Library for Optical Systems Raytracing Reference Manual

0.2.1

Generated by Doxygen 1.2.8.1

Sat Dec 22 10:58:13 2001

Contents

1 Library for Optical Systems Raytracing Hierarchical Index 1

1.1 Library for Optical Systems Raytracing Class Hierarchy 1

2 Library for Optical Systems Raytracing Compound Index 3

2.1 Library for Optical Systems Raytracing Compound List 3

3 Library for Optical Systems Raytracing Class Documentation 5

3.1 LOSR::Analyzer Class Reference 5

3.2 LOSR::ApproxRaytracer Class Reference 13

3.3 LOSR::ExactRaytracer Class Reference 21

3.4 LOSR::OpticalSystem Class Reference 30

3.5 LOSR::Pupil Struct Reference 34

3.6 LOSR::QU Struct Reference 36

3.7 LOSR::Surface Class Reference 40

3.8 LOSR::Y_NU Struct Reference 44

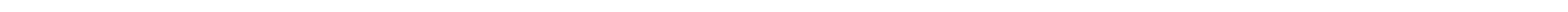
Chapter 1

Library for Optical Systems Raytracing Hierarchical Index

1.1 Library for Optical Systems Raytracing Class Hierarchy

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

- LOSR::Analyzer 5
- LOSR::Pupil 34
- LOSR::QU 36
- LOSR::Surface 40
- std::vector
  - LOSR::ApproxRaytracer 13
  - LOSR::ExactRaytracer 21
  - LOSR::OpticalSystem 30
- LOSR::Y_NU 44



Chapter 2

Library for Optical Systems Raytracing Compound Index

2.1 Library for Optical Systems Raytracing Compound List

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

LOSR::Analyzer (LOSR::Analyzer is an high level tool to analyze optical systems) . . . 5
LOSR::ApproxRaytracer (LOSR::ApproxRaytracer is an approximative raytracer table, using y-nu trace algorithms) . . . 13
LOSR::ExactRaytracer (LOSR::ExactRaytracer is an exact raytracer table, using QU trace algorithms) . . . 21
LOSR::OpticalSystem (LOSR::OpticalSystem is a on-axis lenses system made of optical materials) . . . 30
LOSR::Pupil (LOSR::Pupil describe the value of an entrance or exit pupil) . . . 34
LOSR::QU (LOSR::QU describe a surface entry into an exact raytracing table (QU trace)) . . . 36
LOSR::Surface (LOSR::Surface describe a sperical contact between two optical materials) . . . 40
LOSR::Y_NU (LOSR::Y_NU describe a surface entry into an approximative raytracing table (y-nu trace)) . . . 44

Chapter 3

Library for Optical Systems Raytracing Class Documentation

3.1 LOSR::Analyzer Class Reference

LOSR::Analyzer is an high level tool to analyze optical systems.

`#include <Analyzer.hpp>`

Public Methods

- Analyzer ()

  Constructor (do nothing).

- ~Analyzer ()

  Destructor (do nothing).

- ApproxRaytracer::const_iterator getApertureStop (const ApproxRaytracer &inAxialRay) const

  Get the aperture stop of a system using the axial y-nu trace.

- ApproxRaytracer::iterator getApertureStop (ApproxRaytracer &inAxialRay) const

  Get the aperture stop of a system using the axial y-nu trace.

- ApproxRaytracer::const_iterator getFieldStop (const ApproxRaytracer &inChiefRay) const

  Get the field stop of a system using the chief y-nu trace.

- ApproxRaytracer::iterator getFieldStop (ApproxRaytracer &inAxialRay) const

  Get the field stop of a system using the chief y-nu trace.

- Pupil getEntrancePupil (const ApproxRaytracer &inAxialRay, ApproxRaytracer::const_iterator inApertureStop, const ApproxRaytracer &inChiefRay) const

  Get the entrance pupil of a system using the axial and chief y-nu trace.

- Pupil getExitPupil (const ApproxRaytracer &inAxialRay, ApproxRaytracer::const_iterator inApertureStop, const ApproxRaytracer &inChiefRay) const

  Get the exit pupil of a system using the chief y-nu trace.

- double getMagnification (const ApproxRaytracer &inChiefRay) const

  Get the magnification of a system using the chief y-nu trace.

- double evaluateLastCurvature (const ApproxRaytracer &inMarginalRay) const

  Evaluation the last curvature of the optical system using the marginal y-nu trace.

- bool traceChiefRay (const ApproxRaytracer &inAxialRay, ApproxRaytracer::const_iterator inApertureStop, ApproxRaytracer &outChiefRay, long double inCY0=1.0, long double inCU0=0.0) const

  Trace an chief ray using an axial raytrace.

- bool traceMarginalRay (const ApproxRaytracer &inAxialRay, ApproxRaytracer::const_iterator inApertureStop, const ApproxRaytracer &inChiefRay, ApproxRaytracer &outMarginalRay) const

  Trace a marginal ray using an axial raytrace.

3.1.1 Detailed Description

LOSR::Analyzer is an high level tool to analyze optical systems.

LOSR::Analyzer is an high level tool to analyze optical systems. It provides different methods that analyze optical system using approximative raytrace tables, such marginal and chief ray.

Author:
Christian Gagné and Julie Beaulieu

Version:
0.2

Date:
9/05/2001

3.1.2 Member Function Documentation

3.1.2.1 double Analyzer::evaluateLastCurvature (const ApproxRaytracer & *inMarginalRay*) const

Evaluation the last curvature of the optical system using the marginal y-nu trace.

This method evaluate the value of the curvature of the last surface before the image space to get a clean image exactly on the image space. This mean that all the rays that are starting on the same point on the object will be on the point on the image (using approximative raytracing).

Returns:
Curvature of the last, before the image, curvature.

Parameters:
inMarginalRay Marginal ray from which the last curvature is determined.

```
00187 {
00188 #ifndef NDEBUG
00189   if(inMarginalRay.size()<2) throw LOSR_RuntimeErrorM("Marginal ray must have at least two entries!"
00190 #endif // NDEBUG
00191   ApproxRaytracer::const_iterator lIterJ   = inMarginalRay.end()-1;
00192   ApproxRaytracer::const_iterator lIterJM1 = inMarginalRay.end()-2;
00193   return ( (lIterJ->mY * lIterJ->mN) + (lIterJM1->mNU * lIterJ->mT) ) /
00194          ( lIterJ->mT * lIterJ->mY * (lIterJ->mN- lIterJM1->mN) );
00195 }
```

3.1.2.2 ApproxRaytracer::iterator Analyzer::getApertureStop (ApproxRaytracer & *inAxialRay*) const

Get the aperture stop of a system using the axial y-nu trace.

Returns:
Iterator to the aperture stop.

Parameters:

inAxialRay Axial ray from which the aperture stop is determined.

```
00081 {
00082   const ApproxRaytracer& lConstAxialRay = inAxialRay;
00083   ApproxRaytracer::const_iterator lConstApertureStop = getApertureStop(lConstAxialRay);
00084   return inAxialRay.begin() + ( lConstApertureStop - inAxialRay.begin() );
00085 }
```

3.1.2.3 ApproxRaytracer::const_iterator Analyzer::getApertureStop (const ApproxRaytracer & *inAxialRay*) const

Get the aperture stop of a system using the axial y-nu trace.

Returns:

Iterator to the aperture stop.

Parameters:

inAxialRay Axial ray from which the aperture stop is determined.

```
00055 {
00056 #ifndef NDEBUG
00057   if(inAxialRay.size() < 3) throw LOSR_RuntimeErrorM("The axial ray must have at least 3
00058 #endif // NDEBUG
00059   ApproxRaytracer::const_iterator lApertureStop = inAxialRay.begin()+1;
00060   double lMinimumRatio = DBL_MAX;
00061   for(ApproxRaytracer::const_iterator i = (inAxialRay.begin()+1); i != (inAxialRay.end()-
00062     if( (i->mR != 0) && (i->mY != 0) )  {
00063       double lRatio = fabs(i->mR / i->mY);
00064       if(lRatio < lMinimumRatio) {
00065         lMinimumRatio = lRatio;
00066         lApertureStop = i;
00067       }
00068     }
00069   }
00070   return lApertureStop;
00071 }
```

3.1.2.4 Pupil Analyzer::getEntrancePupil (const ApproxRaytracer & *inAxialRay*, ApproxRaytracer::const_iterator *inApertureStop*, const ApproxRaytracer & *inChiefRay*) const

Get the entrance pupil of a system using the axial and chief y-nu trace.

Returns:

Values of the entrance pupil.

Parameters:

inAxialRay Axial ray from which the entrance pupil is determined.

inApertureStop Iterator to the aperture stop of axial ray.

inChiefRay Chief ray from which the entrance pupil is determined.

```
00123 {
00124 #ifndef NDEBUG
00125   if(inChiefRay.size() < 2) throw LOSR_RuntimeErrorM("Chief ray must have at least two entry!");
00126   if(inAxialRay.size() != inChiefRay.size())
00127     throw LOSR_RuntimeErrorM("Axial ray and chief ray are not of the same size!");
00128 #endif // NDEBUG
00129   ApproxRaytracer::const_iterator lApStopChief =
00130     inChiefRay.begin() + (inApertureStop - inAxialRay.begin());
00131   Pupil lPupil;
00132   lPupil.mM = lApStopChief->mU / inChiefRay.front().mU;
00133   lPupil.mR = lApStopChief->mR * lPupil.mM;
00134   lPupil.mT = inChiefRay.front().mT - ( inChiefRay[1].mY / inChiefRay.front().mU );
00135   return lPupil;
00136 }
```

3.1.2.5 Pupil Analyzer::getExitPupil (const ApproxRaytracer & *inAxialRay*, ApproxRaytracer::const_iterator *inApertureStop*, const ApproxRaytracer & *inChiefRay*) const

Get the exit pupil of a system using the chief y-nu trace.

Returns:

Values of the exit pupil.

Parameters:

inAxialRay Axial ray from which the exit pupil is determined.

inApertureStop Iterator to the aperture stop of axial ray.

inChiefRay Chief ray from which the exit pupil is determined.

```
00148 {
00149 #ifndef NDEBUG
00150   if(inChiefRay.size() < 2) throw LOSR_RuntimeErrorM("Chief ray must have at least two entry!");
00151   if(inAxialRay.size() != inChiefRay.size())
00152     throw LOSR_RuntimeErrorM("Axial ray and chief ray are not of the same size!");
00153 #endif // NDEBUG
00154   ApproxRaytracer::const_iterator lApStopChief =
00155     inChiefRay.begin() + (inApertureStop - inAxialRay.begin());
00156   Pupil lPupil;
00157   lPupil.mM = lApStopChief->mU / inChiefRay.back().mU;
00158   lPupil.mR = lApStopChief->mR * lPupil.mM;
00159   lPupil.mT = (inChiefRay.end()-2)->mT + ( (inChiefRay.end()-2)->mY / (inChiefRay.end()-2)->mU );
00160   return lPupil;
00161 }
```

3.1.2.6 ApproxRaytracer::iterator Analyzer::getFieldStop (ApproxRaytracer & *inChiefRay*) const

Get the field stop of a system using the chief y-nu trace.

Returns:

Iterator to the field stop.

Parameters:

inChiefRay Chief ray from which the field stop is determined.

```
00107 {
00108   const ApproxRaytracer& lConstChiefRay = inChiefRay;
00109   ApproxRaytracer::const_iterator lConstFieldStop = getFieldStop(lConstChiefRay);
00110   return inChiefRay.begin() + ( lConstFieldStop - inChiefRay.begin() );
00111 }
```

3.1.2.7 ApproxRaytracer::const_iterator Analyzer::getFieldStop (const ApproxRaytracer & *inChiefRay*) const

Get the field stop of a system using the chief y-nu trace.

Returns:

Iterator to the field stop.

Parameters:

inChiefRay Chief ray from which the field stop is determined.

```
00094        {
00095   // The algorithm is the same than getting the aperture stop. Only the names are differe
00096   return getApertureStop(inChiefRay);
00097 }
```

3.1.2.8 double Analyzer::getMagnification (const ApproxRaytracer & *inChiefRay*) const

Get the magnification of a system using the chief y-nu trace.

Returns:

Magnification of the system.

Parameters:

inChiefRay Chief ray from which the magnification is determined.

```
00171 {
00172   return inChiefRay.back().mY / inChiefRay.front().mY;
00173 }
```

3.1.2.9 bool Analyzer::traceChiefRay (const ApproxRaytracer & *inAxialRay*, ApproxRaytracer::const_iterator *inApertureStop*, ApproxRaytracer & *outChiefRay*, long double *inCY0* = 1.0, long double *inCU0* = 0.0) const

Trace an chief ray using an axial raytrace.

Note that one value of the inCY0 and inCU0 are used to calculate the optical invariant.

Returns:
Whether the axial ray was inside the apertures (true) or there was vigneting (false).

Parameters:

inAxialRay Axial ray from which the chief ray will be trace.

inApertureStop Iterator to the aperture stop of axial ray.

outChiefRay Resulting chief ray trace.

inCY0 Height of the chief ray at the object.

inCU0 Angle of the chief ray at the object.

```
00211 {
00212   outChiefRay = inAxialRay;
00213   ApproxRaytracer::iterator lApStopChief =
00214     outChiefRay.begin() + (inApertureStop - inAxialRay.begin());
00215
00216   // Evaluation the optical invariant to get the desired position at the object.
00217   long double lI  = ( inCY0 * outChiefRay.front().mN * outChiefRay.front().mU ) -
00218                   ( outChiefRay.front().mY * outChiefRay.front().mN * inCU0 );
00219   long double lCU = -lI / ( lApStopChief->mY * lApStopChief->mN );
00220
00221   // Doing chief ray trace y-nu du chief ray
00222   return outChiefRay.trace(lApStopChief,0.0,lCU);
00223 }
```

3.1.2.10 bool Analyzer::traceMarginalRay (const ApproxRaytracer & *inAxialRay*, ApproxRaytracer::const_iterator *inApertureStop*, const ApproxRaytracer & *inChiefRay*, ApproxRaytracer & *outMarginalRay*) const

Trace a marginal ray using an axial raytrace.

Returns:
Whether the axial ray was inside the apertures (true) or there was vigneting (false).

Parameters:

inAxialRay Axial ray from which the marginal ray will be trace.

inApertureStop Iterator to the aperture stop of axial ray.

inChiefRay Chief ray from which the marginal ray will be trace.

outMarginalRay Resulting marginal ray trace.

```
00236 {
00237   outMarginalRay = inAxialRay;
00238   Pupil lEntrancePupil = getEntrancePupil(inAxialRay,inApertureStop,inChiefRay);
00239   long double lU0 = lEntrancePupil.mR / lEntrancePupil.mT;
00240   return outMarginalRay.trace(outMarginalRay.begin(),0.0,lU0);
00241 }
```

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

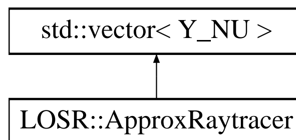
- Analyzer.hpp
- Analyzer.cpp

3.2 LOSR::ApproxRaytracer Class Reference

LOSR::ApproxRaytracer is an approximative raytracer table, using y-nu trace algorithms.

```
#include <ApproxRaytracer.hpp>
```

Inheritance diagram for LOSR::ApproxRaytracer::



Public Methods

- ApproxRaytracer (unsigned int inSize=0)

  Construct an approximative raytracer of the size given as argument.

- ApproxRaytracer (const LOSR::OpticalSystem &inOpticalSystem)

  Construct an approximative raytracer using an optical system.

- ~ApproxRaytracer ()

  Destructor (do nothing).

- ApproxRaytracer& operator= (const LOSR::OpticalSystem &inOpticalSystem)

  Reinitialize the raytracer using the optical system given as argument.

- void initialize (const LOSR::OpticalSystem &inOpticalSystem)

  Initialize the raytracer using the optical system given as argument.

- void read (std::istream &ioIs=std::cin)

  Read a y-nu trace from a C++ input stream.

- bool trace (ApproxRaytracer::iterator inStartingPoint)

  Do the y-nu trace, backward and forward from the starting point, using actual ray values.

- bool trace (ApproxRaytracer::iterator inStartingPoint, long double inY, long double inU)

Do the y-nu trace, backward and forward from the starting point, using ray values given.

- bool traceBackward (ApproxRaytracer::iterator inActualEntry, ApproxRaytracer::iterator inStopEntry)

  Do backward y-nu trace from actual entry to ending entry, using next entry values.

- bool traceForward (ApproxRaytracer::iterator inActualEntry, ApproxRaytracer::iterator inStopEntry)

  Do forward y-nu trace from actual entry to ending entry, using previous entry values.

- void write (std::ostream &ioOs=std::cout) const

  Write a y-nu trace into a C++ output stream.

3.2.1 Detailed Description

LOSR::ApproxRaytracer is an approximative raytracer table, using y-nu trace algorithms.

LOSR::ApproxRaytracer is an approximative raytracer table, using y-nu trace algorithms. It describes a table containing the entries of the y-nu table and the methods to do the traces, analysing it and manage it. An approximative raytracer usually take an optical system as input, before doing the ray trace. See chapter 5 of {\sl Elements of modern optical design}, Donald C. O'Shea, New York, Wiley, c1985, for details about the y-nu algorithm.

Author:
Christian Gagné and Julie Beaulieu

Version:
0.2

Date:
8/31/2001

3.2.2 Constructor & Destructor Documentation

3.2.2.1 ApproxRaytracer::ApproxRaytracer (unsigned int *inSize* = 0)

Construct an approximative raytracer of the size given as argument.

Parameters:
inSize Size of the y-nu trace.

```
00196   :
00197   std::vector<Y_NU>(inSize)
00198 { }
```

3.2.2.2 ApproxRaytracer::ApproxRaytracer (const LOSR::OpticalSystem & *inOpticalSystem*)

Construct an approximative raytracer using an optical system.

Parameters:

inOpticalSystem Optical system to use to build the raytracer.

```
00206   :
00207   std::vector<Y_NU>(inOpticalSystem.size())
00208 {
00209   initialize(inOpticalSystem);
00210 }
```

3.2.3 Member Function Documentation

3.2.3.1 void ApproxRaytracer::initialize (const LOSR::OpticalSystem & *inOpticalSystem*)

Initialize the raytracer using the optical system given as argument.

Parameters:

inOpticalSystem Optical system to use to build the system.

```
00232 {
00233   resize(inOpticalSystem.size());
00234   for(unsigned int i=0; i<size(); i++) (*this)[i] = inOpticalSystem[i];
00235 }
```

3.2.3.2 ApproxRaytracer & ApproxRaytracer::operator= (const LOSR::OpticalSystem & *inOpticalSystem*)

Reinitialize the raytracer using the optical system given as argument.

Returns:

Actual approximative raytracer.

Parameters:

inOpticalSystem Optical system to use to build the raytracer.

```
00220 {
00221   initialize(inOpticalSystem);
00222   return *this;
00223 }
```

3.2.3.3 void ApproxRaytracer::read (std::istream & *ioIs* = std::cin)

Read a y-nu trace from a C++ input stream.

Parameters:

ioIs Input stream to read the trace from.

```
00244 {
00245   char c1 ='\0', c2='\0';
00246   unsigned int sz = 0;
00247   char buf[2048];
00248
00249   // Extract and ignore the rest of lines starting with a #
00250   for(ioIs >> c1; c1=='#'; ioIs >> c1) ioIs.getline(buf,2048,'\n');
00251   if(!ioIs) throw LOSR_RuntimeErrorM("Premature end of stream!");
00252
00253   // Looking for the number of surface, between parenthesis
00254   ioIs >> sz >> c2;
00255   if((c1!='(') || (c2!=')')) throw LOSR_RuntimeErrorM("Bad file format!");
00256   resize(sz);
00257
00258   // Read each surfaces from the stream
00259   for(unsigned int i=0; i<sz; i++) {
00260     if(!ioIs) throw LOSR_RuntimeErrorM("Premature end of stream!");
00261     ioIs >> (*this)[i];
00262   }
00263 }
```

3.2.3.4 bool ApproxRaytracer::trace (ApproxRaytracer::iterator *inStartingPoint*, long double *inY*, long double *inU*)

Do the y-nu trace, backward and forward from the starting point, using ray values given.

NOTE: The value of U in an y-nu (approximative) ray trace is not the same than the U in a QU (exact) ray trace (U_ynu is equivalent to the tangent of U_QU).

Returns:

True if there was not vignetting, false if the ray goes out of an aperture.

Parameters:

inStartingPoint y-nu entry to start the trace.

inY Ray height value at the starting point.

inU Ray angle value at the starting point.

```
00307 {
00308 #ifndef NDEBUG
00309   if(inStartingPoint == end())
00310   throw LOSR_RuntimeErrorM("Cannot trace from the ending iterator of a raytracer!");
00311 #endif // NDEBUG
00312   inStartingPoint->mY = inY;
00313   inStartingPoint->mU = inU;
00314   return trace(inStartingPoint);
00315 }
```

3.2.3.5 bool ApproxRaytracer::trace (ApproxRaytracer::iterator *inStartingPoint*)

Do the y-nu trace, backward and forward from the starting point, using actual ray values.

It is assumed that the ray values of the actual entry (y and u) are correctly set.

Returns:

True if there was not vignetting, false if the ray goes out of an aperture.

Parameters:

inStartingPoint y-nu entry to start the trace.

```
00275 {
00276 #ifndef NDEBUG
00277   if(inStartingPoint == end())
00278     throw LOSR_RuntimeErrorM("Cannot trace from the ending iterator of a raytracer!");
00279 #endif  // NDEBUG
00280
00281   // Evaluate NU
00282   inStartingPoint->mNU  = inStartingPoint->mU * inStartingPoint->mN;
00283
00284   bool lResponse = true;
00285   if(inStartingPoint != begin()) {
00286     if(traceBackward(inStartingPoint-1,begin()) == false) lResponse = false;
00287   }
00288   if(inStartingPoint != end()-1) {
00289     if(traceForward(inStartingPoint+1,end()) == false) lResponse = false;
00290   }
00291   return lResponse;
00292 }
```

3.2.3.6 bool ApproxRaytracer::traceBackward (ApproxRaytracer::iterator *inActualEntry*, ApproxRaytracer::iterator *inStopEntry*)

Do backward y-nu trace from actual entry to ending entry, using next entry values.

The actual y-nu entry ray values must be correctly set before the call.

Returns:
True if there was not vignetting, false if the ray goes out of an aperture.

Parameters:
inActualEntry y-nu entry from which to do the trace.

inStopEntry y-nu entry to stop the trace.

```
00328 {
00329 #ifndef NDEBUG
00330   // Misc tests and assertions
00331   if(inActualEntry == end())
00332     throw LOSR_RuntimeErrorM("Cannot trace the ending iterator of a raytracer!");
00333   if(inActualEntry == end()-1)
00334     throw LOSR_RuntimeErrorM("Cannot backtrace the previous-to-ending iterator of a raytr
00335   if((inActualEntry == begin()) && (inActualEntry != inStopEntry))
00336     throw LOSR_RuntimeErrorM("The stop entry iterator is not reacheable!");
00337 #endif  // NDEBUG
00338
00339   // Get an iterator to the next entry
00340   iterator lNextEntry = inActualEntry+1;
00341
00342   // Apply the y-nu equations
00343   inActualEntry->mNU = lNextEntry->mNU +
00344     ( lNextEntry->mY * lNextEntry->mC * ( lNextEntry->mN - inActualEntry->mN ) );
00345   inActualEntry->mU  = inActualEntry->mNU / inActualEntry->mN;
00346   inActualEntry->mY  = lNextEntry->mY -
00347     ( inActualEntry->mNU * inActualEntry->mT / inActualEntry->mN );
00348
00349   // Recursive call to do the y-nu trace
00350   bool lResponse = true;
00351   if(inActualEntry != inStopEntry) lResponse = traceBackward(inActualEntry-1,inStopEntry)
00352   if(std::fabs(inActualEntry->mY) > std::fabs(inActualEntry->mR)) return false;
00353   return lResponse;
00354 }
```

3.2.3.7 bool ApproxRaytracer::traceForward (ApproxRaytracer::iterator *inActualEntry*, ApproxRaytracer::iterator *inStopEntry*)

Do forward y-nu trace from actual entry to ending entry, using previous entry values.

The actual y-nu entry ray values must be correctly set before the call.

Returns:
True if there was not vignetting, false if the ray goes out of an aperture.

Parameters:
inActualEntry y-nu entry from which to do the trace.
inStopEntry y-nu entry to stop the trace.

```
00367 {
00368 #ifndef NDEBUG
00369   // Misc tests and assertions
00370   if(inActualEntry == end())
00371     throw LOSR_RuntimeErrorM("Cannot trace the ending iterator of a raytracer!");
00372   if(inActualEntry == begin())
00373     throw LOSR_RuntimeErrorM("Cannot forward trace the begining iterator of a raytracer!");
00374   if((inActualEntry == end()-1) && (inActualEntry+1 != inStopEntry))
00375     throw LOSR_RuntimeErrorM("The stop entry iterator is not reacheable!");
00376 #endif // NDEBUG
00377
00378   // Get an iterator to the previous entry
00379   iterator lPreviousEntry = inActualEntry-1;
00380
00381   // Apply the y-nu equations
00382   inActualEntry->mY = lPreviousEntry->mY +
00383     ( lPreviousEntry->mT * ( lPreviousEntry->mNU / lPreviousEntry->mN ) );
00384   inActualEntry->mNU = lPreviousEntry->mNU -
00385     ( inActualEntry->mY * inActualEntry->mC * ( inActualEntry->mN - lPreviousEntry->mN ) );
00386   inActualEntry->mU = inActualEntry->mNU / inActualEntry->mN;
00387
00388   // Recursive call to do the y-nu trace
00389   bool lResponse = true;
00390   if(inActualEntry+1 != inStopEntry) lResponse = traceForward(inActualEntry+1,inStopEntry);
00391   if(std::fabs(inActualEntry->mY) > std::fabs(inActualEntry->mR)) return false;
00392   return lResponse;
00393 }
```

3.2.3.8 void ApproxRaytracer::write (std::ostream & *ioOs* = std::cout) const

Write a y-nu trace into a C++ output stream.

Parameters:
ioOs Output stream to write the trace into.

```
00402 {
00403   ioOs << '(' << size() << ')' << std::endl;
00404   for(unsigned int i=0; i<size(); i++) {
00405     ioOs << (*this)[i];
00406   }
00407 }
```

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

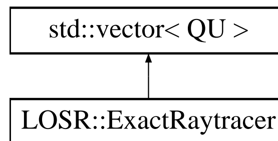
- ApproxRaytracer.hpp
- ApproxRaytracer.cpp

3.3 LOSR::ExactRaytracer Class Reference

LOSR::ExactRaytracer is an exact raytracer table, using QU trace algorithms.

`#include <ExactRaytracer.hpp>`

Inheritance diagram for LOSR::ExactRaytracer::



Public Methods

- ExactRaytracer (unsigned int inSize=0)

  Construct an exact raytracer of the size given as argument.

- ExactRaytracer (const LOSR::OpticalSystem &inOpticalSystem)

  Construct an exact raytracer using an optical system.

- ~ExactRaytracer ()

  Destructor (do nothing).

- ExactRaytracer& operator= (const LOSR::OpticalSystem &inOpticalSystem)

  Reinitialize the raytracer using the optical system given as argument.

- void initialize (const LOSR::OpticalSystem &inOpticalSystem)

  Initialize the raytracer using the optical system given as argument.

- void read (std::istream &ioIs=std::cin)

  Read a QU trace from a C++ input stream.

- bool trace (ExactRaytracer::iterator inStartingPoint)

  Do the QU trace, backward and forward from the starting point, using actual ray values.

- bool trace (ExactRaytracer::iterator inStartingPoint, long double inY, long double inU)

  Do the QU trace, backward and forward from the object space, using ray values given.

- bool traceBackward (ExactRaytracer::iterator inActualEntry, ExactRaytracer::iterator inStopEntry)

  Do backward QU trace from actual entry to ending entry, using next entry values.

- bool traceForward (ExactRaytracer::iterator inActualEntry, ExactRaytracer::iterator inStopEntry)

  Do forward QU trace from actual entry to ending entry, using previous entry values.

- void write (std::ostream &ioOs=std::cout) const

  Write a QU trace into a C++ output stream.

3.3.1 Detailed Description

LOSR::ExactRaytracer is an exact raytracer table, using QU trace algorithms.

LOSR::ExactRaytracer is an exact raytracer table, using QU trace algorithms. It describes a table containing the entries of the QU table and the methods to do the traces, analysing it and manage it. An exact raytracer usually take an optical system as input, before doing the ray trace. See chapter 6 of {\sl Elements of modern optical design}, Donald C. O'Shea, New York, Wiley, c1985, for details about the QU algorithm.

Author:
Christian Gagné and Julie Beaulieu

Version:
0.2

Date:
9/03/2001

3.3.2 Constructor & Destructor Documentation

3.3.2.1 ExactRaytracer::ExactRaytracer (unsigned int *inSize* = 0)

Construct an exact raytracer of the size given as argument.

Parameters:
inSize Size of the QU trace.

```
00349   :
00350   std::vector<QU>(inSize)
00351 { }
```

3.3.2.2 ExactRaytracer::ExactRaytracer (const LOSR::OpticalSystem & *inOpticalSystem*)

Construct an exact raytracer using an optical system.

Parameters:

inOpticalSystem Optical system to use to build the raytracer.

```
00359   :
00360   std::vector<QU>(inOpticalSystem.size())
00361 {
00362   initialize(inOpticalSystem);
00363 }
```

3.3.3 Member Function Documentation

3.3.3.1 void ExactRaytracer::initialize (const LOSR::OpticalSystem & *inOpticalSystem*)

Initialize the raytracer using the optical system given as argument.

Parameters:

inOpticalSystem Optical system to use to build the system.

```
00385 {
00386   resize(inOpticalSystem.size());
00387   for(unsigned int i=0; i<size(); i++) (*this)[i] = inOpticalSystem[i];
00388 }
```

3.3.3.2 ExactRaytracer & ExactRaytracer::operator= (const LOSR::OpticalSystem & *inOpticalSystem*)

Reinitialize the raytracer using the optical system given as argument.

Returns:

Actual exact raytracer.

Parameters:

inOpticalSystem Optical system to use to build the raytracer.

```
00373 {
00374   initialize(inOpticalSystem);
00375   return *this;
00376 }
```

3.3.3.3 void ExactRaytracer::read (std::istream & *ioIs* = std::cin)

Read a QU trace from a C++ input stream.

Parameters:

ioIs Input stream to read the trace from.

```
00397 {
00398   char c1 ='\0', c2='\0';
00399   unsigned int sz = 0;
00400   char buf[2048];
00401
00402   // Extract and ignore the rest of lines starting with a #
00403   for(ioIs >> c1; c1=='#'; ioIs >> c1) ioIs.getline(buf,2048,'\n');
00404   if(!ioIs) throw LOSR_RuntimeErrorM("Premature end of stream!");
00405
00406   // Looking for the number of surface, between parenthesis
00407   ioIs >> sz >> c2;
00408   if((c1!='(') || (c2!=')')) throw LOSR_RuntimeErrorM("Bad file format!");
00409   resize(sz);
00410
00411   // Read each surfaces from the stream
00412   for(unsigned int i=0; i<sz; i++) {
00413     if(!ioIs) throw LOSR_RuntimeErrorM("Premature end of stream!");
00414     ioIs >> (*this)[i];
00415   }
00416 }
```

3.3.3.4 bool ExactRaytracer::trace (ExactRaytracer::iterator *inStartingPoint*, long double *inY*, long double *inU*)

Do the QU trace, backward and forward from the object space, using ray values given.

Returns:

True if there was no total internal reflexion, false if there was.

Parameters:

inStartingPoint QU entry to start the trace.

inY Ray height value at the object space.

inU Ray angle value at the object space.

```
00492 {
00493 #ifndef NDEBUG
00494   if(begin() == end())
00495     throw LOSR_RuntimeErrorM("Cannot trace from an empty raytracer!");
00496   if(inStartingPoint == end())
00497     throw LOSR_RuntimeErrorM("Cannot trace from the ending iterator of a raytracer!");
00498 #endif // NDEBUG
```

```
00499   inStartingPoint->mY    = inY;
00500   inStartingPoint->mSinU = sin(inU);
00501   inStartingPoint->mCosU = cos(inU);
00502   inStartingPoint->mQ    = (inY * inStartingPoint->mCosU) +
00503     (inStartingPoint->mT * inStartingPoint->mSinU);
00504   return trace(inStartingPoint);
00505 }
```

3.3.3.5 bool ExactRaytracer::trace (ExactRaytracer::iterator *inStartingPoint*)

Do the QU trace, backward and forward from the starting point, using actual ray values.

It is assumed that the ray values of the actual entry (Q, sinU and cosU) are correctly set.

Returns:

True if there was no total internal reflexion, false if there was.

Parameters:

inStartingPoint QU entry to start the trace.

```
00428 {
00429 #ifndef NDEBUG
00430   if(inStartingPoint == end())
00431     throw LOSR_RuntimeErrorM("Cannot trace from the ending iterator of a raytracer!");
00432 #endif  // NDEBUG
00433
00434   // Evaluate the values of the actual entry
00435   inStartingPoint->mSinI   = ( inStartingPoint->mQ * inStartingPoint->mC ) + inStartingPoint->mSinU;
00436   inStartingPoint->mCosI   = sqrt( 1.0 - (inStartingPoint->mSinI * inStartingPoint->mSinI) );
00437   // inStartingPoint->mCosI   = cos( asin(inStartingPoint->mSinI) );
00438   inStartingPoint->mSinI_U = ( inStartingPoint->mSinI * inStartingPoint->mCosU ) -
00439     ( inStartingPoint->mCosI * inStartingPoint->mSinU );
00440   inStartingPoint->mCosI_U = ( inStartingPoint->mCosI * inStartingPoint->mCosU ) +
00441     ( inStartingPoint->mSinU * inStartingPoint->mSinI );
00442
00443   if(inStartingPoint != begin()) {
00444     inStartingPoint->mSinIp = inStartingPoint->mSinI * (inStartingPoint-1)->mN / inStartingPoint->mN
00445   } else {
00446     inStartingPoint->mSinIp = inStartingPoint->mSinI;
00447   }
00448
00449   inStartingPoint->mCosIp = sqrt( 1.0 - (inStartingPoint->mSinIp * inStartingPoint->mSinIp) );
00450   // inStartingPoint->mCosIp   = cos( asin(inStartingPoint->mSinIp) );
00451   inStartingPoint->mSinUp = ( inStartingPoint->mCosI_U * inStartingPoint->mSinIp ) -
00452     ( inStartingPoint->mSinI_U * inStartingPoint->mCosIp );
00453   inStartingPoint->mCosUp = ( inStartingPoint->mCosI_U * inStartingPoint->mCosIp ) +
00454     ( inStartingPoint->mSinI_U * inStartingPoint->mSinIp );
00455   inStartingPoint->mQp    = inStartingPoint->mQ *
00456     ( inStartingPoint->mCosUp + inStartingPoint->mCosIp ) /
00457     ( inStartingPoint->mCosU + inStartingPoint->mCosI );
```

```
00458
00459   //inStartingPoint->mY = inStartingPoint->mQ * (1.0 + inStartingPoint->mCosI_U) /
00460   //  (inStartingPoint->mCosU + inStartingPoint->mCosI);
00461   inStartingPoint->mY = (inStartingPoint->mQ - (inStartingPoint->mT * inStartingPoint->mS
00462     (inStartingPoint->mCosU);
00463   inStartingPoint->mU = atan2(inStartingPoint->mSinU,inStartingPoint->mCosU);
00464
00465   // Do QU for the system
00466   bool lResponse = true;
00467   if((std::fabs(inStartingPoint->mSinI) > 1.0) || (std::fabs(inStartingPoint->mSinU) > 1
00468      (std::fabs(inStartingPoint->mCosI) > 1.0) || (std::fabs(inStartingPoint->mCosU) > 1
00469      (std::fabs(inStartingPoint->mSinIp) > 1.0) || (std::fabs(inStartingPoint->mSinUp) >
00470      (std::fabs(inStartingPoint->mCosIp) > 1.0) || (std::fabs(inStartingPoint->mCosUp) >
00471     lResponse = false;
00472   }
00473   if(inStartingPoint != begin()) {
00474     if(traceBackward(inStartingPoint-1,begin()) == false) lResponse = false;
00475   }
00476   if(inStartingPoint != end()-1) {
00477     if(traceForward(inStartingPoint+1,end()) == false) lResponse = false;
00478   }
00479   return lResponse;
00480 }
```

3.3.3.6 bool ExactRaytracer::traceBackward (ExactRaytracer::iterator *inActualEntry*, ExactRaytracer::iterator *inStopEntry*)

Do backward QU trace from actual entry to ending entry, using next entry values.

The actual QU entry ray values must be correctly set before the call.

Returns:

True if there was no total internal reflexion, false if there was.

Parameters:

inActualEntry y-nu entry from which to do the trace.

inStopEntry y-nu entry to stop the trace

```
00518 {
00519 #ifndef NDEBUG
00520   // Misc tests and assertions
00521   if(inActualEntry == end())
00522     throw LOSR_RuntimeErrorM("Cannot trace the ending iterator of a raytracer!");
00523   if(inActualEntry == end()-1)
00524     throw LOSR_RuntimeErrorM("Cannot backtrace the previous-to-ending iterator of a raytr
00525   if((inActualEntry == begin()) && (inActualEntry != inStopEntry))
00526     throw LOSR_RuntimeErrorM("The stop entry iterator is not reacheable!");
00527 #endif  // NDEBUG
00528
00529   // Get an iterator to the next entry
```

```
00530   iterator lNextEntry = inActualEntry+1;
00531
00532   // Apply the QU equations
00533   inActualEntry->mSinUp  = lNextEntry->mSinU;
00534   inActualEntry->mCosUp  = lNextEntry->mCosU;
00535   inActualEntry->mQp     = lNextEntry->mQ - ( inActualEntry->mT * inActualEntry->mSinUp );
00536   inActualEntry->mSinIp  = ( inActualEntry->mQp * inActualEntry->mC ) + inActualEntry->mSinUp;
00537   inActualEntry->mCosIp  = sqrt( 1.0 - (inActualEntry->mSinIp * inActualEntry->mSinIp) );
00538   // inActualEntry->mCosIp   = cos( asin(inActualEntry->mSinIp) );
00539   inActualEntry->mSinI_U = ( inActualEntry->mSinIp * inActualEntry->mCosUp ) -
00540     ( inActualEntry->mCosIp * inActualEntry->mSinUp );
00541   inActualEntry->mCosI_U = ( inActualEntry->mCosIp * inActualEntry->mCosUp ) +
00542     ( inActualEntry->mSinUp * inActualEntry->mSinIp );
00543
00544   if(inActualEntry == begin()) {
00545     inActualEntry->mSinI = inActualEntry->mSinIp;
00546     inActualEntry->mCosI = inActualEntry->mCosIp;
00547     inActualEntry->mSinU = inActualEntry->mSinUp;
00548     inActualEntry->mCosU = inActualEntry->mCosUp;
00549     inActualEntry->mQ    = inActualEntry->mQp;
00550   } else {
00551     inActualEntry->mSinI = inActualEntry->mSinIp * inActualEntry->mN / (inActualEntry-1)->mN;
00552     inActualEntry->mCosI = sqrt( 1.0 - (inActualEntry->mSinI * inActualEntry->mSinI) );
00553     // inActualEntry->mCosI   = cos( asin(inActualEntry->mSinI) );
00554     inActualEntry->mSinU = ( inActualEntry->mCosI_U * inActualEntry->mSinI ) -
00555       ( inActualEntry->mSinI_U * inActualEntry->mCosI );
00556     inActualEntry->mCosU = ( inActualEntry->mCosI_U * inActualEntry->mCosI ) +
00557       ( inActualEntry->mSinI_U * inActualEntry->mSinI );
00558     inActualEntry->mQ    = inActualEntry->mQp * ( inActualEntry->mCosU + inActualEntry->mCosI ) /
00559       ( inActualEntry->mCosUp + inActualEntry->mCosIp );
00560   }
00561
00562   //inActualEntry->mY = inActualEntry->mQ * (1.0 + inActualEntry->mCosI_U) /
00563   //  (inActualEntry->mCosU + inActualEntry->mCosI);
00564   inActualEntry->mY = (inActualEntry->mQ - (inActualEntry->mT * inActualEntry->mSinU) ) /
00565     (inActualEntry->mCosU);
00566   inActualEntry->mU = atan2(inActualEntry->mSinU,inActualEntry->mCosU);
00567
00568   // Recursive call to do the y-nu trace
00569   bool lResponse = true;
00570   if(inActualEntry != inStopEntry) lResponse = traceBackward(inActualEntry-1,inStopEntry);
00571   if((std::fabs(inActualEntry->mSinI) > 1.0) || (std::fabs(inActualEntry->mSinU) > 1.0) ||
00572      (std::fabs(inActualEntry->mCosI) > 1.0) || (std::fabs(inActualEntry->mCosU) > 1.0) ||
00573      (std::fabs(inActualEntry->mSinIp) > 1.0) || (std::fabs(inActualEntry->mSinUp) > 1.0) ||
00574      (std::fabs(inActualEntry->mCosIp) > 1.0) || (std::fabs(inActualEntry->mCosUp) > 1.0)) {
00575     lResponse = false;
00576   }
00577   return lResponse;
00578 }
```

3.3.3.7 bool ExactRaytracer::traceForward (ExactRaytracer::iterator *inActualEntry*, ExactRaytracer::iterator *inStopEntry*)

Do forward QU trace from actual entry to ending entry, using previous entry values.

The actual QU entry ray values must be correctly set before the call.

Returns:
True if there was no total internal reflexion, false if there was.

Parameters:
inActualEntry QU entry from which to do the trace.

inStopEntry y-nu entry to stop the trace.

```
00591 {
00592 #ifndef NDEBUG
00593   // Misc tests and assertions
00594   if(inActualEntry == end())
00595     throw LOSR_RuntimeErrorM("Cannot trace the ending iterator of a raytracer!");
00596   if(inActualEntry == begin())
00597     throw LOSR_RuntimeErrorM("Cannot forward trace the begining iterator of a raytracer!
00598   if((inActualEntry == end()-1) && (inActualEntry+1 != inStopEntry))
00599     throw LOSR_RuntimeErrorM("The stop entry iterator is not reacheable!");
00600 #endif // NDEBUG
00601
00602   // Get an iterator to the previous entry
00603   iterator lPreviousEntry = inActualEntry-1;
00604
00605   // Apply the QU equations
00606   inActualEntry->mQ    = lPreviousEntry->mQp + ( lPreviousEntry->mT * lPreviousEntry->mS
00607   inActualEntry->mSinU = lPreviousEntry->mSinUp;
00608   inActualEntry->mCosU = lPreviousEntry->mCosUp;
00609
00610   inActualEntry->mSinI   = ( inActualEntry->mQ * inActualEntry->mC ) + inActualEntry->mS
00611   inActualEntry->mCosI   = sqrt( 1.0 - (inActualEntry->mSinI * inActualEntry->mSinI) );
00612   // inActualEntry->mCosI   = cos( asin(inActualEntry->mSinI) );
00613   inActualEntry->mSinI_U = ( inActualEntry->mSinI * inActualEntry->mCosU ) -
00614     ( inActualEntry->mCosI * inActualEntry->mSinU );
00615   inActualEntry->mCosI_U = ( inActualEntry->mCosI * inActualEntry->mCosU ) +
00616     ( inActualEntry->mSinU * inActualEntry->mSinI );
00617   inActualEntry->mSinIp = inActualEntry->mSinI * lPreviousEntry->mN / inActualEntry->mN;
00618   inActualEntry->mCosIp = sqrt( 1.0 - (inActualEntry->mSinIp * inActualEntry->mSinIp) );
00619   // inActualEntry->mCosIp = cos( asin(inActualEntry->mSinIp) );
00620   inActualEntry->mSinUp = ( inActualEntry->mCosI_U * inActualEntry->mSinIp ) -
00621     ( inActualEntry->mSinI_U * inActualEntry->mCosIp );
00622   inActualEntry->mCosUp = ( inActualEntry->mCosI_U * inActualEntry->mCosIp ) +
00623     ( inActualEntry->mSinI_U * inActualEntry->mSinIp );
00624   inActualEntry->mQp    = inActualEntry->mQ *
00625     ( inActualEntry->mCosUp + inActualEntry->mCosIp ) /
00626     ( inActualEntry->mCosU + inActualEntry->mCosI );
00627
00628   // inActualEntry->mY = inActualEntry->mQ * (1.0 + inActualEntry->mCosI_U) /
```

```
00629   //   (inActualEntry->mCosU + inActualEntry->mCosI);
00630   inActualEntry->mY = (inActualEntry->mQ - (inActualEntry->mT * inActualEntry->mSinU) ) /
00631     (inActualEntry->mCosU);
00632   inActualEntry->mU = atan2(inActualEntry->mSinU,inActualEntry->mCosU);
00633
00634   // Recursive call to do the y-nu trace
00635   bool lResponse = true;
00636   if(inActualEntry+1 != inStopEntry) lResponse = traceForward(inActualEntry+1,inStopEntry);
00637   if((std::fabs(inActualEntry->mSinI) > 1.0) || (std::fabs(inActualEntry->mSinU) > 1.0) ||
00638      (std::fabs(inActualEntry->mCosI) > 1.0) || (std::fabs(inActualEntry->mCosU) > 1.0) ||
00639      (std::fabs(inActualEntry->mSinIp) > 1.0) || (std::fabs(inActualEntry->mSinUp) > 1.0) ||
00640      (std::fabs(inActualEntry->mCosIp) > 1.0) || (std::fabs(inActualEntry->mCosUp) > 1.0)) {
00641     lResponse = false;
00642   }
00643   return lResponse;
00644 }
```

3.3.3.8 void ExactRaytracer::write (std::ostream & *ioOs* = std::cout) const

Write a QU trace into a C++ output stream.

Parameters:

ioOs Output stream to write the trace into.

```
00653 {
00654   ioOs << '(' << size() << ')' << std::endl;
00655   for(unsigned int i=0; i<size(); i++) {
00656     ioOs << (*this)[i];
00657   }
00658 }
```

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

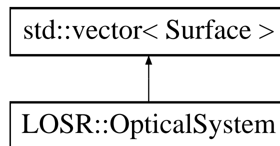
- ExactRaytracer.hpp
- ExactRaytracer.cpp

3.4 LOSR::OpticalSystem Class Reference

LOSR::OpticalSystem is a on-axis lenses system made of optical materials.

```
#include <OpticalSystem.hpp>
```

Inheritance diagram for LOSR::OpticalSystem::



Public Methods

- OpticalSystem (unsigned int inSize=0)

  Construct an optical system of the size given.

- ~OpticalSystem ()

  Destructor (do nothing).

- void read (std::istream &ioIs=std::cin)

  Read a LOSR::OpticalSystem from a C++ input stream.

- void write (std::ostream &ioOs=std::cout) const

  Write a LOSR::OpticalSystem into a C++ output stream.

- bool validate ()

  Validate and correct the apertures of the optical system.

3.4.1 Detailed Description

LOSR::OpticalSystem is a on-axis lenses system made of optical materials.

LOSR::OpticalSystem is a on-axis lenses system made of optical materials. It is a std::vector of LOSR::Surface, that is a sequence of surface between different optical materials.

Author:

Christian Gagné and Julie Beaulieu

Version:
0.2

Date:
8/28/2001

3.4.2 Constructor & Destructor Documentation

3.4.2.1 OpticalSystem::OpticalSystem (unsigned int *inSize* = 0)

Construct an optical system of the size given.

Parameters:
inSize Size of the optical system.

```
00041   :
00042   std::vector<Surface>(inSize)
00043 { }
```

3.4.3 Member Function Documentation

3.4.3.1 void OpticalSystem::read (std::istream & *ioIs* = std::cin)

Read a LOSR::OpticalSystem from a C++ input stream.

Parameters:
ioIs Input stream to extract the LOSR::OpticalSystem from.

```
00052 {
00053   char c1 ='\0', c2='\0';
00054   unsigned int sz = 0;
00055   char buf[2048];
00056
00057   // Extract and ignore the rest of lines starting with a #
00058   for(ioIs >> c1; c1=='#'; ioIs >> c1) ioIs.getline(buf,2048,'\n');
00059   if(!ioIs) throw LOSR_RuntimeErrorM("Premature end of stream!");
00060
00061   // Looking for the number of surface, between parenthesis
00062   ioIs >> sz >> c2;
00063   if((c1!='(') || (c2!=')')) throw LOSR_RuntimeErrorM("Bad file format!");
00064   resize(sz);
00065
00066   // Read each surfaces from the stream
00067   for(unsigned int i=0; i<sz; i++) {
00068     if(!ioIs) throw LOSR_RuntimeErrorM("Premature end of stream!");
00069     (*this)[i].read(ioIs);
00070   }
```

```
00071
00072 }
```

3.4.3.2 bool OpticalSystem::validate ()

Validate and correct the apertures of the optical system.

The system is validate by calculating whether the curvature and the distance between of two adjacent surfaces is physically possible with the given apertures. If the system is not physically possible, the apertures are reduced to the greatest value valid for the actual setup.

Returns:

True if the optical system is valid, false if it needed to be corrected.

```
00098 {
00099   const double lcMinRadius = 0.0001;
00100   bool lSystemValid = true;
00101
00102   // For all surfaces of the optical system
00103   for(unsigned int i=0; i<(size()-1); i++) {
00104     // Determine the minimal aperture between the two surfaces
00105     double lMinAperture = LOSR_MinM( (*this)[i].getAperture(), (*this)[i+1].getAperture(
00106
00107     // Calculates values for the first surface
00108     double lAlpha1 = 0;
00109     double lR1 = (*this)[i].getCurvature();
00110     if(fabs(lR1) > lcMinRadius) {
00111       lR1 = 1.0 / lR1;
00112       lAlpha1 = lR1 - sqrt((lR1*lR1) - (lMinAperture*lMinAperture));
00113     }
00114
00115     // Calculates values for the second surface
00116     double lAlpha2 = 0;
00117     double lR2 = (*this)[i+1].getCurvature();
00118     if(fabs(lR2) > lcMinRadius) {
00119       lR2 = 1.0 / lR2;
00120       lAlpha2 = lR2 - sqrt((lR2*lR2) - (lMinAperture*lMinAperture));
00121     }
00122
00123     // Test whether the apertures are valid. If not, correct it
00124     // See maple spreadsheet file opsys_validate.pdf for calculation details.
00125     if((*this)[i].getThickness() < (lAlpha1+lAlpha2)) {
00126       lSystemValid = false;
00127       double lY = 0.0;
00128       double lT = (*this)[i].getThickness();
00129       if(fabs(lR1) <= lcMinRadius) {
00130         // R1 = infinity, x=0, using formula for y2b (see file opsys_validate.pdf)
00131         lY = sqrt( LOSR_Pow2M(lR2) - LOSR_Pow2M(lR2 - lT) );
00132       } else if(fabs(lR2) <= lcMinRadius) {
00133         // R2 = infinity, x=T, using formula for y2a (see file opsys_validate.pdf)
```

```
00134         lY = sqrt( LOSR_Pow2M(lR1) - LOSR_Pow2M(lT - lR1) );
00135       } else {
00136         // Using formula for y (see file opsys_validate.pdf)
00137         lY = sqrt(
00138           LOSR_Pow2M(lR1) -
00139           LOSR_Pow2M( ( - (lT * (lT - 2.0 * lR2)) / (2.0 * (lR1 + lR2 - lT)) ) - lR1 )
00140         );
00141       }
00142       (*this)[i].setAperture( LOSR_MinM( (*this)[i].getAperture(), lY ) );
00143       (*this)[i+1].setAperture( LOSR_MinM( (*this)[i+1].getAperture(), lY ) );
00144     }
00145
00146   }
00147   return lSystemValid;
00148 }
```

3.4.3.3 void OpticalSystem::write (std::ostream & *ioOs* = std::cout) const

Write a LOSR::OpticalSystem into a C++ output stream.

Parameters:

ioOs Output stream to write the LOSR::OpticalSystem into.

```
00081 {
00082   ioOs << '(' << size() << ')' << std::endl;
00083   for(unsigned int i=0; i<size(); i++) {
00084     (*this)[i].write(ioOs);
00085   }
00086 }
```

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

- OpticalSystem.hpp
- OpticalSystem.cpp

3.5 LOSR::Pupil Struct Reference

LOSR::Pupil describe the value of an entrance or exit pupil.

```
#include <Analyzer.hpp>
```

Public Methods

- Pupil (long double inT=0.0, long double inR=0.0, long double inM=0.0)
  Construct a LOSR::Pupil object with the value given.

Public Attributes

- long double mT
  Distance to the pupil from the actual point of view.
- long double mR
  Aperture of the pupil from the actual point of view.
- long double mM
  Magnification at the pupil.

3.5.1 Detailed Description

LOSR::Pupil describe the value of an entrance or exit pupil.

Author:
Christian Gagné and Julie Beaulieu

Version:
0.2

Date:
9/05/2001

3.5.2 Constructor & Destructor Documentation

3.5.2.1 Pupil::Pupil (long double *inT* = 0.0, long double *inR* = 0.0, long double *inM* = 0.0)

Construct a LOSR::Pupil object with the value given.

Parameters:

inT Value of the distance mT.

inR Value of the aperture mR.

inM Value of the magnification fact mM.

```
00041   :
00042   mT(inT),
00043   mR(inR),
00044   mM(inM)
00045 { }
```

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

- Analyzer.hpp
- Analyzer.cpp

3.6 LOSR::QU Struct Reference

LOSR::QU describe a surface entry into an exact raytracing table (QU trace).

`#include <ExactRaytracer.hpp>`

Public Methods

- QU ()
  Construct a QU trace entry.
- QU (const LOSR::Surface &inSurface)
  Construct a QU trace entry using a surface.
- QU& operator= (const LOSR::Surface &inSurface)
  Initialize a QU trace entry using a surface.

Public Attributes

- long double mT
  Distance to the next surface.
- long double mN
  Refraction index of the material at the right.
- long double mC
  Curvature of the surface.
- long double mR
  Aperture radius of the surface.
- long double mY
  Height of the ray at the surface.
- long double mU
  Angle of the ray at the surface.
- long double mQ
  Initial ray-vertex perpenticular.
- long double mSinU

Sine of the initial slope angle.

- long double mCosU

  Cosine of the initial slope angle.

- long double mSinI

  Sine of the initial incidence angle.

- long double mCosI

  Cosine of the initial incidence angle.

- long double mSinI_U

  Sine of (initial incidence angle minus initial slope angle).

- long double mCosI_U

  Cosine of (initial incidence angle minus initial slope angle).

- long double mSinIp

  Sine of the refracted incidence angle.

- long double mCosIp

  Cosine of the refracted incidence angle.

- long double mSinUp

  Sine of the refracted slope angle.

- long double mCosUp

  Cosine of the refracted slope angle.

- long double mQp

  Refracted ray-vertex perpenticular.

3.6.1 Detailed Description

LOSR::QU describe a surface entry into an exact raytracing table (QU trace).

Author:
Christian Gagné and Julie Beaulieu

Version:
0.2

Date:
9/03/2001

3.6.2 Constructor & Destructor Documentation

3.6.2.1 QU::QU (const LOSR::Surface & *inSurface*)

Construct a QU trace entry using a surface.

Parameters:

inSurface Surface from which the QU entry is built

```
00064   :
00065   mT(inSurface.getThickness()),
00066   mN(inSurface.getIndex()),
00067   mC(inSurface.getCurvature()),
00068   mR(inSurface.getAperture()),
00069   mY(0),
00070   mU(0),
00071   mQ(0),
00072   mSinU(0),
00073   mCosU(1),
00074   mSinI(0),
00075   mCosI(1),
00076   mSinI_U(0),
00077   mCosI_U(1),
00078   mSinIp(0),
00079   mCosIp(1),
00080   mSinUp(0),
00081   mCosUp(1),
00082   mQp(0)
00083 { }
```

3.6.3 Member Function Documentation

3.6.3.1 QU & QU::operator= (const LOSR::Surface & *inSurface*)

Initialize a QU trace entry using a surface.

Returns:

The actual QU trace entry.

Parameters:

inSurface Surface from which the QU entry is built

```
00093 {
00094   mT = inSurface.getThickness();
00095   mN = inSurface.getIndex();
00096   mC = inSurface.getCurvature();
00097   mR = inSurface.getAperture();
00098   mY = 0;
```

```
00099   mU = 0;
00100   mQ = 0;
00101   mSinU = 0;
00102   mCosU = 1;
00103   mSinI = 0;
00104   mCosI = 1;
00105   mSinI_U = 0;
00106   mCosI_U = 1;
00107   mSinIp = 0;
00108   mCosIp = 1;
00109   mSinUp = 0;
00110   mCosUp = 1;
00111   mQp = 0;
00112   return *this;
00113 }
```

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

- ExactRaytracer.hpp
- ExactRaytracer.cpp

3.7 LOSR::Surface Class Reference

LOSR::Surface describe a sperical contact between two optical materials.

```
#include <Surface.hpp>
```

Public Methods

- Surface (double inCurvature=0, double inIndex=0, double inAperture=0, double inThickness=0)

  Construct a Surface object with the values given. By default, values are set to 0.

- ~Surface ()

  Destructor (do nothing).

- double getAperture () const

  Return aperture radius.

- double getCurvature () const

  Return curvature.

- double getIndex () const

  Return glass refraction index to the right.

- double getThickness () const

  Return distance to the next surface.

- void read (std::istream &ioIs=std::cin)

  Read a LOSR::Surface from a C++ input stream.

- void setAperture (double inAperture)

  Set aperture radius.

- void setCurvature (double inCurvature)

  Set curvature.

- void setIndex (double inIndex)

  Set glass refraction index.

- void setThickness (double inThickness)

  Set distance to next surface.

- void write (std::ostream &ioOs=std::cout) const

Write a LOSR::Surface into a C++ output stream.

Protected Attributes

- double mCurvature

  Curvature of the spherical contact.

- double mIndex

  Glass refraction index of the material at the right of the surface.

- double mAperture

  Radius of the aperture of the surface.

- double mThickness

  Distance to the next surface at the right, on the optical axis.

3.7.1 Detailed Description

LOSR::Surface describe a sperical contact between two optical materials.

LOSR::Surface describe a sperical contact between two optical materials. It haves as attributes, the aperture radius of the surface, the glass refraction index of the material at the right of the surface, the curvature (1/radius) of the surface and the distance to the next surface on the axis.

Author:
Christian Gagné and Julie Beaulieu

Version:
0.2

Date:
8/23/2001

3.7.2 Constructor & Destructor Documentation

3.7.2.1 Surface::Surface (double *inCurvature* = 0, double *inIndex* = 0, double *inAperture* = 0, double *inThickness* = 0)

Construct a Surface object with the values given. By default, values are set to 0.

Parameters:

inCurvature Curvature value of the sperical surface.

inIndex Glass refraction index value at the right of the surface.

inAperture Aperture radius value.

inThickness Distance value to the next right surface.

```
00039   :
00040   mCurvature(inCurvature),
00041   mIndex(inIndex),
00042   mAperture(inAperture),
00043   mThickness(inThickness)
00044 { }
```

3.7.3 Member Function Documentation

3.7.3.1 void Surface::read (std::istream & *ioIs* = std::cin)

Read a LOSR::Surface from a C++ input stream.

Parameters:

ioIs Input stream to extract the LOSR::Surface from.

```
00053 {
00054   char c1 ='\0', c2='\0';
00055   char buf[2048];
00056 
00057   // Extract and ignore the rest of lines starting with a #
00058   for(ioIs >> c1; c1=='#'; ioIs >> c1) ioIs.getline(buf,2048,'\n');
00059   if(!ioIs) throw LOSR_RuntimeErrorM("Premature end of stream!");
00060   // Looking for chars 'c:'
00061   ioIs >> c2;
00062   if((c1!='c') || (c2!=':')) throw LOSR_RuntimeErrorM("Invalide format!");
00063   if(!ioIs) throw LOSR_RuntimeErrorM("Premature end of stream!");
00064   // Found what was needed, read curvature
00065   ioIs >> mCurvature;
00066 
00067   // Extract and ignore the rest of lines starting with a #
00068   for(ioIs >> c1; c1=='#'; ioIs >> c1) ioIs.getline(buf,2048,'\n');
00069   if(!ioIs) throw LOSR_RuntimeErrorM("Premature end of stream!");
00070   // Looking for chars 'i:'
00071   ioIs >> c2;
00072   if((c1!='i') || (c2!=':')) throw LOSR_RuntimeErrorM("Invalide format!");
00073   if(!ioIs) throw LOSR_RuntimeErrorM("Premature end of stream!");
00074   // Found what was needed, read glass refraction index
00075   ioIs >> mIndex;
00076 
00077   // Extract and ignore the rest of lines starting with a #
00078   for(ioIs >> c1; c1=='#'; ioIs >> c1) ioIs.getline(buf,2048,'\n');
```

```
00079   if(!ioIs) throw LOSR_RuntimeErrorM("Premature end of stream!");
00080   // Looking for chars 'a:'
00081   ioIs >> c2;
00082   if((c1!='a') || (c2!=':')) throw LOSR_RuntimeErrorM("Invalide format!");
00083   if(!ioIs) throw LOSR_RuntimeErrorM("Premature end of stream!");
00084   // Found what was needed, read aperture of the surface
00085   ioIs >> mAperture;
00086
00087   // Extract and ignore the rest of lines starting with a #
00088   for(ioIs >> c1; c1=='#'; ioIs >> c1) ioIs.getline(buf,2048,'\n');
00089   if(!ioIs) throw LOSR_RuntimeErrorM("Premature end of stream!");
00090   // Looking for chars 't:'
00091   ioIs >> c2;
00092   if((c1!='t') || (c2!=':')) throw LOSR_RuntimeErrorM("Invalide format!");
00093   if(!ioIs) throw LOSR_RuntimeErrorM("Premature end of stream!");
00094   // Found what was needed, read distance to next surface
00095   ioIs >> mThickness;
00096 }
```

3.7.3.2 void Surface::write (std::ostream & *ioOs* = std::cout) const

Write a LOSR::Surface into a C++ output stream.

Parameters:

ioOs Output stream to write the LOSR::Surface into.

```
00105 {
00106   ioOs << "c:"    << mCurvature
00107        << " \ti:" << mIndex
00108        << " \ta:" << mAperture
00109        << " \tt:" << mThickness
00110        << std::endl;
00111 }
```

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

- Surface.hpp
- Surface.cpp

3.8 LOSR::Y_NU Struct Reference

LOSR::Y_NU describe a surface entry into an approximative raytracing table (y-nu trace).

`#include <ApproxRaytracer.hpp>`

Public Methods

- Y_NU ()

  Construct a Y_NU trace entry.

- Y_NU (const LOSR::Surface &inSurface)

  Construct a Y_NU trace entry using the surface to initialize the y-nu entry.

- Y_NU& operator= (const LOSR::Surface &inSurface)

  Initialize a y-nu trace entry with a surface.

Public Attributes

- long double mT

  Distance to the next surface.

- long double mN

  Refraction index of the material at the right.

- long double mC

  Curvature of the surface.

- long double mR

  Aperture radius of the surface.

- long double mY

  Height of the ray on the surface.

- long double mU

  Angle (in radian) of the ray.

- long double mNU

  Angle times de refraction index.

3.8.1 Detailed Description

LOSR::Y_NU describe a surface entry into an approximative raytracing table (y-nu trace).

Author:
Christian Gagné and Julie Beaulieu

Version:
0.2

Date:
8/31/2001

3.8.2 Constructor & Destructor Documentation

3.8.2.1 Y_NU::Y_NU (const LOSR::Surface & *inSurface*)

Construct a Y_NU trace entry using the surface to initialize the y-nu entry.

Parameters:
inSurface Surface to made to initialize the y-nu trace entry.

```
00055   :
00056   mT(inSurface.getThickness()),
00057   mN(inSurface.getIndex()),
00058   mC(inSurface.getCurvature()),
00059   mR(inSurface.getAperture()),
00060   mY(0),
00061   mU(0),
00062   mNU(0)
00063 { }
```

3.8.3 Member Function Documentation

3.8.3.1 Y_NU & Y_NU::operator= (const LOSR::Surface & *inSurface*)

Initialize a y-nu trace entry with a surface.

Parameters:
inSurface Surface to made to initialize the y-nu trace entry.

```
00072 {
00073   mT = inSurface.getThickness();
```

```
00074   mN = inSurface.getIndex();
00075   mC = inSurface.getCurvature();
00076   mR = inSurface.getAperture();
00077   mY = 0;
00078   mU = 0;
00079   mNU = 0;
00080   return *this;
00081 }
```

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

- ApproxRaytracer.hpp
- ApproxRaytracer.cpp