImageTexture<Treturn, Tmemory>::ImageTexture(const std::string &filename,
128                                              std::shared_ptr<PrefilteredImage<Tmemory>> imageSampler,
129                                              std::shared_ptr<TextureMapping2D> mapping) :
    imageSampler(imageSampler), StdTexture<Treturn, TextureCoord2D>(mapping)
130 {
131     this->imageSampler->loadImage(filename);
132 }
```

9.116 /mnt/renderer/Zero/src/FunctionLayer/Texture/ProceduralTexture.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

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

9.118 ProceduralTexture.h

Go to the documentation of this file.

```
1
13 #include "Texture.h"
14
15 class Checkerboard2D : public StdTexture<double, TextureCoord2D>
16 {
17 protected:
18 public:
19     Checkerboard2D();
20     using StdTexture::eval;
21     virtual double eval(const TextureCoord2D &coord) const override;
22 };
23
24 class Checkerboard3D : public StdTexture<double, TextureCoord3D>
25 {
26 protected:
27 public:
28     Checkerboard3D();
29     using StdTexture::eval;
30     virtual double eval(const TextureCoord3D &coord) const override;
31 };
```

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

9.120 Texture.h

Go to the documentation of this file.

```
1
12 #pragma once
13
14 #include <cmath>
15 #include "FunctionLayer/Intersection.h"
16 #include "CoreLayer/Geometry/Geometry.h"
17
18 // Tvalue can be float or any specturm type
19 template <typename Tvalue>
20 class Texture
21 {
22 public:
23     virtual Tvalue eval(const Intersection &intersection) const = 0;
24 };
25
26
27 // @brief Struct for texture coord C and dC/dx, dC/dy.
28 template <typename Tpos, typename Tvec>
29 struct TextureCoord
30 {
31     Tpos coord;
32     Tvec dcdx;
33     Tvec dcdy;
34 };
35
36 typedef TextureCoord<Point2d, Vec2d> TextureCoord2D;
```

```
37 typedef TextureCoord<Point3d, Vec3d> TextureCoord3D;
38
39 // @brief TextureMapping maps an intersection to texture coordinates of type Tcoord.
40 template <typename Tcoord>
41 class TextureMapping
42 {
43 protected:
44 public:
45     virtual Tcoord mapping(const Intersection &intersection) const = 0;
46 };
47
48 typedef TextureMapping<TextureCoord2D> TextureMapping2D;
49 typedef TextureMapping<TextureCoord3D> TextureMapping3D;
50
51 template <typename Tvalue>
52 class ConstantTexture : public Texture<Tvalue>
53 {
54 protected:
55     Tvalue value;
56
57 public:
58     ConstantTexture(const Tvalue &value);
59     virtual Tvalue eval(const Intersection &intersection) const;
60 };
61
62 template <typename Tvalue>
63 class MixTexture : public Texture<Tvalue>
64 {
65 protected:
66     std::shared_ptr<Texture<Tvalue» srcA;
67     std::shared_ptr<Texture<Tvalue» srcB;
68     std::shared_ptr<Texture<double» factor;
69
70 public:
71     MixTexture(std::shared_ptr<Texture<Tvalue» srcA,
72                std::shared_ptr<Texture<Tvalue» srcB,
73                std::shared_ptr<Texture<double» factor);
74
75     virtual Tvalue eval(const Intersection &intersection) const;
76 };
77
78 // @brief StdTexture refers to textures that needs a texture mapping to generate Tcoord from Intersection
79 template <typename Tvalue, typename Tcoord>
80 class StdTexture : public Texture<Tvalue>
81 {
82 protected:
83     std::shared_ptr<TextureMapping<Tcoord» mapping;
84
85 public:
86     StdTexture();
87     StdTexture(std::shared_ptr<TextureMapping<Tcoord» mapping);
88
89     // @brief This function just redirects the query to eval(coord) using member TextureMapping. Derived
       should NOT overwrite this.
90     virtual Tvalue eval(const Intersection &intersection) const final;
91
92     // @brief Eval texture value at given texture coord. (Derived needs to implement this)
93     virtual Tvalue eval(const Tcoord &coord) const = 0;
94 };
95
96 template <typename Tvalue>
97 ConstantTexture<Tvalue>::ConstantTexture(const Tvalue &value) : value(value)
98 {
99 }
100
101 template <typename Tvalue>
102 Tvalue ConstantTexture<Tvalue>::eval(const Intersection &intersection) const
103 {
104     return value;
105 }
106
107 template <typename Tvalue>
108 MixTexture<Tvalue>::MixTexture(std::shared_ptr<Texture<Tvalue» srcA,
109                                std::shared_ptr<Texture<Tvalue» srcB,
110                                std::shared_ptr<Texture<double» factor) : srcA(srcA), srcB(srcB),
       factor(factor)
111 {
112 }
113
114 template <typename Tvalue>
115 Tvalue MixTexture<Tvalue>::eval(const Intersection &intersection) const
116 {
117     double alpha = factor->eval(intersection);
118     return srcA->eval(intersection) * alpha + srcB->eval(intersection) * (1 - alpha);
119 }
120
121 template <typename Tvalue, typename Tcoord>
```

```
122 StdTexture<Tvalue, Tcoord>::StdTexture(std::shared_ptr<TextureMapping<Tcoord» mapping): mapping(mapping)
123 {
124 }
125
126 template <typename Tvalue, typename Tcoord>
127 Tvalue StdTexture<Tvalue, Tcoord>::eval(const Intersection &intersection) const
128 {
129     return eval(mapping->mapping(intersection));
130 }
```

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

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.

```
1
12 #pragma once
13
14 #include <cmath>
15 #include "CoreLayer/ColorSpace/Color.h"
16 #include "Texture.h"
17 #include "FunctionLayer/Intersection.h"
18 // Various kinds of TextureMapping
19
20 class UVTextureMapping2D : public TextureMapping<TextureCoord2D>
21 {
22 protected:
23     // todo: add a affine transform
24 public:
25     virtual TextureCoord2D mapping(const Intersection &intersection) const override;
26 };
27
28 class NaturalTextureMapping3D : public TextureMapping<TextureCoord3D>
29 {
30 protected:
31     // todo: add a affine transform
32 public:
33     virtual TextureCoord3D mapping(const Intersection &intersection) const override;
34 };
```

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.

```
1
12 #pragma once
13
14 #include "TileGenerator.h"
15
16 class SequenceTileGenerator : public TileGenerator
17 {
18
19 private:
20
21     std::vector<int> xList;
22     std::vector<int> yList;
23
24     Point2i currentBeginIndex;
25     // currentEndIndex=[currentBeginIndex.x+1,currentBeginIndex.y+1]
26     bool reachedEnd=false;
27
28 protected:
29
30     //@brief the size of a tile. The default size of a tile is 16x16.
31     int size;
32
33 public:
34
35     SequenceTileGenerator(const Point2i& _resolution,int _size=16);
36
37     virtual std::vector<std::shared_ptr<Tile» generateTiles();
38
39     virtual std::optional<std::shared_ptr<Tile» generateNextTile();
40
41 };
```

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

```
1
12 #pragma once
13
14 #include "CoreLayer/Geometry/Geometry.h"
15
16 #include <vector>
17 #include <mutex>
18 #include <optional>
19
20 /*@brief Iterator for Point2i.
21 * It will generate points between pBegin and pEnd but pEnd is not included.
22 * e.g, pBegin=[0,0], pEnd=[2,2], and the generated points will be
23 * [0,0], [1,0], [0,1] and [1,1].
24 * */
25 class PointIterator
26 {
27
28 protected:
29
30     // @brief the first point of a tile.
31     Point2i pBegin;
32
33     // @brief the last point of a tile.
34     Point2i pEnd;
35
```

```
36      // @brief current point in tile.
37      Point2i currentP;
38
39      int xMin, xMax;
40      int yMin, yMax;
41
42  public:
43
44      PointIterator(const Point2i& _pBegin, const Point2i& _pEnd,const Point2i& p);
45
46      Point2i operator*() const;
47
48      PointIterator& operator++();
49
50      bool operator!=(const PointIterator& anotherIt);
51
52  };
53
54  /*
55  * @brief Tile representation, and generate points within the tile.
56  * Check PointIterator for more information.
57  */
58  class Tile
59  {
60
61  protected:
62
63      Point2i pBegin;
64      Point2i pEnd;
65
66  public:
67
68      Tile(const Point2i& _pBegin, const Point2i& _pEnd);
69
70      virtual PointIterator begin() const=0;
71      virtual PointIterator end() const=0;
72  };
73
74  /*
75  * @brief Basic Tile implemention. Generate points in a square.
76  */
77  class SquareTile : public Tile
78  {
79  private:
80
81      PointIterator beginIte;
82      PointIterator endIte;
83  public:
84
85      SquareTile(const Point2i& _pBegin, const Point2i& _pEnd);
86
87      virtual PointIterator begin() const;
88      virtual PointIterator end() const;
89  };
90
91  class TileGenerator
92  {
93
94  protected:
95
96      // @brief the resolution of render image.
97      Point2i resolution;
98
99      // @brief lock for generateNextTile.
100      std::mutex mute;
101
102 public:
103
104     TileGenerator(const Point2i& _resolution);
105
106     // @brief generate all tiles. No mutex.
107     virtual std::vector<std::shared_ptr<Tile» generateTiles() = 0;
108
109     /*
110     * @brief generate next tile.
111     * Use mutex to make sure that threads will have different tiles.
112     */
113     virtual std::optional<std::shared_ptr<Tile» generateNextTile() = 0;
114
115 };
```

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.

```
1
12 #pragma once
13
14 #include "CoreLayer/Geometry/Geometry.h"
15 #include "CoreLayer/ColorSpace/Color.h"
16
17 #include <string>
18
19 // todo: support various data type
20
21 class Image
22 {
23     unsigned char *imageRawData;
24     Point2i resolution;
25     int channels;
26
27 public:
28     Image();
29     ~Image();
30
31     enum class ImageLoadMode
32     {
33         IMAGE_LOAD_BW,
34         IMAGE_LOAD_COLOR
35     };
36     // todo: support alpha reading
37     // todo: do gamma correction
38     Image(const std::string &path, ImageLoadMode = ImageLoadMode::IMAGE_LOAD_COLOR);
```

```
39
40      // @brief generate one black image with resolution [width,height] and channels.
41      Image(const Point2i &resolution, int channels);
42      Image(const Point3i &shape);
43
44      friend class ImageManager;
45
46      Point2i getResolution() const;
47      int getChannels() const;
48      int getWidth() const;
49      int getHeight() const;
50
51      void setColorAt(const Point2i &p, const Spectrum &s);
52      void setColorAt(const Point2i &p, const RGB3 &rgb);
53      RGB3 getRGBColorAt(const Point2i &p);
54      Spectrum getSpectrumColorAt(const Point2i &p);
55
56      bool saveTo(const std::string &path);
57  };
```

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

9.133 MeshData.h

Go to the documentation of this file.

```
1
12 #pragma once
13
14 #include "CoreLayer/Geometry/Geometry.h"
15
16 #include <string>
17
18 class MeshData
19 {
20     double *vertexRaw;
21     double *normalRaw;
22     double *uvRaw;
23     double *tangentRaw;
24     double *bitangentRaw;
25     int *indiceRaw;
26
27     // @brief init MeshData from raw data pointer. MeshData can not be initialized from file path cause
       one single file may cantain multiple MeshData.
28     MeshData(double *_v, double *_n, double *_uv, double *_tan, double *_bi, int *_indice);
29
30 public:
31     friend class MeshDataManager;
32
33     Point3d getVertexAt(int i) const;
34     Normal3d getNormalAt(int i) const;
35     Point2d getUvAt(int i) const;
36     Vec3d getTangentAt(int i) const;
37     Vec3d getBitangentAt(int i) const;
38
39     // @brief get 3 indices for ist triangle mesh. In order.
40     Point3i getTriangleIndiceAt(int i) const;
41
42     int getTriangleNum() const;
43 };
```

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

9.135 ResourceManager.h

Go to the documentation of this file.

```
1
12 #pragma once
13
14 #include "ResourceLayer/File/Image.h"
15 #include "ResourceLayer/File/MeshData.h"
16
17 #include <map>
18 #include <string>
19 #include <memory>
20
21 template <typename BaseType>
22 class ResourceManager
23 {
24 protected:
25     std::map<std::string, std::shared_ptr<BaseType» hash;
26
27 public:
28     ResourceManager();
29 };
30
31 class ImageManager : public ResourceManager<Image>
32 {
33     ImageManager();
34     std::shared_ptr<ImageManager> instance;
35
36 public:
37     // @brief singleton pattern get.
38     static std::shared_ptr<ImageManager> getInstance();
39
40     std::shared_ptr<Image> getImage(const std::string &path, Image::ImageLoadMode mode);
41 };
42
43 class MeshDataManager : public ResourceManager<MeshData>
44 {
45     MeshDataManager();
46     std::shared_ptr<MeshDataManager> instance;
47
48 public:
49     // @brief singleton pattern get.
50     static std::shared_ptr<MeshDataManager> getInstance();
51
52     std::vector<std::shared_ptr<MeshData» getMeshData(const std::string &path);
53 };
```