My Project

Generated by Doxygen 1.8.6

Fri Jan 31 2014 17:05:11

Contents

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Module Documentation

5.1 Auxiliary classes

Classes

- class Arr2D

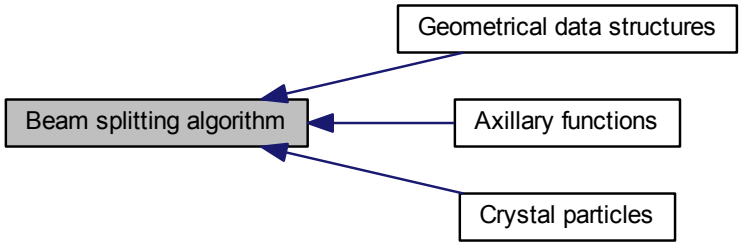
  *The array with (N-rows x M-columns) dimensions of small **real-value** matrixes with (n x m) dimensions.*
- class Arr2DC

  *The array with (N-rows x M-columns) dimensions of small **complex-value** matrixes with (n x m) dimensions.*

5.1.1 Detailed Description

5.2 Beam splitting algorithm

Collaboration diagram for Beam splitting algorithm:



Modules

- Axillary functions
- Crystal particles
- Geometrical data structures

Classes

- class Beam

  The Beam class consist of 3D polygon, propagation direction and it's history, Jones matrix and so on.
- class matrix

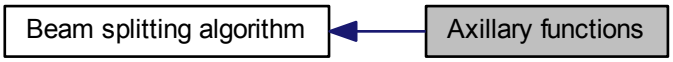
  *The array with (n-rows x m-columns) dimensions of **real** values. Size of the array can't be changed.*
- class matrixC

  *The array with (n-rows x m-columns) dimensions of **complex** values. Size of the array can't be changed.*

5.2.1 Detailed Description

5.3 Axillary functions

Collaboration diagram for Axillary functions:



Classes

- class complex

  This class provides a complex numbers and operation with them.
- class Chain

  The class represents chain of number of facets.

Functions

- complex exp (const complex &z)
- complex exp_im (double x)
- void SinCos (long double, long double &, long double &)
- matrix Mueller (const matrixC &in)

  The function returns Mueller matrix calculated from Jones matrix.
- void RightRotateMueller (matrix &, double, double)

  *Right multiplication of matrix **m** by rotation matrix with cos(f)=cs, sin(f)=sn.*
- void LeftRotateMueller (matrix &, double, double)

  *Left multiplication of matrix **m** by rotation matrix with cos(f)=cs, sin(f)=sn.*
- void RotateMueller (matrix &m, double _cs, double _sn, double cs, double sn)

  *Multiplication of matrix **m** by left and right rotation matrixes*
- void ForwardScattering (matrix &)

  *Calculate Mueller matrix in forward direction for randomly oriented particle. **See equation below.***
- void BackwardScattering (matrix &)

  *Calculate Mueller matrix in backward direction for randomly oriented particle. **See equation below.***

5.3.1 Detailed Description

5.3.2 Function Documentation

5.3.2.1 void BackwardScattering (matrix &)

Calculate Mueller matrix in backward direction for randomly oriented particle. **See equation below.**

Mueller matrix in backward direction for randomly oriented particle

```
| M11    0           0           0    |
| 0      (M22-M33)/2 0           0    |
| 0      0           (M33-M22)/2 0    |
| 0      0           0           M44  |
```

Definition at line 109 of file Mueller.cpp.

5.3.2.2 complex exp (const complex & *z*)

Definition at line 112 of file compl.hpp.

5.3.2.3 complex exp_im (double *x*)

Definition at line 30 of file compl.cpp.

5.3.2.4 void ForwardScattering (matrix &)

Calculate Mueller matrix in forward direction for randomly oriented particle. **See equation below.**

Mueller matrix in forward direction for randomly oriented particle

```
| M11    0           0           0   |
| 0      (M22+M33)/2 0           0   |
| 0      0           (M22+M33)/2 0   |
| 0      0           0           M44 |
```

Definition at line 98 of file Mueller.cpp.

5.3.2.5 void LeftRotateMueller (matrix & , double , double)

Left multiplication of matrix **m** by *rotation matrix* with cos(f)=cs, sin(f)=sn.

Rotation matrix

```
| 1  0   0   0 |
| 0  cs  sn  0 |
| 0  -sn cs  0 |
| 0  0   0   1 |
```

Definition at line 51 of file Mueller.cpp.

5.3.2.6 matrix Mueller (const matrixC & *in*)

The function returns Mueller matrix calculated from Jones matrix.

The dimension of Mueller matrix is 4x4. The dimension of Jones matrix must be 2x2.

Parameters

in	complex-value matrix (matrixC)

Returns

real-value matrix

Definition at line 6 of file Mueller.cpp.

5.3.2.7 void RightRotateMueller (matrix & , double , double)

Right multiplication of matrix **m** by *rotation matrix* with cos(f)=cs, sin(f)=sn.

Rotation matrix

```
| 1  0   0   0 |
| 0  cs  sn  0 |
| 0  -sn cs  0 |
| 0  0   0   1 |
```

Definition at line 33 of file Mueller.cpp.

5.3.2.8 void RotateMueller (matrix & *m*, double *_cs*, double *_sn*, double *cs*, double *sn*)

Multiplication of matrix **m** by left and right *rotation matrixes*

Right rotation matrix

```
| 1  0   0   0 |
| 0  cs  sn  0 |
| 0  -sn cs  0 |
| 0  0   0   1 |
```

Left rotation matrix

```
| 1   0      0   0 |
| 0   _cs    _sn 0 |
| 0  -(_sn)  _cs 0 |
| 0   0      0   1 |
```

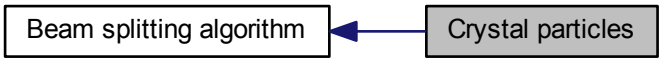
Definition at line 69 of file Mueller.cpp.

5.3.2.9 void SinCos (long *double*, long double & , long double &)

Definition at line 8 of file compl.cpp.

5.4 Crystal particles

Collaboration diagram for Crystal particles:



Classes

- class BeamSplitting

  This class is a prototype for Crystal class. It contains parameters for BeamSplitting algorithm.
- class Frame

  This class is a prototype for Crystal class.
- class Crystal

  This class contains all information about particle's geometry and the parameters of tracing.
- class Prism

  This class defines hexagonal prism.
- class Pyramid

  This class defines the pyramid.
- class Bullet

  This class defines the hexagonal bullet.
- class TaperedPrism

  This class defines the hexagonal tapered prism.
- class CavityPrism

  This class defines the hexagonal prism with cavity on bottom.
- class Cup

  This class defines the cupped prism.
- class TrianglePrism

  This class defines the triangle prism.
- class Parallelepiped

  This class defines the parallelepiped.
- class Tetrahedron

  This class defines the tetrahedron.
- class Dodecahedron

  This class defines the dodecahedron.

Typedefs

- typedef void(∗ Hand)(Beam &)

  Function prototype witch should be used for handling the outgoing beam in main program.

5.4.1 Detailed Description

5.4.2 Typedef Documentation

5.4.2.1 typedef void(∗ Hand)(Beam &)

Function prototype witch should be used for handling the outgoing beam in main program.

Returns

none

Definition at line 29 of file Crystal.hpp.

5.5 Geometrical data structures

Collaboration diagram for Geometrical data structures:

Classes

- class Point2D

  The class Point2D contains 2D point in Cartesian coordinates.
- class Edge

  The Edge class represents all forms of lines in 2D.
- class Point3D

  The class Point3D contains 3D point in Cartesian coordinates.
- class SphCrd

  Direction in 3D space (spherical coordinates on unit sphere).
- class Node

  A pointer-based implementation of a linked list.
- class Vertex

  The class contains vertexes of 2D polygon.
- class Polyg

  The Polygon class represents 2D polygon.

Macros

- #define m_2pi 6.28318530717958647692528676655900

Enumerations

- enum direction { NON_PARALLEL, OPPOSITE, ACCORDANT, PERPENDICULAR }

  relative position of two edges
- enum rotation { CLOCKWISE, COUNTER_CLOCKWISE }

  direction of rotation (p.84)
- enum state {
  COLLINEAR, PARALLEL, SKEW, SKEW_CROSS,
  SKEW_NO_CROSS }

  relative state of two edges (p.93)
- enum position {
  LEFT, RIGHT, BEYOND, BEHIND,
  BETWEEN, ORIGIN, DESTINATION }

  position of point regarding to vector (p.76)
- enum crossing { UNKNOWN, P_IS_INSIDE, Q_IS_INSIDE }

  mutual position of two 2D polygons (p.158)

Variables

- const double E_DBL = 1e7∗DBL_EPSILON

5.5.1 Detailed Description

5.5.2 Macro Definition Documentation

5.5.2.1 #define m_2pi 6.28318530717958647692528676655900

Definition at line 33 of file Geometry.hpp.

5.5.3 Enumeration Type Documentation

5.5.3.1 enum crossing

mutual position of two 2D polygons (p.158)

Enumerator

UNKNOWN
P_IS_INSIDE
Q_IS_INSIDE

Definition at line 45 of file Geometry.hpp.

5.5.3.2 enum direction

relative position of two edges

Enumerator

NON_PARALLEL
OPPOSITE
ACCORDANT
PERPENDICULAR

Definition at line 37 of file Geometry.hpp.

5.5.3.3 enum position

position of point regarding to vector (p.76)

Enumerator

LEFT
RIGHT
BEYOND
BEHIND
BETWEEN
ORIGIN
DESTINATION

Definition at line 43 of file Geometry.hpp.

5.5.3.4 enum rotation

direction of rotation (p.84)

Enumerator

CLOCKWISE

COUNTER_CLOCKWISE

Definition at line 39 of file Geometry.hpp.

5.5.3.5 enum state

relative state of two edges (p.93)

Enumerator

COLLINEAR

PARALLEL

SKEW

SKEW_CROSS

SKEW_NO_CROSS

Definition at line 41 of file Geometry.hpp.

5.5.4 Variable Documentation

5.5.4.1 const double E_DBL = 1e7∗DBL_EPSILON

Definition at line 34 of file Geometry.hpp.

Chapter 6

Class Documentation

6.1 Arr2D Class Reference

The array with (N-rows x M-columns) dimensions of small **real-value** matrixes with (n x m) dimensions.

`#include <PhysMtr.hpp>`

Public Member Functions

- Arr2D (unsigned int _N, unsigned int _M, unsigned int _n, unsigned int _m)

  Creates the array with (_N-rows x _M-columns) dimensions of small real-value matrixes with (_n x _m) dimensions.
- Arr2D (void)
- Arr2D (const Arr2D &)
- ~Arr2D (void)
- matrix operator() (unsigned int, unsigned int) const

  Returns the matrix stored in the array by address (_N,_M)
- double & operator() (unsigned int _N, unsigned int _M, unsigned int _n, unsigned int _m)

  Returns a reference to the matrix element (_n,_m) stored in the array by address (_N,_M)
- double operator() (unsigned int _N, unsigned int _M, unsigned int _n, unsigned int _m) const

  Returns the matrix element (_n,_m) stored in the array by address (_N,_M)
- Arr2D operator= (const Arr2D &)
- Arr2D operator+= (const Arr2D &)
- Arr2D operator* (double)
- void ClearArr (void) const

  The function clears all elements of all matrixes in the array.
- void insert (unsigned int _N, unsigned int _M, const matrix &mt)

  The function adds the matrix mt to existing matrix, located in the array by address (_N,_M)
- void replace (unsigned int _N, unsigned int _M, const matrix &mt)

  The function replaces existing matrix, located in the array by address (_N,_M), by the matrix mt.

Friends

- double Max (const Arr2D &Arr)

  The function returns the maximal value of all elements contained in the array.
- matrix SumArr (const Arr2D &)

  The function sums up all matrixes contained in the array.
- unsigned int StrArr (const Arr2D &Arr)

  The function returns row counts.
- unsigned int ColArr (const Arr2D &Arr)

  The function returns column counts.

6.1.1 Detailed Description

The array with (N-rows x M-columns) dimensions of small **real-value** matrixes with (n x m) dimensions.

Example of using:

```
...
Arr2D Arr(0,0,0,0); //creating an array
main()
{
  ...
  Arr = Arr2D(1, 1, 2, 2); //changing the size
  Arr.ClearArr();   //clearing the array
  matrix mt(2,2);
  Arr.insert(0,0, mt); //adding matrix mt to the array
  matrix M=Arr(0,0); //taking matrix M from the array
  ...
}
```

Definition at line 34 of file PhysMtr.hpp.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 Arr2D::Arr2D (unsigned int *_N,* unsigned int *_M,* unsigned int *_n,* unsigned int *_m*) [inline]

Creates the array with (_N-rows x _M-columns) dimensions of small real-value matrixes with (_n x _m) dimensions.

Definition at line 45 of file PhysMtr.hpp.

6.1.2.2 Arr2D::Arr2D (void) [inline]

Definition at line 47 of file PhysMtr.hpp.

6.1.2.3 Arr2D::Arr2D (const Arr2D & *Arr*)

Definition at line 28 of file PhysMtr.cpp.

6.1.2.4 Arr2D::~Arr2D (void) [inline]

Definition at line 51 of file PhysMtr.hpp.

6.1.3 Member Function Documentation

6.1.3.1 void Arr2D::ClearArr (void) const

The function clears all elements of all matrixes in the array.

Definition at line 75 of file PhysMtr.cpp.

6.1.3.2 void Arr2D::insert (unsigned int *_N,* unsigned int *_M,* const matrix & *mt*)

The function adds the matrix mt to existing matrix, located in the array by address (_N,_M)

Parameters

_N,_M	number of a cell in the array
mt	adding matrix

Returns

none

Definition at line 82 of file PhysMtr.cpp.

6.1.3.3 matrix Arr2D::operator() (unsigned int *_N,* unsigned int *_M*) const

Returns the matrix stored in the array by address (_N,_M)

Definition at line 37 of file PhysMtr.cpp.

6.1.3.4 double& Arr2D::operator() (unsigned int *_N,* unsigned int *_M,* unsigned int *_n,* unsigned int *_m*) [inline]

Returns a reference to the matrix element (_n,_m) stored in the array by address (_N,_M)

Definition at line 56 of file PhysMtr.hpp.

6.1.3.5 double Arr2D::operator() (unsigned int *_N,* unsigned int *_M,* unsigned int *_n,* unsigned int *_m*) const [inline]

Returns the matrix element (_n,_m) stored in the array by address (_N,_M)

Definition at line 60 of file PhysMtr.hpp.

6.1.3.6 Arr2D Arr2D::operator∗ (double *x*)

Definition at line 132 of file PhysMtr.cpp.

6.1.3.7 Arr2D Arr2D::operator+= (const Arr2D & *arr*)

Definition at line 63 of file PhysMtr.cpp.

6.1.3.8 Arr2D Arr2D::operator= (const Arr2D & *Arr*)

Definition at line 50 of file PhysMtr.cpp.

6.1.3.9 void Arr2D::replace (unsigned int *_N,* unsigned int *_M,* const matrix & *mt*)

The function replaces existing matrix, located in the array by address (_N,_M), by the matrix mt.

Parameters

_N,_M	number of a cell in the array
mt	new matrix

Returns

none

Definition at line 93 of file PhysMtr.cpp.

6.1.4 Friends And Related Function Documentation

6.1.4.1 unsigned int ColArr (const Arr2D & *Arr*) [friend]

The function returns column counts.

Parameters

Arr	pointer to the array

Returns

column counts, (Arr.M)

Definition at line 114 of file PhysMtr.hpp.

6.1.4.2 double Max (const Arr2D & *Arr*) [friend]

The function returns the maximal value of all elements contained in the array.

Parameters

Arr	pointer to the array

Returns

maximal element of the array

Definition at line 106 of file PhysMtr.cpp.

6.1.4.3 unsigned int StrArr (const Arr2D & *Arr*) [friend]

The function returns row counts.

Parameters

Arr	pointer to the array

Returns

row counts, (Arr.N)

Definition at line 106 of file PhysMtr.hpp.

6.1.4.4 matrix SumArr (const Arr2D & *Arr*) [friend]

The function sums up all matrixes contained in the array.

Parameters

Arr	pointer to the array

Returns

summed matrix

Definition at line 119 of file PhysMtr.cpp.

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

- PhysMtr.hpp
- PhysMtr.cpp

6.2 Arr2DC Class Reference

The array with (N-rows x M-columns) dimensions of small **complex-value** matrixes with (n x m) dimensions.

`#include <PhysMtr.hpp>`

Public Member Functions

- Arr2DC (unsigned int _N, unsigned int _M, unsigned int _n, unsigned int _m)

  Creates the array with (_N-rows x _M-columns) dimensions of small real-value matrixes with (_n x _m) dimensions.
- Arr2DC (void)
- Arr2DC (const Arr2DC &)
- ~Arr2DC (void)
- matrixC operator() (unsigned int, unsigned int) const

  Returns the matrix stored in the array by address (_N,_M)
- complex & operator() (unsigned int _N, unsigned int _M, unsigned int _n, unsigned int _m)

  Returns a reference to the matrix element (_n,_m) stored in the array by address (_N,_M)
- complex operator() (unsigned int _N, unsigned int _M, unsigned int _n, unsigned int _m) const

  Returns the matrix element (_n,_m) stored in the array by address (_N,_M)
- Arr2DC operator= (const Arr2DC &)
- Arr2DC operator+= (const Arr2DC &)
- Arr2DC operator/= (double)
- void ClearArr (void) const

  The function clears all elements of all matrixes in the array.
- void insert (unsigned int, unsigned int, const matrixC &)

  The function adds the matrix mt to existing matrix, located in the array by address (_N,_M)
- void replace (unsigned int, unsigned int, const matrixC &)

  The function replaces existing matrix, located in the array by address (_N,_M), by the matrix mt.

Friends

- matrixC SumArr (const Arr2DC &)

  The function sums up all matrixes contained in the array.
- unsigned int StrArr (const Arr2DC &Arr)

  The function returns row counts.
- unsigned int ColArr (const Arr2DC &Arr)

  The function returns column counts.

6.2.1 Detailed Description

The array with (N-rows x M-columns) dimensions of small **complex-value** matrixes with (n x m) dimensions.

Example of using:

```
...
Arr2DC Arr(0,0,0,0); //creating an array
main()
{
  ...
  Arr = Arr2DC(1, 1, 2, 2); //changing the size
  Arr.ClearArr();   //clearing the array
  matrixC mt(2,2);
  Arr.insert(0,0, mt); //adding matrix mt to the array
  matrixC M=Arr(0,0); //taking matrix M from the array
  ...
}
```

Definition at line 138 of file PhysMtr.hpp.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 Arr2DC::Arr2DC (unsigned int *_N,* unsigned int *_M,* unsigned int *_n,* unsigned int *_m*) [inline]

Creates the array with (_N-rows x _M-columns) dimensions of small real-value matrixes with (_n x _m) dimensions.

Definition at line 148 of file PhysMtr.hpp.

6.2.2.2 Arr2DC::Arr2DC (void) [inline]

Definition at line 150 of file PhysMtr.hpp.

6.2.2.3 Arr2DC::Arr2DC (const Arr2DC & *Arr*)

Definition at line 167 of file PhysMtr.cpp.

6.2.2.4 Arr2DC::~Arr2DC (void) [inline]

Definition at line 154 of file PhysMtr.hpp.

6.2.3 Member Function Documentation

6.2.3.1 void Arr2DC::ClearArr (void) const

The function clears all elements of all matrixes in the array.

Definition at line 214 of file PhysMtr.cpp.

6.2.3.2 void Arr2DC::insert (unsigned int *_N,* unsigned int *_M,* const matrixC & *mt*)

The function adds the matrix mt to existing matrix, located in the array by address (_N,_M)

Parameters

_N,_M	number of a cell in the array
mt	adding matrix

Returns

none

Definition at line 221 of file PhysMtr.cpp.

6.2.3.3 matrixC Arr2DC::operator() (unsigned int *_N,* unsigned int *_M*) const

Returns the matrix stored in the array by address (_N,_M)

Definition at line 176 of file PhysMtr.cpp.

6.2.3.4 complex& Arr2DC::operator() (unsigned int *_N,* unsigned int *_M,* unsigned int *_n,* unsigned int *_m*) [inline]

Returns a reference to the matrix element (_n,_m) stored in the array by address (_N,_M)

Definition at line 160 of file PhysMtr.hpp.

6.2.3.5 complex Arr2DC::operator() (unsigned int *_N,* unsigned int *_M,* unsigned int *_n,* unsigned int *_m*) const [inline]

Returns the matrix element (_n,_m) stored in the array by address (_N,_M)

Definition at line 164 of file PhysMtr.hpp.

6.2.3.6 Arr2DC Arr2DC::operator+= (const Arr2DC & *arr*)

Definition at line 202 of file PhysMtr.cpp.

6.2.3.7 Arr2DC Arr2DC::operator/= (double *x*)

Definition at line 258 of file PhysMtr.cpp.

6.2.3.8 Arr2DC Arr2DC::operator= (const Arr2DC & *Arr*)

Definition at line 189 of file PhysMtr.cpp.

6.2.3.9 void Arr2DC::replace (unsigned int *_N,* unsigned int *_M,* const matrixC & *mt*)

The function replaces existing matrix, located in the array by address (_N,_M), by the matrix mt.

Parameters

_N,_M	number of a cell in the array
mt	new matrix

Returns

none

Definition at line 232 of file PhysMtr.cpp.

6.2.4 Friends And Related Function Documentation

6.2.4.1 unsigned int ColArr (const Arr2DC & *Arr*) [friend]

The function returns column counts.

Parameters

Arr	pointer to the array

Returns

column counts, (Arr.M)

Definition at line 211 of file PhysMtr.hpp.

6.2.4.2 unsigned int StrArr (const Arr2DC & *Arr*) [friend]

The function returns row counts.

Parameters

Arr	pointer to the array

Returns

row counts, (Arr.N)

Definition at line 204 of file PhysMtr.hpp.

6.2.4.3 matrixC SumArr (const Arr2DC & *Arr*) [friend]

The function sums up all matrixes contained in the array.

Parameters

Arr	pointer to the array

Returns

summed matrix

Definition at line 245 of file PhysMtr.cpp.

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

- PhysMtr.hpp
- PhysMtr.cpp

6.3 Beam Class Reference

The Beam class consist of 3D polygon, propagation direction and it's history, Jones matrix and so on.

`#include <Beam.hpp>`

Collaboration diagram for Beam:

Public Member Functions

- Beam (void)
- Beam (const std::list< Point3D > &_v, double l=0)
- ~Beam (void)
- Beam & operator= (const Beam &b)
- complex operator() (unsigned int n, unsigned int m) const

  < *Returns Jones matrix element of the beam*
- complex & operator() (unsigned int n, unsigned int m)
- matrixC operator() (void) const

  Returns Jones matrix of the beam.
- matrixC & operator() (void)

  Return reference to Jones matrix.
- void Clear (void)

  Clears beam vertex list.
- void ClearPr (void)

  Clears beam vertex projection list.
- std::list< Point3D >
  ::const_iterator Begin (void) const
- std::list< Point3D >
  ::const_iterator End (void) const
- std::list< unsigned int >
  ::const_iterator BeginP (void) const
- std::list< unsigned int >
  ::const_iterator EndP (void) const
- Polyg Projection (const double *const, int) const
- Beam & Rotate (Point3D k, Point3D Ey)

  Rotate Jones matrix of the beam to basis of scattering plane.
- Beam RotatePlane (const Point3D &NewE)

  Rotate Jones matrix to new basis.
- void PushFront (const Point3D &p)
- void PushBack (const Point3D &p)
- void PushFrontPr (const Point2D &p)
- void PushBackPr (const Point2D &p)
- void PushFrontP (const int &f)
- void PushBackP (const int &f)
- SphCrd Spherical (void) const

Public Attributes

- std::list< Point3D > v

  The beam's vertexes.
- std::list< unsigned int > path

  beam's trajectory
- std::list< Point2D > vpr

  Projection of the beam's vertexes on the normal plane.
- double lng

  optical path of the beam
- double D

  current position of phase front from Ax+By+Cz+D=0
- Point3D e

  the basis of polarization plane

- Point3D r

  direction of the beam in 3D space
- Point3D T
- Point3D F
- Point3D N

Friends

- unsigned int Size (const Beam &b)
- unsigned int SizeP (const Beam &b)
- double & OpticalPath (Beam &b)

  returns optical path
- Point3D CenterOfBeam (const Beam &)

  returns center of the beam
- double CrossSection (const Beam &)
- double AreaOfBeam (const Beam &)
- void OutBeam (char *, const Beam &)
- Point2D Proj_Vertex (const Beam &, Point3D &)
- void Rot (const Point3D &, const Point3D &, const Point3D &, matrixC &)

  Rotates coordinates system.

6.3.1 Detailed Description

The Beam class consist of 3D polygon, propagation direction and it's history, Jones matrix and so on.

Definition at line 25 of file Beam.hpp.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 Beam::Beam (void) [inline]

Definition at line 39 of file Beam.hpp.

6.3.2.2 Beam::Beam (const std::list< Point3D > & _v, double l = 0) [inline]

Definition at line 41 of file Beam.hpp.

6.3.2.3 Beam::~Beam (void) [inline]

Definition at line 44 of file Beam.hpp.

6.3.3 Member Function Documentation

6.3.3.1 std::list<Point3D>::const_iterator Beam::Begin (void) const [inline]

Definition at line 69 of file Beam.hpp.

6.3.3.2 std::list<unsigned int>::const_iterator Beam::BeginP (void) const [inline]

Definition at line 73 of file Beam.hpp.

6.3.3.3 void Beam::Clear (void) [inline]

Clears beam vertex list.

Definition at line 67 of file Beam.hpp.

6.3.3.4 void Beam::ClearPr (void) [inline]

Clears beam vertex projection list.

Definition at line 68 of file Beam.hpp.

6.3.3.5 std::list<Point3D>::const_iterator Beam::End (void) const [inline]

Definition at line 71 of file Beam.hpp.

6.3.3.6 std::list<unsigned int>::const_iterator Beam::EndP (void) const [inline]

Definition at line 75 of file Beam.hpp.

6.3.3.7 complex Beam::operator() (unsigned int *n*, unsigned int *m*) const [inline]

< Returns Jones matrix element of the beam

Definition at line 61 of file Beam.hpp.

6.3.3.8 complex& Beam::operator() (unsigned int *n*, unsigned int *m*) [inline]

Parameters

m	Return reference to Jones matrix element

Definition at line 63 of file Beam.hpp.

6.3.3.9 matrixC Beam::operator() (void) const [inline]

Returns Jones matrix of the beam.

Definition at line 65 of file Beam.hpp.

6.3.3.10 matrixC& Beam::operator() (void) [inline]

Return reference to Jones matrix.

Definition at line 66 of file Beam.hpp.

6.3.3.11 Beam& Beam::operator= (const Beam & *b*) [inline]

Definition at line 45 of file Beam.hpp.

6.3.3.12 Polyg Beam::Projection (const double * const *pp*, int *k*) const

Definition at line 40 of file beam.cpp.

6.3.3.13 void Beam::PushBack (const Point3D & *p*) [inline]

Definition at line 82 of file Beam.hpp.

6.3.3.14 void Beam::PushBackP (const int & *f*) [inline]

Definition at line 86 of file Beam.hpp.

6.3.3.15 void Beam::PushBackPr (const Point2D & *p*) [inline]

Definition at line 84 of file Beam.hpp.

6.3.3.16 void Beam::PushFront (const Point3D & *p*) [inline]

Definition at line 81 of file Beam.hpp.

6.3.3.17 void Beam::PushFrontP (const int & *f*) [inline]

Definition at line 85 of file Beam.hpp.

6.3.3.18 void Beam::PushFrontPr (const Point2D & *p*) [inline]

Definition at line 83 of file Beam.hpp.

6.3.3.19 Beam & Beam::Rotate (Point3D *k,* Point3D *Ey*)

Rotate Jones matrix of the beam to basis of scattering plane.

Definition at line 141 of file beam.cpp.

6.3.3.20 Beam Beam::RotatePlane (const Point3D & *NewE*)

Rotate Jones matrix to new basis.

Definition at line 165 of file beam.cpp.

6.3.3.21 SphCrd Beam::Spherical (void) const

Definition at line 159 of file beam.cpp.

6.3.4 Friends And Related Function Documentation

6.3.4.1 double AreaOfBeam (const Beam & *bm*) [friend]

Definition at line 108 of file beam.cpp.

6.3.4.2 Point3D CenterOfBeam (const Beam & *bm*) [friend]

returns center of the beam

Definition at line 74 of file beam.cpp.

6.3.4.3 double CrossSection (const Beam & *bm*) [friend]

Definition at line 84 of file beam.cpp.

6.3.4.4 double& OpticalPath (Beam & *b*) [friend]

returns optical path

Definition at line 91 of file Beam.hpp.

6.3.4.5 void OutBeam (char * *name,* const Beam & *bm*) [friend]

Definition at line 124 of file beam.cpp.

6.3.4.6 Point2D Proj_Vertex (const Beam & *bm,* Point3D & *pt*) [friend]

Definition at line 134 of file beam.cpp.

6.3.4.7 void Rot (const Point3D & *e,* const Point3D & *E,* const Point3D & *v,* matrixC & *m*) [friend]

Rotates coordinates system.

Definition at line 12 of file beam.cpp.

6.3.4.8 unsigned int Size (const Beam & *b*) [friend]

Definition at line 89 of file Beam.hpp.

6.3.4.9 unsigned int SizeP (const Beam & *b*) [friend]

Definition at line 90 of file Beam.hpp.

6.3.5 Member Data Documentation

6.3.5.1 double Beam::D

current position of phase front from Ax+By+Cz+D=0

Definition at line 33 of file Beam.hpp.

6.3.5.2 Point3D Beam::e

the basis of polarization plane

Definition at line 35 of file Beam.hpp.

6.3.5.3 Point3D Beam::F

Definition at line 35 of file Beam.hpp.

6.3.5.4 double Beam::lng

optical path of the beam

Definition at line 33 of file Beam.hpp.

6.3.5.5 Point3D Beam::N

Definition at line 35 of file Beam.hpp.

6.3.5.6 std::list<unsigned int> Beam::path

beam's trajectory

Definition at line 30 of file Beam.hpp.

6.3.5.7 Point3D Beam::r

direction of the beam in 3D space

Definition at line 35 of file Beam.hpp.

6.3.5.8 Point3D Beam::T

Definition at line 35 of file Beam.hpp.

6.3.5.9 std::list<Point3D> Beam::v

The beam's vertexes.

Definition at line 29 of file Beam.hpp.

6.3.5.10 std::list<Point2D> Beam::vpr

Projection of the beam's vertexes on the normal plane.

Definition at line 31 of file Beam.hpp.

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

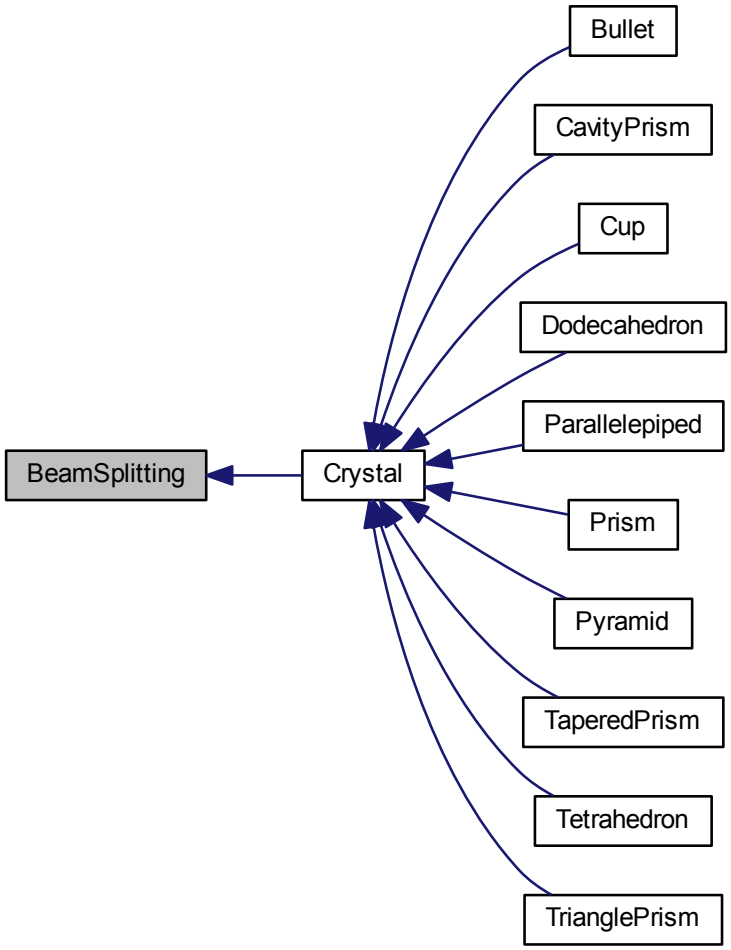
- Beam.hpp
- beam.cpp

6.4 BeamSplitting Class Reference

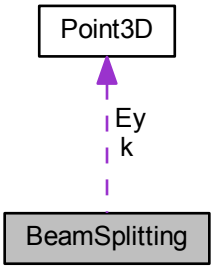
This class is a prototype for Crystal class. It contains parameters for BeamSplitting algorithm.

`#include <Crystal.hpp>`

Inheritance diagram for BeamSplitting:



Collaboration diagram for BeamSplitting:



Public Member Functions

- BeamSplitting (unsigned int _Itr, unsigned int _NoF, Point3D _k, Point3D _Ey)
- ~BeamSplitting (void)

Protected Attributes

- unsigned int Itr
  Number of internal reflection.
- unsigned int NoF
  Number of facets.
- unsigned int ** Facets
  Mask for calculation only specified trajectories.
- Point3D k
  Direction to the incident wave.
- Point3D Ey
  Normal to the incident plane.
- double Ep
  the level of negligibility energy
- double d
  distance to far zone
- double S_eps
  Minimal area of a beam.

6.4.1 Detailed Description

This class is a prototype for Crystal class. It contains parameters for BeamSplitting algorithm.

Definition at line 35 of file Crystal.hpp.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 BeamSplitting::BeamSplitting (unsigned int *_Itr,* unsigned int *_NoF,* Point3D *_k,* Point3D *_Ey*) [inline]

Definition at line 48 of file Crystal.hpp.

6.4.2.2 BeamSplitting::~BeamSplitting (void) [inline]

Definition at line 51 of file Crystal.hpp.

6.4.3 Member Data Documentation

6.4.3.1 double BeamSplitting::d [protected]

distance to far zone

Definition at line 44 of file Crystal.hpp.

6.4.3.2 double BeamSplitting::Ep [protected]

the level of negligibility energy

Definition at line 44 of file Crystal.hpp.

6.4.3.3 Point3D BeamSplitting::Ey [protected]

Normal to the incident plane.

Definition at line 42 of file Crystal.hpp.

6.4.3.4 unsigned int ** BeamSplitting::Facets [protected]

Mask for calculation only specified trajectories.

Definition at line 39 of file Crystal.hpp.

6.4.3.5 unsigned int BeamSplitting::Itr [protected]

Number of internal reflection.

Definition at line 39 of file Crystal.hpp.

6.4.3.6 Point3D BeamSplitting::k [protected]

Direction to the incident wave.

Definition at line 42 of file Crystal.hpp.

6.4.3.7 unsigned int BeamSplitting::NoF [protected]

Number of facets.

Definition at line 39 of file Crystal.hpp.

6.4.3.8 double BeamSplitting::S_eps [protected]

Minimal area of a beam.

Definition at line 44 of file Crystal.hpp.

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

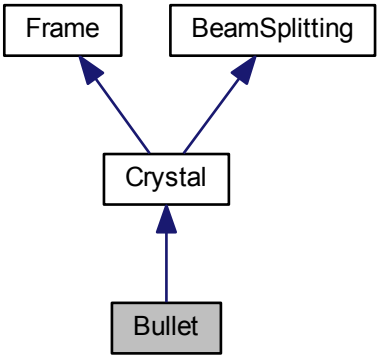
- Crystal.hpp
- Crystal.cpp

6.5 Bullet Class Reference

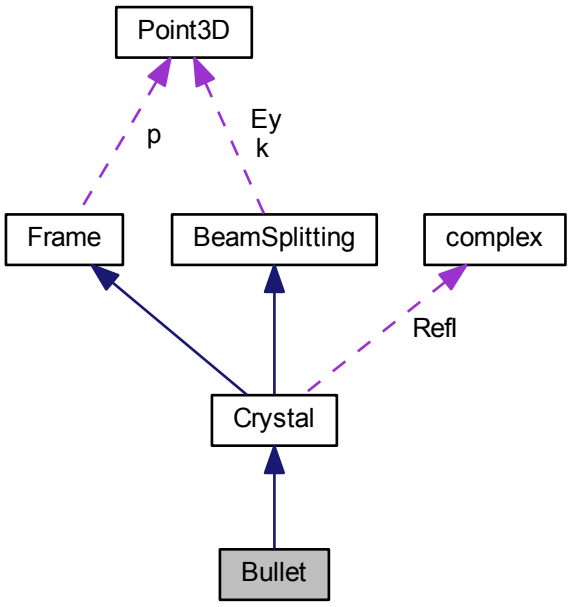
This class defines the hexagonal bullet.

`#include <particle.hpp>`

Inheritance diagram for Bullet:



Collaboration diagram for Bullet:



Public Member Functions

- Bullet (const complex &r, double W, double H, double HP, unsigned int Itr, Point3D _k, Point3D _Ey)
- virtual ~Bullet (void)
- double First (void)

  Returns a parameter of the size of the crystal, depends on the realization.

- double Second (void)

  Returns a parameter of the size of the crystal, depends on the realization.
- double Third (void)

  Returns a parameter of the size of the crystal, depends on the realization.
- void SetVertices (void)

  This function defines the coordinates of the vertexes.
- void SetFacets (void)

Additional Inherited Members

6.5.1 Detailed Description

This class defines the hexagonal bullet.

Definition at line 60 of file particle.hpp.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 Bullet::Bullet (const complex & *r,* double *W,* double *H,* double *HP,* unsigned int *ltr,* Point3D *_k,* Point3D *_Ey*) [inline]

Definition at line 66 of file particle.hpp.

6.5.2.2 virtual Bullet::~Bullet (void) [inline], [virtual]

Definition at line 69 of file particle.hpp.

6.5.3 Member Function Documentation

6.5.3.1 double Bullet::First (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented from Frame.

Definition at line 71 of file particle.hpp.

6.5.3.2 double Bullet::Second (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented from Frame.

Definition at line 72 of file particle.hpp.

6.5.3.3 void Bullet::SetFacets (void)

Definition at line 88 of file particle.cpp.

6.5.3.4 void Bullet::SetVertices (void) [virtual]

This function defines the coordinates of the vertexes.

Reimplemented from Crystal.

Definition at line 69 of file particle.cpp.

6.5.3.5 double Bullet::Third (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented from Frame.

Definition at line 73 of file particle.hpp.

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

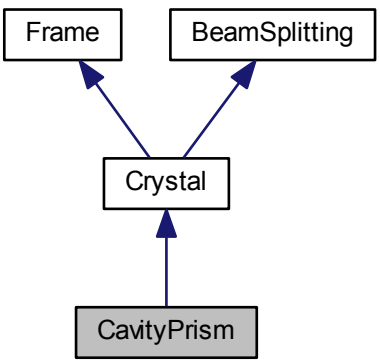
- particle.hpp
- particle.cpp

6.6 CavityPrism Class Reference

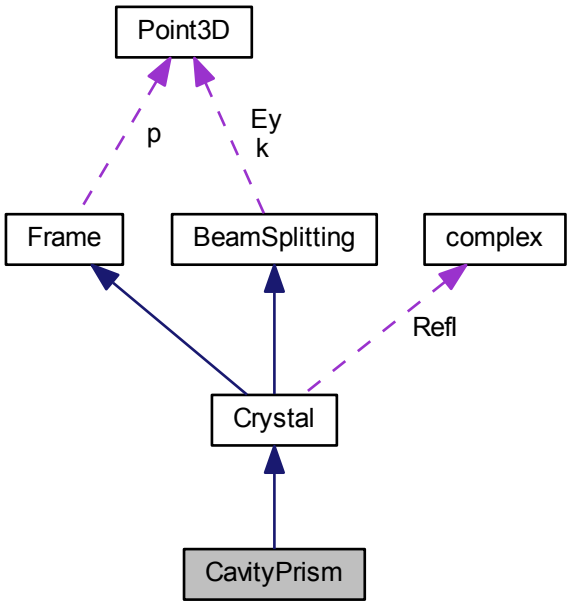
This class defines the hexagonal prism with cavity on bottom.

`#include <particle.hpp>`

Inheritance diagram for CavityPrism:



Collaboration diagram for CavityPrism:



Public Member Functions

- CavityPrism (const complex &r, double W, double H, double C, unsigned int ltr, Point3D _k, Point3D _Ey)
- virtual ~CavityPrism (void)
- double First (void)

  Returns a parameter of the size of the crystal, depends on the realization.
- double Second (void)

  Returns a parameter of the size of the crystal, depends on the realization.
- double Third (void)

  Returns a parameter of the size of the crystal, depends on the realization.
- void SetVertices (void)

  This function defines the coordinates of the vertexes.
- void SetFacets (void)

Additional Inherited Members

6.6.1 Detailed Description

This class defines the hexagonal prism with cavity on bottom.

Definition at line 101 of file particle.hpp.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 CavityPrism::CavityPrism (const complex & *r,* double *W,* double *H,* double *C,* unsigned int *ltr,* Point3D *_k,* Point3D *_Ey*) [inline]

Definition at line 107 of file particle.hpp.

6.6.2.2 virtual CavityPrism::~CavityPrism (void) [inline], [virtual]

Definition at line 110 of file particle.hpp.

6.6.3 Member Function Documentation

6.6.3.1 double CavityPrism::First (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented from Frame.

Definition at line 112 of file particle.hpp.

6.6.3.2 double CavityPrism::Second (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented from Frame.

Definition at line 113 of file particle.hpp.

6.6.3.3 void CavityPrism::SetFacets (void)

Definition at line 168 of file particle.cpp.

6.6.3.4 void CavityPrism::SetVertices (void) [virtual]

This function defines the coordinates of the vertexes.

Reimplemented from Crystal.

Definition at line 149 of file particle.cpp.

6.6.3.5 double CavityPrism::Third (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented from Frame.

Definition at line 114 of file particle.hpp.

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

- particle.hpp
- particle.cpp

6.7 Chain Class Reference

The class represents chain of number of facets.

`#include <trajectory.hpp>`

Public Member Functions

- Chain (const std::list< unsigned int > &_Ch)
- virtual ~Chain ()
- unsigned int Size (void) const
- std::list< unsigned int > ::const_iterator Begin (void) const
- std::list< unsigned int > ::const_iterator End (void) const

Public Attributes

- std::list< unsigned int > Ch
- unsigned int sz

6.7.1 Detailed Description

The class represents chain of number of facets.

This class allows to contain the trajectory that had been defined in input data file. This allows us to calculate only the trajectories we are interested in, thereby highly decrease calculation time.

Definition at line 22 of file trajectory.hpp.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 Chain::Chain (const std::list< unsigned int > & *_Ch*) [inline]

Definition at line 26 of file trajectory.hpp.

6.7.2.2 virtual Chain::~Chain ( ) [inline], [virtual]

Definition at line 28 of file trajectory.hpp.

6.7.3 Member Function Documentation

6.7.3.1 std::list<unsigned int>::const_iterator Chain::Begin (void) const [inline]

Definition at line 30 of file trajectory.hpp.

6.7.3.2 std::list<unsigned int>::const_iterator Chain::End (void) const [inline]

Definition at line 32 of file trajectory.hpp.

6.7.3.3 unsigned int Chain::Size (void) const [inline]

Definition at line 29 of file trajectory.hpp.

6.7.4 Member Data Documentation

6.7.4.1 std::list<unsigned int> Chain::Ch

Definition at line 24 of file trajectory.hpp.

6.7.4.2 unsigned int Chain::sz

Definition at line 25 of file trajectory.hpp.

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

- trajectory.hpp

6.8 complex Class Reference

This class provides a complex numbers and operation with them.

`#include <compl.hpp>`

Public Member Functions

- complex (double r=0, double i=0)
- complex & operator= (const complex &b)
- complex operator+ () const
- complex operator+ (double x) const
- complex operator+ (const complex &z) const
- complex operator+= (const complex &z)
- complex operator+= (double x)
- complex operator- () const
- complex operator- (double x) const
- complex operator- (const complex &z) const
- complex operator-= (const complex &z)
- complex operator-= (double x)
- complex operator∗ (double x) const
- complex operator∗ (const complex &z) const
- complex operator∗= (const complex &z)
- complex operator∗= (double x)
- complex operator/ (double x) const
- complex operator/ (const complex &z) const
- complex operator/= (double x)
- complex operator/= (const complex &z)
- bool operator== (const complex &z) const
- bool operator!= (const complex &z) const
- bool operator== (double x) const
- bool operator!= (double x) const

Friends

- complex operator+ (double x, const complex &z)
- complex operator- (double x, const complex &z)
- complex operator/ (double x, const complex &z)
- double & real (complex &z)
- double & imag (complex &z)
- double real (const complex &z)
- double imag (const complex &z)
- double norm (const complex &z)
- complex conj (const complex &z)
- complex exp (const complex &z)
- complex operator∗ (double x, const complex &z)

- double arg (const complex &z)
- double abs (const complex &z)
- complex sqrt (const complex &)

6.8.1 Detailed Description

This class provides a complex numbers and operation with them.

Definition at line 29 of file compl.hpp.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 complex::complex (double *r* = 0, double *i* = 0) [inline]

Definition at line 32 of file compl.hpp.

6.8.3 Member Function Documentation

6.8.3.1 bool complex::operator!= (const complex & *z*) const [inline]

Definition at line 85 of file compl.hpp.

6.8.3.2 bool complex::operator!= (double *x*) const [inline]

Definition at line 89 of file compl.hpp.

6.8.3.3 complex complex::operator* (double *x*) const [inline]

Definition at line 55 of file compl.hpp.

6.8.3.4 complex complex::operator* (const complex & *z*) const [inline]

Definition at line 57 of file compl.hpp.

6.8.3.5 complex complex::operator*= (const complex & *z*) [inline]

Definition at line 62 of file compl.hpp.

6.8.3.6 complex complex::operator*= (double *x*) [inline]

Definition at line 67 of file compl.hpp.

6.8.3.7 complex complex::operator+ ( ) const [inline]

Definition at line 36 of file compl.hpp.

6.8.3.8 complex complex::operator+ (double *x*) const [inline]

Definition at line 37 of file compl.hpp.

6.8.3.9 complex complex::operator+ (const complex & *z*) const [inline]

Definition at line 39 of file compl.hpp.

6.8.3.10 complex complex::operator+= (const complex & *z*) [inline]

Definition at line 41 of file compl.hpp.

6.8.3.11 complex complex::operator+= (double *x*) [inline]

Definition at line 43 of file compl.hpp.

6.8.3.12 complex complex::operator- ( ) const [inline]

Definition at line 45 of file compl.hpp.

6.8.3.13 complex complex::operator- (double *x*) const [inline]

Definition at line 47 of file compl.hpp.

6.8.3.14 complex complex::operator- (const complex & *z*) const [inline]

Definition at line 49 of file compl.hpp.

6.8.3.15 complex complex::operator-= (const complex & *z*) [inline]

Definition at line 51 of file compl.hpp.

6.8.3.16 complex complex::operator-= (double *x*) [inline]

Definition at line 53 of file compl.hpp.

6.8.3.17 complex complex::operator/ (double *x*) const [inline]

Definition at line 69 of file compl.hpp.

6.8.3.18 complex complex::operator/ (const complex & *z*) const [inline]

Definition at line 71 of file compl.hpp.

6.8.3.19 complex complex::operator/= (double *x*) [inline]

Definition at line 80 of file compl.hpp.

6.8.3.20 complex complex::operator/= (const complex & *z*) [inline]

Definition at line 82 of file compl.hpp.

6.8.3.21 complex& complex::operator= (const complex & *b*) [inline]

Definition at line 34 of file compl.hpp.

6.8.3.22 bool complex::operator== (const complex & *z*) const [inline]

Definition at line 83 of file compl.hpp.

6.8.3.23 bool complex::operator== (double *x*) const [inline]

Definition at line 87 of file compl.hpp.

6.8.4 Friends And Related Function Documentation

6.8.4.1 double abs (const complex & *z*) [friend]

Definition at line 118 of file compl.hpp.

6.8.4.2 double arg (const complex & *z*) [friend]

Definition at line 116 of file compl.hpp.

6.8.4.3 complex conj (const complex & *z*) [friend]

Definition at line 110 of file compl.hpp.

6.8.4.4 complex exp (const complex & *z*) [friend]

Definition at line 112 of file compl.hpp.

6.8.4.5 double& imag (complex & *z*) [friend]

Definition at line 105 of file compl.hpp.

6.8.4.6 double imag (const complex & *z*) [friend]

Definition at line 107 of file compl.hpp.

6.8.4.7 double norm (const complex & *z*) [friend]

Definition at line 108 of file compl.hpp.

6.8.4.8 complex operator* (double *x,* const complex & *z*) [friend]

Definition at line 114 of file compl.hpp.

6.8.4.9 complex operator+ (double *x,* const complex & *z*) [friend]

Definition at line 92 of file compl.hpp.

6.8.4.10 complex operator- (double *x*, const complex & *z*) [friend]

Definition at line 94 of file compl.hpp.

6.8.4.11 complex operator/ (double *x*, const complex & *z*) [friend]

Definition at line 96 of file compl.hpp.

6.8.4.12 double& real (complex & *z*) [friend]

Definition at line 104 of file compl.hpp.

6.8.4.13 double real (const complex & *z*) [friend]

Definition at line 106 of file compl.hpp.

6.8.4.14 complex sqrt (const complex & *z*) [friend]

Definition at line 18 of file compl.cpp.

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

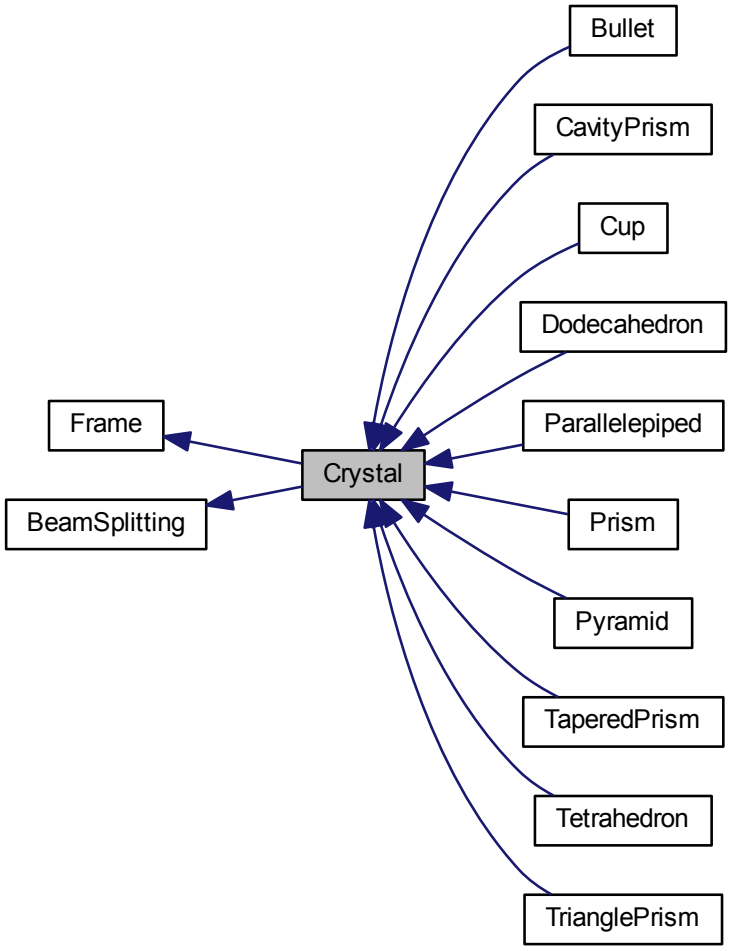
- compl.hpp

6.9 Crystal Class Reference

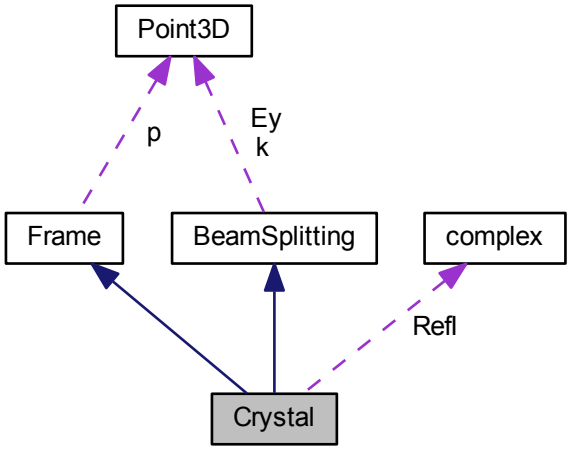
This class contains all information about particle's geometry and the parameters of tracing.

`#include <Crystal.hpp>`

Inheritance diagram for Crystal:



Collaboration diagram for Crystal:



Public Member Functions

- virtual ~Crystal (void)
- bool & Phase (void)

  The function sets whether the optical path is taking into account or not.
- bool Phase (void) const

  The function returns if the optical path is taking into account or not.
- complex RefIn (void) const

  The function returns complex value of the refraction index.
- Point3D NormToFacet (unsigned int i) const

  The function returns a normal to the face number i.
- Point3D VertexOfFacet (unsigned int i, unsigned int j) const

  The function returns coordinates of vertex number j on the face number i.
- const double * Facet (unsigned int i) const

  *The function returns a pointer on an array with coefficients of an equation of plane which contains the facet number **i** of the crystal.*
- Polyg Projection (unsigned int i, int k) const

  *The function returns a projection of a facet number **i** to the one of 2D planes: YZ (k=0), XZ (k=1), XY (k=2).*
- std::list< Point3D > Retrieve (Polyg &ProjectionPolygon, unsigned int i, int k) const
- virtual void ChangePosition (double t, double p, double f)

  The function rotates the particle by Euler's angles: Betta, Gamma, Alpha.
- Point3D CenterOfFacet (unsigned int i) const

  *The function returns the center of gravity of the facet number **i**.*
- double AreaOfFacet (unsigned int i) const

  *The function returns the area of the facet number **i**.*
- double FTforConvexCrystal (Hand Handler) const

  This function runs the beam splitting algorithm. The realization is in scattering.cpp.
- bool Intersection (const Beam &Bm, unsigned int i, std::list< Point3D > &pl) const

Protected Member Functions

- Crystal (const complex &r, unsigned int m, unsigned int k, unsigned int km, unsigned int ltr, Point3D _k, Point3D _Ey)

  The complex value of refraction index of the particle.
- virtual void SetVertices (void)

  This function defines the coordinates of the vertexes.
- void ChangeVertices (double bt, double gm)

  The function rotates the vertex of particle by Euler's angles: Betta, Gamma. (Alpha = 0)
- void ChangeVertices (double bt, double gm, double al)

  The function rotates the vertex of particle by Euler's angles: Betta, Gamma, Alpha.
- void ChangeFacets (void)

  Sets up facet's normals.

Protected Attributes

- bool PhaseDelay

  This flag specifies whether the optical path is taking into account or not.
- double mi
- double mr

  Imaginary part of refraction index.
- complex RefI

  Real part of refraction index.

6.9.1 Detailed Description

This class contains all information about particle's geometry and the parameters of tracing.

Definition at line 103 of file Crystal.hpp.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 Crystal::Crystal (const complex & *r,* unsigned int *m,* unsigned int *k,* unsigned int *km,* unsigned int *ltr,* Point3D *_k,* Point3D *_Ey*) [inline], [protected]

The complex value of refraction index of the particle.

Definition at line 114 of file Crystal.hpp.

6.9.2.2 virtual Crystal::~Crystal (void) [inline], [virtual]

Definition at line 132 of file Crystal.hpp.

6.9.3 Member Function Documentation

6.9.3.1 double Crystal::AreaOfFacet (unsigned int *i*) const

The function returns the area of the facet number **i**.

Definition at line 185 of file Crystal.cpp.

6.9.3.2 Point3D Crystal::CenterOfFacet (unsigned int *i*) const

The function returns the center of gravity of the facet number **i**.

Definition at line 176 of file Crystal.cpp.

6.9.3.3 void Crystal::ChangeFacets (void) [protected]

Sets up facet's normals.

Definition at line 248 of file Crystal.cpp.

6.9.3.4 virtual void Crystal::ChangePosition (double *t,* double *p,* double *f*) [inline], [virtual]

The function rotates the particle by Euler's angles: Betta, Gamma, Alpha.

Definition at line 157 of file Crystal.hpp.

6.9.3.5 void Crystal::ChangeVertices (double *bt,* double *gm*) [protected]

The function rotates the vertex of particle by Euler's angles: Betta, Gamma. (Alpha = 0)

Definition at line 229 of file Crystal.cpp.

6.9.3.6 void Crystal::ChangeVertices (double *bt,* double *gm,* double *al*) [protected]

The function rotates the vertex of particle by Euler's angles: Betta, Gamma, Alpha.

Definition at line 205 of file Crystal.cpp.

6.9.3.7 const double* Crystal::Facet (unsigned int *i*) const [inline]

The function returns a pointer on an array with coefficients of an equation of plane which contains the facet number **i** of the crystal.

Definition at line 150 of file Crystal.hpp.

6.9.3.8 double Crystal::FTforConvexCrystal (Hand *Handler*) const

This function runs the beam splitting algorithm. The realization is in scattering.cpp.

Parameters

Handler	Function to handle outgoing beam

Returns

Scattering Cross-Section

Definition at line 157 of file Scattering.cpp.

6.9.3.9 bool Crystal::Intersection (const Beam & *Bm,* unsigned int *i,* std::list< Point3D > & *pl*) const

The function calculates an intersection between a beam **Bm** and the facet number **i**. The result of intersection is in **pl**.

Returns

true, if there is an intersection

Definition at line 117 of file Crystal.cpp.

6.9.3.10 Point3D Crystal::NormToFacet (unsigned int *i*) const [inline]

The function returns a normal to the face number i.

Definition at line 144 of file Crystal.hpp.

6.9.3.11 bool& Crystal::Phase (void) [inline]

The function sets whether the optical path is taking into account or not.

Definition at line 135 of file Crystal.hpp.

6.9.3.12 bool Crystal::Phase (void) const [inline]

The function returns if the optical path is taking into account or not.

Definition at line 138 of file Crystal.hpp.

6.9.3.13 Polyg Crystal::Projection (unsigned int *i,* int *k*) const

The function returns a projection of a facet number **i** to the one of 2D planes: YZ (k=0), XZ (k=1), XY (k=2).

Definition at line 65 of file Crystal.cpp.

6.9.3.14 complex Crystal::RefIn (void) const [inline]

The function returns complex value of the refraction index.

Definition at line 141 of file Crystal.hpp.

6.9.3.15 std::list< Point3D > Crystal::Retrieve (Polyg & *ProjectionPolygon,* unsigned int *i,* int *k*) const

Definition at line 94 of file Crystal.cpp.

6.9.3.16 virtual void Crystal::SetVertices (void) [inline], [protected], [virtual]

This function defines the coordinates of the vertexes.

Reimplemented in Dodecahedron, Tetrahedron, Parallelepiped, TrianglePrism, Cup, CavityPrism, TaperedPrism, Bullet, Pyramid, and Prism.

Definition at line 121 of file Crystal.hpp.

6.9.3.17 Point3D Crystal::VertexOfFacet (unsigned int *i,* unsigned int *j*) const [inline]

The function returns coordinates of vertex number j on the face number i.

Definition at line 147 of file Crystal.hpp.

6.9.4 Member Data Documentation

6.9.4.1 double Crystal::mi [protected]

Definition at line 111 of file Crystal.hpp.

6.9.4.2 double Crystal::mr [protected]

Imaginary part of refraction index.

Definition at line 111 of file Crystal.hpp.

6.9.4.3 bool Crystal::PhaseDelay [protected]

This flag specifies whether the optical path is taking into account or not.

Definition at line 110 of file Crystal.hpp.

6.9.4.4 complex Crystal::Refl [protected]

Real part of refraction index.

Definition at line 113 of file Crystal.hpp.

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

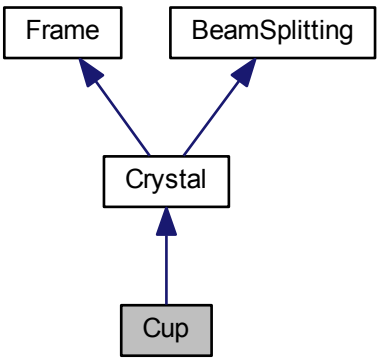
- Crystal.hpp
- Crystal.cpp
- Scattering.cpp

6.10 Cup Class Reference

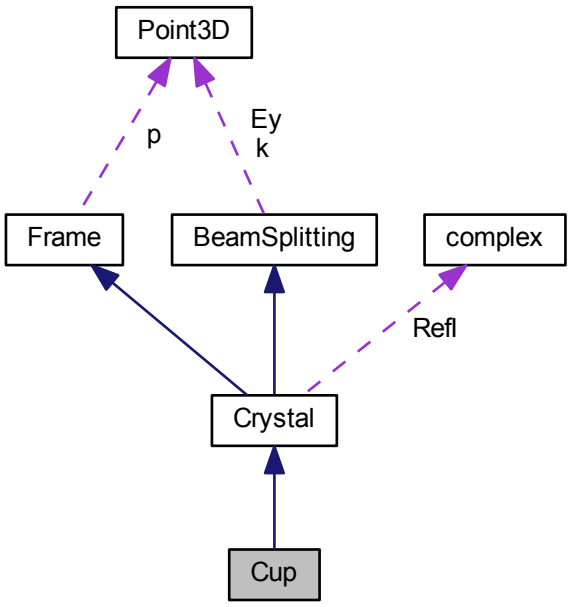
This class defines the cupped prism.

`#include <particle.hpp>`

Inheritance diagram for Cup:



Collaboration diagram for Cup:



Public Member Functions

- Cup (const complex &r, double WP, double HP, double WC, double HC, unsigned int Itr, Point3D _k, Point3D _Ey)
- virtual ~Cup (void)
- double First (void)

  Returns a parameter of the size of the crystal, depends on the realization.
- double Second (void)

  Returns a parameter of the size of the crystal, depends on the realization.
- double Third (void)

  Returns a parameter of the size of the crystal, depends on the realization.
- double Forth (void)

  Returns a parameter of the size of the crystal, depends on the realization.
- void SetVertices (void)

  This function defines the coordinates of the vertexes.
- void SetFacets (void)

Additional Inherited Members

6.10.1 Detailed Description

This class defines the cupped prism.

Definition at line 121 of file particle.hpp.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 Cup::Cup (const complex & *r*, double *WP*, double *HP*, double *WC*, double *HC*, unsigned int *ltr*, Point3D *_k*, Point3D *_Ey*) [inline]

Definition at line 127 of file particle.hpp.

6.10.2.2 virtual Cup::~Cup (void) [inline], [virtual]

Definition at line 130 of file particle.hpp.

6.10.3 Member Function Documentation

6.10.3.1 double Cup::First (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented from Frame.

Definition at line 132 of file particle.hpp.

6.10.3.2 double Cup::Forth (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented from Frame.

Definition at line 135 of file particle.hpp.

6.10.3.3 double Cup::Second (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented from Frame.

Definition at line 133 of file particle.hpp.

6.10.3.4 void Cup::SetFacets (void)

Definition at line 226 of file particle.cpp.

6.10.3.5 void Cup::SetVertices (void) [virtual]

This function defines the coordinates of the vertexes.

Reimplemented from Crystal.

Definition at line 188 of file particle.cpp.

6.10.3.6 double Cup::Third (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented from Frame.

Definition at line 134 of file particle.hpp.

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

- particle.hpp

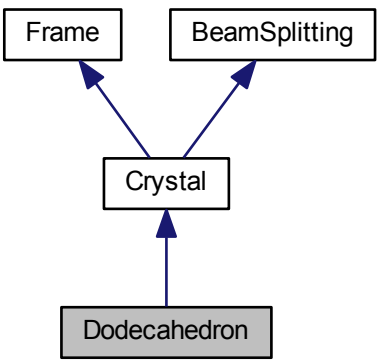
- particle.cpp

6.11 Dodecahedron Class Reference

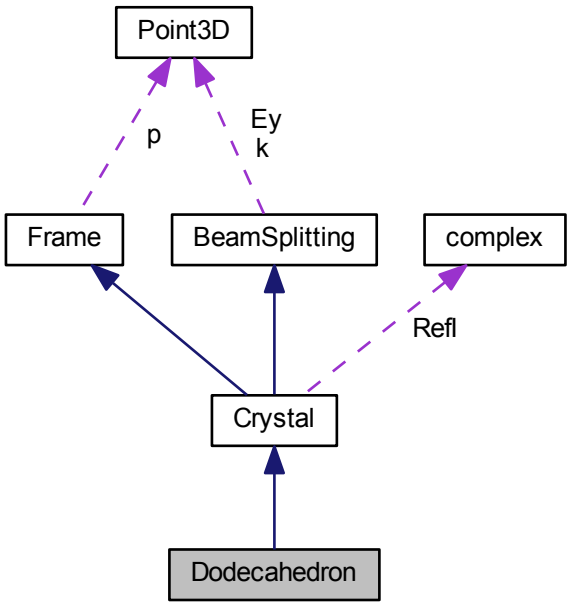
This class defines the dodecahedron.

`#include <particle.hpp>`

Inheritance diagram for Dodecahedron:



Collaboration diagram for Dodecahedron:



Public Member Functions

- Dodecahedron (const complex &r, double W, double H, unsigned int Itr, Point3D _k, Point3D _Ey)
- virtual ~Dodecahedron (void)
- double First (void)

  Returns a parameter of the size of the crystal, depends on the realization.
- double Second (void)

  Returns a parameter of the size of the crystal, depends on the realization.
- void SetVertices (void)

  This function defines the coordinates of the vertexes.
- void SetFacets (void)

Additional Inherited Members

6.11.1 Detailed Description

This class defines the dodecahedron.

Definition at line 198 of file particle.hpp.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 Dodecahedron::Dodecahedron (const complex & *r*, double *W*, double *H*, unsigned int *Itr*, Point3D *_k*, Point3D *_Ey*) [inline]

Definition at line 205 of file particle.hpp.

6.11.2.2 virtual Dodecahedron::~Dodecahedron (void) [inline], [virtual]

Definition at line 208 of file particle.hpp.

6.11.3 Member Function Documentation

6.11.3.1 double Dodecahedron::First (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented from Frame.

Definition at line 210 of file particle.hpp.

6.11.3.2 double Dodecahedron::Second (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented from Frame.

Definition at line 211 of file particle.hpp.

6.11.3.3 void Dodecahedron::SetFacets (void)

Definition at line 339 of file particle.cpp.

6.11.3.4 void Dodecahedron::SetVertices (void) [virtual]

This function defines the coordinates of the vertexes.

Reimplemented from Crystal.

Definition at line 313 of file particle.cpp.

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

- particle.hpp
- particle.cpp

6.12 Edge Class Reference

The Edge class represents all forms of lines in 2D.

`#include <Geometry.hpp>`

Collaboration diagram for Edge:

Public Member Functions

- Edge (const Point2D &_org, const Point2D &_dest)
- Edge (void)
- Edge & operator∗= (double x)
- Edge operator∗ (double x) const
- Edge & rot90 (void)

  rotation of the edge through angle 90 deg.
- Edge & flip (void)
- Point2D point (double t) const
- state intersect (const Edge &, double &) const

  intersection of 2D edges
- double slope (void) const
- double y (double t) const
- double signedAngle (const Point2D &a) const

Public Attributes

- Point2D org
- Point2D dest

Friends

- double length (const Edge &e)
- direction parallel (const Edge &e, const Edge &f)
- state crossingPoint (const Edge &, const Edge &, Point2D &)

6.12.1 Detailed Description

The Edge class represents all forms of lines in 2D.

See M.Laszlo "Computational Geometry and Computer Graphics in C++", page 88.

Definition at line 114 of file Geometry.hpp.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 Edge::Edge (const Point2D & *_org,* const Point2D & *_dest*) [inline]

Definition at line 117 of file Geometry.hpp.

6.12.2.2 Edge::Edge (void) [inline]

Definition at line 119 of file Geometry.hpp.

6.12.3 Member Function Documentation

6.12.3.1 Edge& Edge::flip (void) [inline]

Definition at line 131 of file Geometry.hpp.

6.12.3.2 state Edge::intersect (const Edge & *e,* double & *t*) const

intersection of 2D edges

Definition at line 145 of file Geometry.cpp.

6.12.3.3 Edge Edge::operator∗ (double *x*) const [inline]

Definition at line 123 of file Geometry.hpp.

6.12.3.4 Edge& Edge::operator∗= (double *x*) [inline]

Definition at line 121 of file Geometry.hpp.

6.12.3.5 Point2D Edge::point (double *t*) const [inline]

Definition at line 133 of file Geometry.hpp.

6.12.3.6 Edge& Edge::rot90 (void) [inline]

rotation of the edge through angle 90 deg.

Definition at line 125 of file Geometry.hpp.

6.12.3.7 double Edge::signedAngle (const Point2D & *a*) const

Definition at line 79 of file Geometry.cpp.

6.12.3.8 double Edge::slope (void) const [inline]

$<$ the slope of the edge

Definition at line 136 of file Geometry.hpp.

6.12.3.9 double Edge::y (double *t*) const [inline]

Definition at line 141 of file Geometry.hpp.

6.12.4 Friends And Related Function Documentation

6.12.4.1 state crossingPoint (const Edge & *e*, const Edge & *f*, Point2D & *p*) [friend]

Definition at line 368 of file Geometry.cpp.

6.12.4.2 double length (const Edge & *e*) [friend]

Definition at line 145 of file Geometry.hpp.

6.12.4.3 direction parallel (const Edge & *e*, const Edge & *f*) [friend]

Definition at line 162 of file Geometry.cpp.

6.12.5 Member Data Documentation

6.12.5.1 Point2D Edge::dest

Definition at line 116 of file Geometry.hpp.

6.12.5.2 Point2D Edge::org

Definition at line 116 of file Geometry.hpp.

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

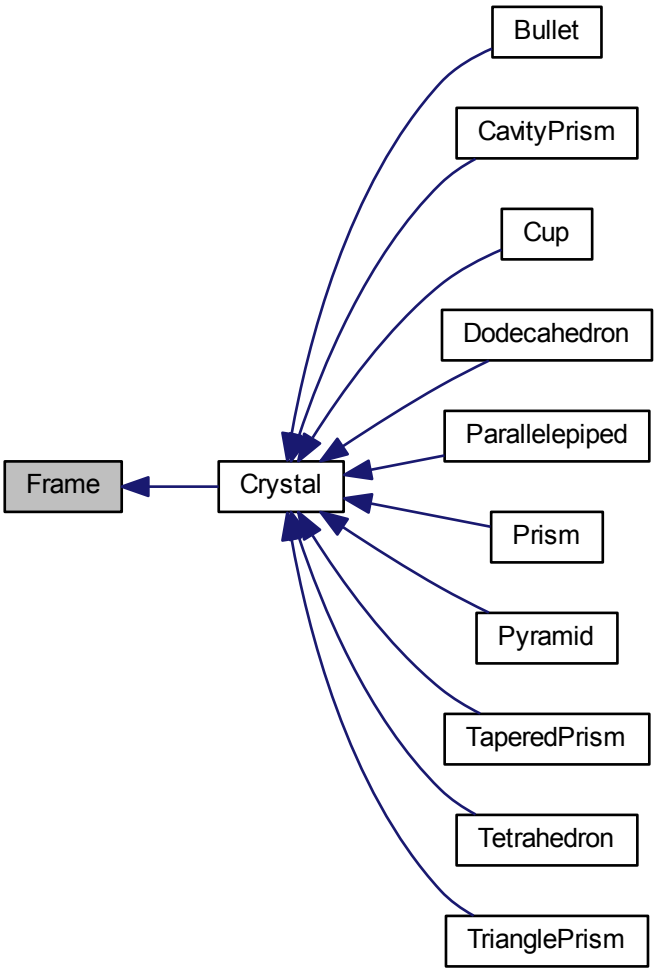
- Geometry.hpp
- Geometry.cpp

6.13 Frame Class Reference

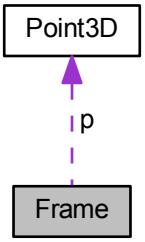
This class is a prototype for Crystal class.

`#include <Crystal.hpp>`

Inheritance diagram for Frame:



Collaboration diagram for Frame:



Public Member Functions

- Frame operator= (const Frame &)
- ~Frame (void)
- virtual double First (void)

  Returns a parameter of the size of the crystal, depends on the realization.
- virtual double Second (void)

  Returns a parameter of the size of the crystal, depends on the realization.
- virtual double Third (void)

  Returns a parameter of the size of the crystal, depends on the realization.
- virtual double Forth (void)

  Returns a parameter of the size of the crystal, depends on the realization.

Protected Member Functions

- Frame (int m, int k, int km)
- Frame (const Frame &f)

Protected Attributes

- int ** Gr

  Numbers of the vertexes from which the facet consist of.
- unsigned int M

  Total count of the crystal vertexes.
- unsigned int K

  Total count of the facets.
- unsigned int Km

  Maximal count of vertexes at any facet.
- double ** Gran

  The coefficients of an equation of plane which contains the facet of the crystal.
- Point3D * p

  Coordinates of vertexes of the crystal.

6.13.1 Detailed Description

This class is a prototype for Crystal class.

Definition at line 58 of file Crystal.hpp.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 Frame::Frame (int *m,* int *k,* int *km*) [inline], [protected]

Definition at line 78 of file Crystal.hpp.

6.13.2.2 Frame::Frame (const Frame & *f*) [inline], [protected]

Definition at line 80 of file Crystal.hpp.

6.13.2.3 Frame::~Frame (void) [inline]

Definition at line 85 of file Crystal.hpp.

6.13.3 Member Function Documentation

6.13.3.1 virtual double Frame::First (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented in Dodecahedron, Tetrahedron, Parallelepiped, TrianglePrism, Cup, CavityPrism, TaperedPrism, Bullet, Pyramid, and Prism.

Definition at line 88 of file Crystal.hpp.

6.13.3.2 virtual double Frame::Forth (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented in Parallelepiped, Cup, and TaperedPrism.

Definition at line 94 of file Crystal.hpp.

6.13.3.3 Frame Frame::operator= (const Frame & *f*)

Definition at line 45 of file Crystal.cpp.

6.13.3.4 virtual double Frame::Second (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented in Dodecahedron, Parallelepiped, TrianglePrism, Cup, CavityPrism, TaperedPrism, Bullet, Pyramid, and Prism.

Definition at line 90 of file Crystal.hpp.

6.13.3.5 virtual double Frame::Third (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented in Cup, CavityPrism, TaperedPrism, and Bullet.

Definition at line 92 of file Crystal.hpp.

6.13.4 Member Data Documentation

6.13.4.1 int** Frame::Gr [protected]

Numbers of the vertexes from which the facet consist of.

For example: GR[0][1]=5, means that the vertex number 1 of the facet number 0 is defined in p[5]

Definition at line 67 of file Crystal.hpp.

6.13.4.2 double** Frame::Gran [protected]

The coefficients of an equation of plane which contains the facet of the crystal.

For example: Gran[0]={A,B,C,D}, means that the facet 0 lies in a plane with equation Ax+By+Cz+D=0

Definition at line 76 of file Crystal.hpp.

6.13.4.3 unsigned int Frame::K [protected]

Total count of the facets.

Definition at line 68 of file Crystal.hpp.

6.13.4.4 unsigned int Frame::Km [protected]

Maximal count of vertexes at any facet.

Definition at line 68 of file Crystal.hpp.

6.13.4.5 unsigned int Frame::M [protected]

Total count of the crystal vertexes.

Definition at line 68 of file Crystal.hpp.

6.13.4.6 Point3D* Frame::p [protected]

Coordinates of vertexes of the crystal.

Definition at line 77 of file Crystal.hpp.

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

- Crystal.hpp
- Crystal.cpp

6.14 matrix Class Reference

The array with (n-rows x m-columns) dimensions of **real** values. Size of the array can't be changed.

`#include <matrix.hpp>`

Public Member Functions

- matrix (unsigned int _n, unsigned int _m)
  Creates and initializes new matrix.
- matrix (const matrix &)
- virtual ~matrix ()
- matrix operator= (const matrix &)
- matrix operator+ (const matrix &) const
- matrix operator+= (const matrix &m)
- matrix operator- (const matrix &) const
- matrix operator-= (const matrix &m)
- double * operator[] (unsigned int i) const
  The operator returns a pointer to the i-th row of the array.
- matrix operator* (double) const
- matrix operator* (const matrix &) const
- matrix operator*= (double)

- matrix operator/ (double) const
- matrix operator/= (double x)
- bool operator== (const matrix &) const
- bool operator!= (const matrix &m) const
- bool isSquare (void) const

  The function checks if the dimensions of the array are equal (n==m).
- void Fill (double x)

  The function sets all values of the array to x.
- void Identity (void)

  The function sets all the elements of the principal diagonal to one and all other elements to zero. The function do not check if the matrix is square.
- void Exchange (unsigned int i, unsigned int j)

  The function exchanges two rows.

Friends

- matrix operator∗ (double x, const matrix &m)
- double norm (const matrix &)

  The function returns a sum of squares of all elements.
- double Max (const matrix &)

  The function returns the square of maximal value of the array.
- unsigned int Str (const matrix &m)

  The function returns row counts.
- unsigned int Col (const matrix &m)

  The function returns column counts.
- std::ofstream & operator<< (std::ofstream &, const matrix &)

  The operator outputs all elements of the array to the file separated by space.

6.14.1 Detailed Description

The array with (n-rows x m-columns) dimensions of **real** values. Size of the array can't be changed.

Definition at line 24 of file matrix.hpp.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 matrix::matrix (unsigned int *_n,* unsigned int *_m*) [inline]

Creates and initializes new matrix.

Definition at line 32 of file matrix.hpp.

6.14.2.2 matrix::matrix (const matrix & *mt*)

Definition at line 25 of file matrix.cpp.

6.14.2.3 virtual matrix::∼matrix ( ) [inline], [virtual]

Definition at line 36 of file matrix.hpp.

6.14.3 Member Function Documentation

6.14.3.1 void matrix::Exchange (unsigned int *i,* unsigned int *j*) [inline]

The function exchanges two rows.

Definition at line 61 of file matrix.hpp.

6.14.3.2 void matrix::Fill (double *x*)

The function sets all values of the array to x.

Definition at line 116 of file matrix.cpp.

6.14.3.3 void matrix::Identity (void)

The function sets all the elements of the principal diagonal to one and all other elements to zero. The function do not check if the matrix is square.

Definition at line 122 of file matrix.cpp.

6.14.3.4 bool matrix::isSquare (void) const [inline]

The function checks if the dimensions of the array are equal (n==m).

Definition at line 54 of file matrix.hpp.

6.14.3.5 bool matrix::operator!= (const matrix & *m*) const [inline]

Definition at line 52 of file matrix.hpp.

6.14.3.6 matrix matrix::operator∗ (double *x*) const

Definition at line 66 of file matrix.cpp.

6.14.3.7 matrix matrix::operator∗ (const matrix & *mt*) const

Definition at line 75 of file matrix.cpp.

6.14.3.8 matrix matrix::operator∗= (double *x*)

Definition at line 89 of file matrix.cpp.

6.14.3.9 matrix matrix::operator+ (const matrix & *mt*) const

Definition at line 44 of file matrix.cpp.

6.14.3.10 matrix matrix::operator+= (const matrix & *m*) [inline]

Definition at line 40 of file matrix.hpp.

6.14.3.11 matrix matrix::operator- (const matrix & *mt*) const

Definition at line 55 of file matrix.cpp.

6.14.3.12 matrix matrix::operator-= (const matrix & *m*) [inline]

Definition at line 42 of file matrix.hpp.

6.14.3.13 matrix matrix::operator/ (double *x*) const

Definition at line 96 of file matrix.cpp.

6.14.3.14 matrix matrix::operator/= (double *x*) [inline]

Definition at line 50 of file matrix.hpp.

6.14.3.15 matrix matrix::operator= (const matrix & *mt*)

Definition at line 32 of file matrix.cpp.

6.14.3.16 bool matrix::operator== (const matrix & *mt*) const

Definition at line 107 of file matrix.cpp.

6.14.3.17 double* matrix::operator[] (unsigned int *i*) const [inline]

The operator returns a pointer to the i-th row of the array.

Definition at line 45 of file matrix.hpp.

6.14.4 Friends And Related Function Documentation

6.14.4.1 unsigned int Col (const matrix & *m*) [friend]

The function returns column counts.

Parameters

m	the array

Returns

row counts, (m.m)

Definition at line 83 of file matrix.hpp.

6.14.4.2 double Max (const matrix & *mt*) [friend]

The function returns the square of maximal value of the array.

Definition at line 137 of file matrix.cpp.

6.14.4.3 double norm (const matrix & *mt*) [friend]

The function returns a sum of squares of all elements.

Definition at line 129 of file matrix.cpp.

6.14.4.4 matrix operator∗ (double *x*, const matrix & *m*) [friend]

Definition at line 66 of file matrix.hpp.

6.14.4.5 std::ofstream& operator<< (std::ofstream & *out*, const matrix & *m*) [friend]

The operator outputs all elements of the array to the file separated by space.

Definition at line 149 of file matrix.cpp.

6.14.4.6 unsigned int Str (const matrix & *m*) [friend]

The function returns row counts.

Parameters

m	the array

Returns

row counts, (m.n)

Definition at line 77 of file matrix.hpp.

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

- matrix.hpp
- matrix.cpp

6.15 matrixC Class Reference

The array with (n-rows x m-columns) dimensions of **complex** values. Size of the array can't be changed.

`#include <matrix.hpp>`

Public Member Functions

- matrixC (unsigned int _n, unsigned int _m)
  Creates and initializes new matrix.
- matrixC (const matrixC &)
- virtual ∼matrixC ()
- matrixC operator= (const matrixC &)
- matrixC operator+ (const matrixC &) const
- matrixC operator+= (const matrixC &m)
- matrixC operator- (const matrixC &) const
- matrixC operator-= (const matrixC &m)
- complex ∗ operator[] (unsigned int N) const
  The operator returns a pointer to the i-th row of the array.
- matrixC operator∗ (const complex &) const

- matrixC operator∗ (const matrix &) const
- matrixC operator∗ (const matrixC &) const
- matrixC operator∗= (const complex &z)
- matrixC operator/ (const complex &) const
- matrixC operator/= (const complex &)
- bool operator== (const matrixC &) const
- bool operator!= (const matrixC &m) const
- void Fill (const complex &z)

  The function sets all values of the array to z.
- void Identity (void)

  The function sets all the elements of the principal diagonal to one and all other elements to zero. The function do not check if the matrix is square.
- void Exchange (unsigned int i, unsigned int j)

  The function exchanges two rows.

Friends

- double norm (const matrixC &)

  The function returns a sum of squares of all elements.
- double Max (const matrixC &)

  The function returns the square of maximal value of the array.
- unsigned int Str (const matrixC &m)

  The function returns row counts.
- unsigned int Col (const matrixC &m)

  The function returns column counts.
- matrixC operator∗ (const complex &z, const matrixC &m)
- std::ofstream & operator<< (std::ofstream &, const matrixC &)

  The operator outputs all elements of the array to the file separated by space. Real and Imagen parts are separated by comma.

6.15.1 Detailed Description

The array with (n-rows x m-columns) dimensions of **complex** values. Size of the array can't be changed.

Definition at line 96 of file matrix.hpp.

6.15.2 Constructor & Destructor Documentation

6.15.2.1 matrixC::matrixC (unsigned int *_n,* unsigned int *_m*) [inline]

Creates and initializes new matrix.

Definition at line 104 of file matrix.hpp.

6.15.2.2 matrixC::matrixC (const matrixC & *mt*)

Definition at line 183 of file matrix.cpp.

6.15.2.3 virtual matrixC::∼matrixC ( ) [inline], [virtual]

Definition at line 108 of file matrix.hpp.

6.15.3 Member Function Documentation

6.15.3.1 void matrixC::Exchange (unsigned int *i,* unsigned int *j*) [inline]

The function exchanges two rows.

Definition at line 132 of file matrix.hpp.

6.15.3.2 void matrixC::Fill (const complex & *z*)

The function sets all values of the array to z.

Definition at line 223 of file matrix.cpp.

6.15.3.3 void matrixC::Identity (void)

The function sets all the elements of the principal diagonal to one and all other elements to zero. The function do not check if the matrix is square.

Definition at line 229 of file matrix.cpp.

6.15.3.4 bool matrixC::operator!= (const matrixC & *m*) const [inline]

Definition at line 125 of file matrix.hpp.

6.15.3.5 matrixC matrixC::operator∗ (const complex & *z*) const

Definition at line 235 of file matrix.cpp.

6.15.3.6 matrixC matrixC::operator∗ (const matrix &) const

6.15.3.7 matrixC matrixC::operator∗ (const matrixC & *mt*) const

Definition at line 244 of file matrix.cpp.

6.15.3.8 matrixC matrixC::operator∗= (const complex & *z*) [inline]

Definition at line 121 of file matrix.hpp.

6.15.3.9 matrixC matrixC::operator+ (const matrixC & *mt*) const

Definition at line 201 of file matrix.cpp.

6.15.3.10 matrixC matrixC::operator+= (const matrixC & *m*) [inline]

Definition at line 112 of file matrix.hpp.

6.15.3.11 matrixC matrixC::operator- (const matrixC & *mt*) const

Definition at line 212 of file matrix.cpp.

6.15.3.12 matrixC matrixC::operator-= (const matrixC & *m*) [inline]

Definition at line 114 of file matrix.hpp.

6.15.3.13 matrixC matrixC::operator/ (const complex & *z*) const

Definition at line 259 of file matrix.cpp.

6.15.3.14 matrixC matrixC::operator/= (const complex & *z*)

Definition at line 270 of file matrix.cpp.

6.15.3.15 matrixC matrixC::operator= (const matrixC & *mt*)

Definition at line 190 of file matrix.cpp.

6.15.3.16 bool matrixC::operator== (const matrixC & *mt*) const

Definition at line 277 of file matrix.cpp.

6.15.3.17 complex* matrixC::operator[] (unsigned int *N*) const [inline]

The operator returns a pointer to the i-th row of the array.

Definition at line 117 of file matrix.hpp.

6.15.4 Friends And Related Function Documentation

6.15.4.1 unsigned int Col (const matrixC & *m*) [friend]

The function returns column counts.

Parameters

m	the array

Returns

row counts, (m.m)

Definition at line 151 of file matrix.hpp.

6.15.4.2 double Max (const matrixC & *mt*) [friend]

The function returns the square of maximal value of the array.

Definition at line 295 of file matrix.cpp.

6.15.4.3 double norm (const matrixC & *mt*) [friend]

The function returns a sum of squares of all elements.

Definition at line 287 of file matrix.cpp.

6.15.4.4 matrixC operator∗ (const complex & *z*, const matrixC & *m*) [friend]

Definition at line 152 of file matrix.hpp.

6.15.4.5 std::ofstream& operator<< (std::ofstream & *out*, const matrixC & *m*) [friend]

The operator outputs all elements of the array to the file separated by space. Real and Imagen parts are separated by comma.

Definition at line 307 of file matrix.cpp.

6.15.4.6 unsigned int Str (const matrixC & *m*) [friend]

The function returns row counts.

Parameters

m	the array

Returns

row counts, (m.n)

Definition at line 145 of file matrix.hpp.

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

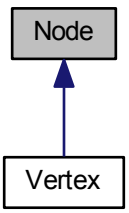
- matrix.hpp
- matrix.cpp

6.16 Node Class Reference

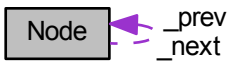
A pointer-based implementation of a linked list.

`#include <Geometry.hpp>`

Inheritance diagram for Node:



Collaboration diagram for Node:



Public Member Functions

- Node (void)
- virtual ~Node (void)
- Node * next (void) const
- Node * prev (void) const
- Node * insert (Node *)
- Node * remove (void)
- void splice (Node *)

Protected Attributes

- Node * _next
- Node * _prev

6.16.1 Detailed Description

A pointer-based implementation of a linked list.

See M.Laszlo "Computational Geometry and Computer Graphics in C++", page 25.

Definition at line 225 of file Geometry.hpp.

6.16.2 Constructor & Destructor Documentation

6.16.2.1 Node::Node (void) [inline]

Definition at line 230 of file Geometry.hpp.

6.16.2.2 virtual Node::~Node (void) [inline], [virtual]

Definition at line 231 of file Geometry.hpp.

6.16.3 Member Function Documentation

6.16.3.1 Node * Node::insert (Node * *b*)

Definition at line 180 of file Geometry.cpp.

6.16.3.2 Node* Node::next (void) const [inline]

Definition at line 232 of file Geometry.hpp.

6.16.3.3 Node* Node::prev (void) const [inline]

Definition at line 233 of file Geometry.hpp.

6.16.3.4 Node * Node::remove (void)

Definition at line 189 of file Geometry.cpp.

6.16.3.5 void Node::splice (Node * *b*)

Definition at line 196 of file Geometry.cpp.

6.16.4 Member Data Documentation

6.16.4.1 Node* Node::_next [protected]

Definition at line 227 of file Geometry.hpp.

6.16.4.2 Node * Node::_prev [protected]

Definition at line 227 of file Geometry.hpp.

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

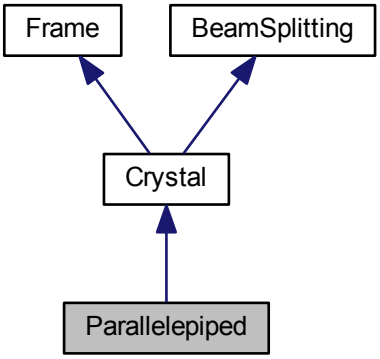
- Geometry.hpp
- Geometry.cpp

6.17 Parallelepiped Class Reference

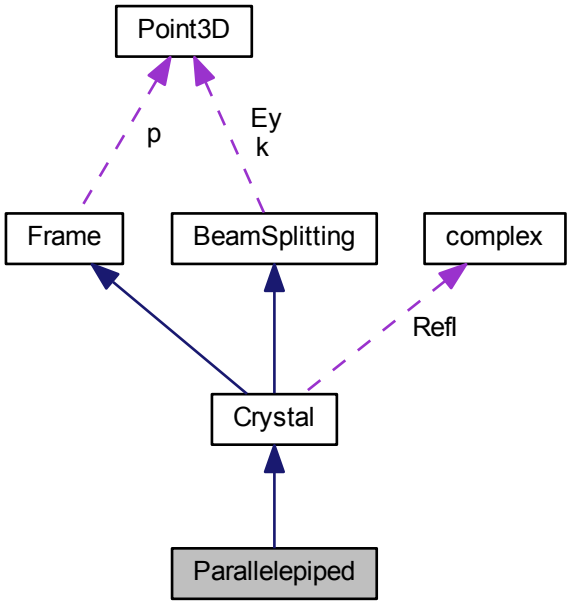
This class defines the parallelepiped.

`#include <particle.hpp>`

Inheritance diagram for Parallelepiped:



Collaboration diagram for Parallelepiped:



Public Member Functions

- Parallelepiped (const complex &r, double W, double H, double WP, unsigned int Itr, Point3D _k, Point3D _Ey)
- virtual ~Parallelepiped (void)
- double First (void)

  Returns a parameter of the size of the crystal, depends on the realization.

- double Second (void)

  Returns a parameter of the size of the crystal, depends on the realization.
- double Forth (void)

  Returns a parameter of the size of the crystal, depends on the realization.
- void SetVertices (void)

  This function defines the coordinates of the vertexes.
- void SetFacets (void)

Additional Inherited Members

6.17.1 Detailed Description

This class defines the parallelepiped.

Definition at line 160 of file particle.hpp.

6.17.2 Constructor & Destructor Documentation

6.17.2.1 Parallelepiped::Parallelepiped (const complex & *r,* double *W,* double *H,* double *WP,* unsigned int *Itr,* Point3D *_k,* Point3D *_Ey*) [inline]

Definition at line 166 of file particle.hpp.

6.17.2.2 virtual Parallelepiped::~Parallelepiped (void) [inline], [virtual]

Definition at line 169 of file particle.hpp.

6.17.3 Member Function Documentation

6.17.3.1 double Parallelepiped::First (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented from Frame.

Definition at line 171 of file particle.hpp.

6.17.3.2 double Parallelepiped::Forth (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented from Frame.

Definition at line 173 of file particle.hpp.

6.17.3.3 double Parallelepiped::Second (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented from Frame.

Definition at line 172 of file particle.hpp.

6.17.3.4 void Parallelepiped::SetFacets (void)

Definition at line 279 of file particle.cpp.

6.17.3.5 void Parallelepiped::SetVertices (void) [virtual]

This function defines the coordinates of the vertexes.

Reimplemented from Crystal.

Definition at line 267 of file particle.cpp.

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

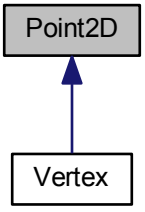
- particle.hpp
- particle.cpp

6.18 Point2D Class Reference

The class Point2D contains 2D point in Cartesian coordinates.

`#include <Geometry.hpp>`

Inheritance diagram for Point2D:



Public Member Functions

- Point2D (double _x=0.0, double _y=0.0)
- Point2D operator- (void) const
- Point2D operator- (const Point2D &a) const
- Point2D operator-= (const Point2D &a)
- Point2D operator+ (void) const
- Point2D operator+ (const Point2D &a) const
- Point2D operator+= (const Point2D &a)
- double operator∗ (const Point2D &a) const

  < *Dot Product of 2D vectors*
- double operator% (const Point2D &a) const

  < *Cross Product of 2D vectors*
- Point2D operator∗ (const double &a) const
- Point2D operator/ (const double &a) const
- Point2D operator/= (const double &a)

- bool operator== (const Point2D &a) const
- bool operator!= (const Point2D &a) const
- position classify (const Edge &) const
- position classify (const Point2D &, const Point2D &) const
- double distance (const Edge &) const

  signed distance
- double dist (const Edge &) const

  unsigned distance
- double norm (const Point2D &a)
- Point2D normalTo (const Point2D &a)

Public Attributes

- double x
- double y

Friends

- double length (const Point2D &a)
- double polarAngle (const Point2D &)

  returns polar angle (radians) of the 2D vector a
- Point2D operator* (const double &x, const Point2D &a)
- double signedAngle (const Point2D &, const Point2D &, const Point2D &)

  returns signed angle

6.18.1 Detailed Description

The class Point2D contains 2D point in Cartesian coordinates.

See M.Laszlo "Computational Geometry and Computer Graphics in C++", page 73.

Definition at line 57 of file Geometry.hpp.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 Point2D::Point2D (double *_x* = `0.0`, double *_y* = `0.0`) `[inline]`

Definition at line 61 of file Geometry.hpp.

6.18.3 Member Function Documentation

6.18.3.1 position Point2D::classify (const Edge & *e*) const

Definition at line 41 of file Geometry.cpp.

6.18.3.2 position Point2D::classify (const Point2D & *org*, const Point2D & *dst*) const

Position of the 2D point *this relatively the edge {org, dest}. Originally it was written by M.Laszlo (p.76) and was refined a bit by us for our beam-splitting code

Definition at line 9 of file Geometry.cpp.

6.18.3.3 double Point2D::dist (const Edge & *e*) const

unsigned distance

Definition at line 46 of file Geometry.cpp.

6.18.3.4 double Point2D::distance (const Edge & *e*) const

signed distance

distance between the edge e and the point with sign

Definition at line 29 of file Geometry.cpp.

6.18.3.5 double Point2D::norm (const Point2D & *a*) [inline]

Definition at line 94 of file Geometry.hpp.

6.18.3.6 Point2D Point2D::normalTo (const Point2D & *a*) [inline]

Definition at line 99 of file Geometry.hpp.

6.18.3.7 bool Point2D::operator!= (const Point2D & *a*) const [inline]

Definition at line 87 of file Geometry.hpp.

6.18.3.8 double Point2D::operator% (const Point2D & *a*) const [inline]

< Cross Product of 2D vectors

Definition at line 77 of file Geometry.hpp.

6.18.3.9 double Point2D::operator* (const Point2D & *a*) const [inline]

< Dot Product of 2D vectors

Definition at line 75 of file Geometry.hpp.

6.18.3.10 Point2D Point2D::operator* (const double & *a*) const [inline]

Definition at line 79 of file Geometry.hpp.

6.18.3.11 Point2D Point2D::operator+ (void) const [inline]

Definition at line 69 of file Geometry.hpp.

6.18.3.12 Point2D Point2D::operator+ (const Point2D & *a*) const [inline]

Definition at line 71 of file Geometry.hpp.

6.18.3.13 Point2D Point2D::operator+= (const Point2D & *a*) [inline]

Definition at line 73 of file Geometry.hpp.

6.18.3.14 Point2D Point2D::operator- (void) const [inline]

Definition at line 63 of file Geometry.hpp.

6.18.3.15 Point2D Point2D::operator- (const Point2D & *a*) const [inline]

Definition at line 65 of file Geometry.hpp.

6.18.3.16 Point2D Point2D::operator-= (const Point2D & *a*) [inline]

Definition at line 67 of file Geometry.hpp.

6.18.3.17 Point2D Point2D::operator/ (const double & *a*) const [inline]

Definition at line 81 of file Geometry.hpp.

6.18.3.18 Point2D Point2D::operator/= (const double & *a*) [inline]

Definition at line 83 of file Geometry.hpp.

6.18.3.19 bool Point2D::operator== (const Point2D & *a*) const [inline]

Definition at line 85 of file Geometry.hpp.

6.18.4 Friends And Related Function Documentation

6.18.4.1 double length (const Point2D & *a*) [friend]

Definition at line 96 of file Geometry.hpp.

6.18.4.2 Point2D operator∗ (const double & *x,* const Point2D & *a*) [friend]

Definition at line 101 of file Geometry.hpp.

6.18.4.3 double polarAngle (const Point2D & *a*) [friend]

returns polar angle (radians) of the 2D vector a

Definition at line 54 of file Geometry.cpp.

6.18.4.4 double signedAngle (const Point2D & *a,* const Point2D & *org,* const Point2D & *dst*) [friend]

returns signed angle

Definition at line 68 of file Geometry.cpp.

6.18.5 Member Data Documentation

6.18.5.1 double Point2D::x

Definition at line 59 of file Geometry.hpp.

6.18.5.2 double Point2D::y

Definition at line 59 of file Geometry.hpp.

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

- Geometry.hpp
- Geometry.cpp

6.19 Point3D Class Reference

The class Point3D contains 3D point in Cartesian coordinates.

`#include <Geometry.hpp>`

Public Member Functions

- Point3D (double X=0, double Y=0, double Z=0)
- Point3D operator- () const
- Point3D operator- (const Point3D &a) const
- Point3D operator-= (const Point3D &a)
- Point3D operator+ (const Point3D &a) const
- Point3D operator+= (const Point3D &a)
- Point3D operator∗ (double a) const
- double operator∗ (const Point3D &a) const

  Dot product.
- Point3D operator/ (double a) const
- Point3D operator∗= (double a)
- Point3D operator/= (double a)
- Point3D operator% (const Point3D &a) const

  < Cross product
- Point3D rotate (double, const Point3D &) const

  *Rotation of the point around the 3D axis **a** through the angle t < 17.10.2002 >.*
- Point3D rotateAxisY (double a) const
- Point3D rotateAxisZ (double a) const
- Point3D RotateEuler (double, double, double)
- bool operator== (const Point3D &a) const
- bool operator!= (const Point3D &a) const

Public Attributes

- double x
- double y
- double z

Friends

- Point3D operator∗ (const Point3D &a, const matrix &m)
- Point3D operator∗ (double x, const Point3D &a)
- double norm (const Point3D &a)
- double length (const Point3D &a)
- Point2D Proj (const Point3D &a, int i)
- SphCrd GetSpherical (const Point3D &r)

6.19.1 Detailed Description

The class Point3D contains 3D point in Cartesian coordinates.

See M.Laszlo "Computational Geometry and Computer Graphics in C++", page 95.

Definition at line 160 of file Geometry.hpp.

6.19.2 Constructor & Destructor Documentation

6.19.2.1 Point3D::Point3D (double *X* = 0, double *Y* = 0, double *Z* = 0) [inline]

Definition at line 163 of file Geometry.hpp.

6.19.3 Member Function Documentation

6.19.3.1 bool Point3D::operator!= (const Point3D & *a*) const [inline]

Definition at line 192 of file Geometry.hpp.

6.19.3.2 Point3D Point3D::operator% (const Point3D & *a*) const [inline]

< Cross product

Definition at line 184 of file Geometry.hpp.

6.19.3.3 Point3D Point3D::operator* (double *a*) const [inline]

Definition at line 174 of file Geometry.hpp.

6.19.3.4 double Point3D::operator* (const Point3D & *a*) const [inline]

Dot product.

Definition at line 176 of file Geometry.hpp.

6.19.3.5 Point3D Point3D::operator*= (double *a*) [inline]

Definition at line 180 of file Geometry.hpp.

6.19.3.6 Point3D Point3D::operator+ (const Point3D & *a*) const [inline]

Definition at line 170 of file Geometry.hpp.

6.19.3.7 Point3D Point3D::operator+= (const Point3D & *a*) [inline]

Definition at line 172 of file Geometry.hpp.

6.19.3.8 Point3D Point3D::operator- (void) const [inline]

Definition at line 164 of file Geometry.hpp.

6.19.3.9 Point3D Point3D::operator- (const Point3D & *a*) const [inline]

Definition at line 166 of file Geometry.hpp.

6.19.3.10 Point3D Point3D::operator-= (const Point3D & *a*) [inline]

Definition at line 168 of file Geometry.hpp.

6.19.3.11 Point3D Point3D::operator/ (double *a*) const [inline]

Definition at line 178 of file Geometry.hpp.

6.19.3.12 Point3D Point3D::operator/= (double *a*) [inline]

Definition at line 182 of file Geometry.hpp.

6.19.3.13 bool Point3D::operator== (const Point3D & *a*) const [inline]

Definition at line 190 of file Geometry.hpp.

6.19.3.14 Point3D Point3D::rotate (double *t,* const Point3D & *a*) const

Rotation of the point around the 3D axis **a** through the angle t $<$ 17.10.2002 $>$.
see G.A.Korn, T.M.Korn, MATHEMATICAL HANDBOOK FOR SCIENTISTS AND ENGINEERS

- 5th edition, 1984, p. 448 (Russian)

Definition at line 88 of file Geometry.cpp.

6.19.3.15 Point3D Point3D::rotateAxisY (double *a*) const

Definition at line 113 of file Geometry.cpp.

6.19.3.16 Point3D Point3D::rotateAxisZ (double *a*) const

Definition at line 120 of file Geometry.cpp.

6.19.3.17 Point3D Point3D::RotateEuler (double *tt,* double *ps,* double *f*)

Definition at line 127 of file Geometry.cpp.

6.19.4 Friends And Related Function Documentation

6.19.4.1 SphCrd GetSpherical (const Point3D & *r*) [friend]

Definition at line 890 of file Geometry.cpp.

6.19.4.2 double length (const Point3D & *a*) [friend]

Definition at line 198 of file Geometry.hpp.

6.19.4.3 double norm (const Point3D & *a*) [friend]

Parameters

a	Returns square of length

Definition at line 196 of file Geometry.hpp.

6.19.4.4 Point3D operator∗ (const Point3D & *a*, const matrix & *m*) [friend]

Definition at line 880 of file Geometry.cpp.

6.19.4.5 Point3D operator∗ (double *x*, const Point3D & *a*) [friend]

Definition at line 194 of file Geometry.hpp.

6.19.4.6 Point2D Proj (const Point3D & *a*, int *i*) [friend]

Definition at line 200 of file Geometry.hpp.

6.19.5 Member Data Documentation

6.19.5.1 double Point3D::x

Definition at line 162 of file Geometry.hpp.

6.19.5.2 double Point3D::y

Definition at line 162 of file Geometry.hpp.

6.19.5.3 double Point3D::z

Definition at line 162 of file Geometry.hpp.

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

- Geometry.hpp
- Geometry.cpp

6.20 Polyg Class Reference

The Polygon class represents 2D polygon.

`#include <Geometry.hpp>`

Public Member Functions

- Polyg (void)
- Polyg (const Polyg &p)
- Polyg (Vertex ∗v)
- ∼Polyg (void)
- Polyg & operator= (const Polyg &p)

- Vertex * setV (Vertex *v) const
- Vertex * v (void) const
- unsigned int size (void) const
- Vertex * cw (void) const
- Vertex * ccw (void) const
- Point2D point (void) const
- Edge edge (void) const
- Vertex * neighbor (rotation r) const
- Vertex * advance (rotation r) const
- Vertex * insert (const Point2D &p)
- void remove (void)
- Polyg * split (Vertex *b)
- bool Test (void)
- double PolygonIntersection (const Polyg &, const Polyg &)

Friends

- void advance (const Polyg &A, Polyg &R, bool inside)

  moving to the next edge (auxiliary routine for convexPolygonIntersect)
- bool convexPolygonIntersect (const Polyg &, const Polyg &, Polyg &)
- bool FastPolygonIntersect (const Polyg &, const Polyg &, Polyg &)
- double AreaOfConvexPolygon (const Polyg &)
- void OutPolyg (char *, char *, const Polyg &)
- bool CheckPolygon (const Polyg &)
- bool pointInConvexPolygon (const Point2D &, const Polyg &)

  *Checks if the 2D point **s** belongs to the 2D convex polygon **p**.*
- bool clipPolygon (const Polyg &, const Polyg &, Polyg &)
- bool clipPolygonToEdge (const Polyg &, const Edge &, Polyg &)

6.20.1 Detailed Description

The Polygon class represents 2D polygon.

See M.Laszlo "Computational Geometry and Computer Graphics in C++", page 82.

Definition at line 274 of file Geometry.hpp.

6.20.2 Constructor & Destructor Documentation

6.20.2.1 Polyg::Polyg (void) [inline]

Definition at line 282 of file Geometry.hpp.

6.20.2.2 Polyg::Polyg (const Polyg & *p*) [inline]

Definition at line 283 of file Geometry.hpp.

6.20.2.3 Polyg::Polyg (Vertex * *v*) [inline]

Definition at line 284 of file Geometry.hpp.

6.20.2.4 Polyg::∼Polyg (void)

Definition at line 241 of file Geometry.cpp.

6.20.3 Member Function Documentation

6.20.3.1 Vertex* Polyg::advance (rotation *r*) const [inline]

Definition at line 299 of file Geometry.hpp.

6.20.3.2 Vertex* Polyg::ccw (void) const [inline]

Definition at line 293 of file Geometry.hpp.

6.20.3.3 Vertex* Polyg::cw (void) const [inline]

Definition at line 292 of file Geometry.hpp.

6.20.3.4 Edge Polyg::edge (void) const [inline]

Definition at line 295 of file Geometry.hpp.

6.20.3.5 Vertex* Polyg::insert (const Point2D & *p*) [inline]

Definition at line 301 of file Geometry.hpp.

6.20.3.6 Vertex* Polyg::neighbor (rotation *r*) const [inline]

Definition at line 297 of file Geometry.hpp.

6.20.3.7 Polyg& Polyg::operator= (const Polyg & *p*) [inline]

Definition at line 287 of file Geometry.hpp.

6.20.3.8 Point2D Polyg::point (void) const [inline]

Definition at line 294 of file Geometry.hpp.

6.20.3.9 double Polyg::PolygonIntersection (const Polyg & *A*, const Polyg & *B*)

Definition at line 304 of file Geometry.cpp.

6.20.3.10 void Polyg::remove (void) [inline]

Definition at line 304 of file Geometry.hpp.

6.20.3.11 Vertex* Polyg::setV (Vertex * *v*) const [inline]

Definition at line 289 of file Geometry.hpp.

6.20.3.12 unsigned int Polyg::size (void) const [inline]

Definition at line 291 of file Geometry.hpp.

6.20.3.13 Polyg* Polyg::split (Vertex * *b*) [inline]

Definition at line 307 of file Geometry.hpp.

6.20.3.14 bool Polyg::Test (void)

Definition at line 253 of file Geometry.cpp.

6.20.3.15 Vertex* Polyg::v (void) const [inline]

Definition at line 290 of file Geometry.hpp.

6.20.4 Friends And Related Function Documentation

6.20.4.1 void advance (const Polyg & *A,* Polyg & *R,* bool *inside*) [friend]

moving to the next edge (auxiliary routine for convexPolygonIntersect)

Definition at line 428 of file Geometry.cpp.

6.20.4.2 double AreaOfConvexPolygon (const Polyg & *p*) [friend]

Definition at line 265 of file Geometry.cpp.

6.20.4.3 bool CheckPolygon (const Polyg & *p*) [friend]

Definition at line 293 of file Geometry.cpp.

6.20.4.4 bool clipPolygon (const Polyg & *s,* const Polyg & *p,* Polyg & *result*) [friend]

Definition at line 869 of file Geometry.cpp.

6.20.4.5 bool clipPolygonToEdge (const Polyg & *s,* const Edge & *e,* Polyg & *result*) [friend]

Definition at line 836 of file Geometry.cpp.

6.20.4.6 bool convexPolygonIntersect (const Polyg & *P,* const Polyg & *Q,* Polyg & *R*) [friend]

The routine of Laszlo was slightly refined for degenerated cases. It returns true if the intersection R of the input polygons P and Q is not empty.

Definition at line 438 of file Geometry.cpp.

6.20.4.7 bool FastPolygonIntersect (const Polyg & *P,* const Polyg & *Q,* Polyg & *R*) [friend]

It is just the same as previous "convexPolygonIntersect".

The routine was specially optimized for beam-splitting algorithm. It returns true if and only if the intersection R of the input convex polygons P and Q is not singular polyhedron.

Definition at line 592 of file Geometry.cpp.

6.20.4.8 void OutPolyg (char * *name,* char * *comment,* const Polyg & *p*) [friend]

Definition at line 282 of file Geometry.cpp.

6.20.4.9 bool pointInConvexPolygon (const Point2D & *s,* const Polyg & *p*) [friend]

Checks if the 2D point **s** belongs to the 2D convex polygon **p**.

Definition at line 350 of file Geometry.cpp.

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

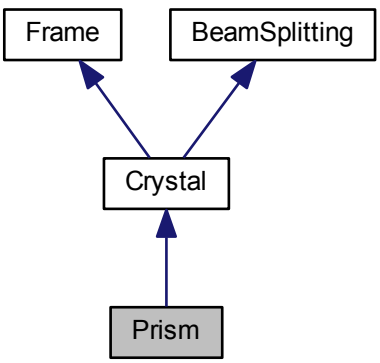
- Geometry.hpp
- Geometry.cpp

6.21 Prism Class Reference

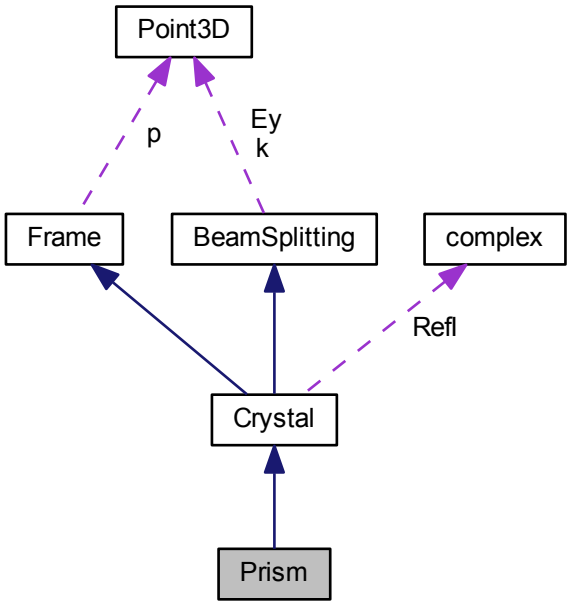
This class defines hexagonal prism.

`#include <particle.hpp>`

Inheritance diagram for Prism:



Collaboration diagram for Prism:



Public Member Functions

- Prism (const complex &r, double W, double H, unsigned int ltr, Point3D _k, Point3D _Ey)
- virtual ~Prism (void)
- double First (void)

  Returns a parameter of the size of the crystal, depends on the realization.
- double Second (void)

  Returns a parameter of the size of the crystal, depends on the realization.
- void SetVertices (void)

  This function defines the coordinates of the vertexes.
- void SetFacets (void)

Additional Inherited Members

6.21.1 Detailed Description

This class defines hexagonal prism.

The prism is defined by its radius and half height

Definition at line 22 of file particle.hpp.

6.21.2 Constructor & Destructor Documentation

6.21.2.1 Prism::Prism (const complex & *r,* double *W,* double *H,* unsigned int *ltr,* Point3D *_k,* Point3D *_Ey*) [inline]

Definition at line 29 of file particle.hpp.

6.21.2.2 virtual Prism::~Prism (void) [inline], [virtual]

Definition at line 32 of file particle.hpp.

6.21.3 Member Function Documentation

6.21.3.1 double Prism::First (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented from Frame.

Definition at line 34 of file particle.hpp.

6.21.3.2 double Prism::Second (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented from Frame.

Definition at line 35 of file particle.hpp.

6.21.3.3 void Prism::SetFacets (void)

Definition at line 29 of file particle.cpp.

6.21.3.4 void Prism::SetVertices (void) [virtual]

This function defines the coordinates of the vertexes.

Reimplemented from Crystal.

Definition at line 11 of file particle.cpp.

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

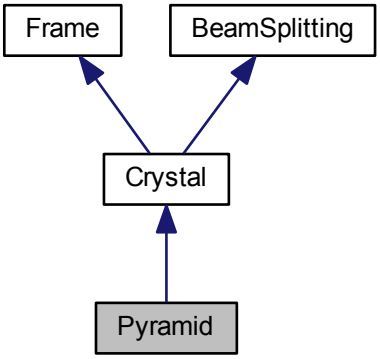
- particle.hpp
- particle.cpp

6.22 Pyramid Class Reference

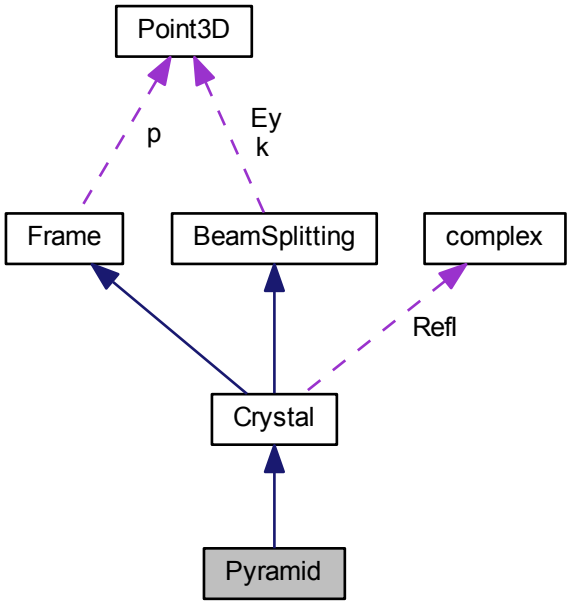
This class defines the pyramid.

`#include <particle.hpp>`

Inheritance diagram for Pyramid:



Collaboration diagram for Pyramid:



Public Member Functions

- Pyramid (const complex &r, double W, double H, unsigned int ltr, Point3D _k, Point3D _Ey)
- virtual ~Pyramid (void)
- double First (void)

  Returns a parameter of the size of the crystal, depends on the realization.

- double Second (void)

  Returns a parameter of the size of the crystal, depends on the realization.
- void SetVertices (void)

  This function defines the coordinates of the vertexes.
- void SetFacets (void)

Additional Inherited Members

6.22.1 Detailed Description

This class defines the pyramid.

Definition at line 41 of file particle.hpp.

6.22.2 Constructor & Destructor Documentation

6.22.2.1 Pyramid::Pyramid (const complex & *r,* double *W,* double *H,* unsigned int *ltr,* Point3D *_k,* Point3D *_Ey*) [inline]

Definition at line 48 of file particle.hpp.

6.22.2.2 virtual Pyramid::~Pyramid (void) [inline], [virtual]

Definition at line 51 of file particle.hpp.

6.22.3 Member Function Documentation

6.22.3.1 double Pyramid::First (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented from Frame.

Definition at line 53 of file particle.hpp.

6.22.3.2 double Pyramid::Second (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented from Frame.

Definition at line 54 of file particle.hpp.

6.22.3.3 void Pyramid::SetFacets (void)

Definition at line 58 of file particle.cpp.

6.22.3.4 void Pyramid::SetVertices (void) [virtual]

This function defines the coordinates of the vertexes.

Reimplemented from Crystal.

Definition at line 41 of file particle.cpp.

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

- particle.hpp
- particle.cpp

6.23 SphCrd Class Reference

Direction in 3D space (spherical coordinates on unit sphere).

`#include <Geometry.hpp>`

Public Member Functions

- SphCrd (double _fi=0, double _tt=0)

Public Attributes

- double fi
- double tetta

6.23.1 Detailed Description

Direction in 3D space (spherical coordinates on unit sphere).

Definition at line 212 of file Geometry.hpp.

6.23.2 Constructor & Destructor Documentation

6.23.2.1 SphCrd::SphCrd (double *_fi* = `0`, double *_tt* = `0`) `[inline]`

Definition at line 215 of file Geometry.hpp.

6.23.3 Member Data Documentation

6.23.3.1 double SphCrd::fi

Definition at line 214 of file Geometry.hpp.

6.23.3.2 double SphCrd::tetta

Definition at line 214 of file Geometry.hpp.

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

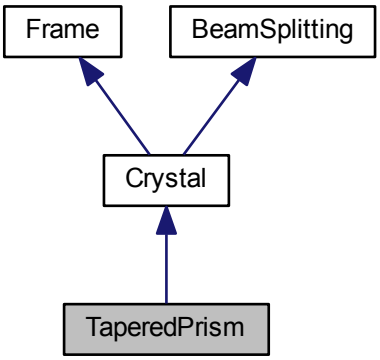
- Geometry.hpp

6.24 TaperedPrism Class Reference

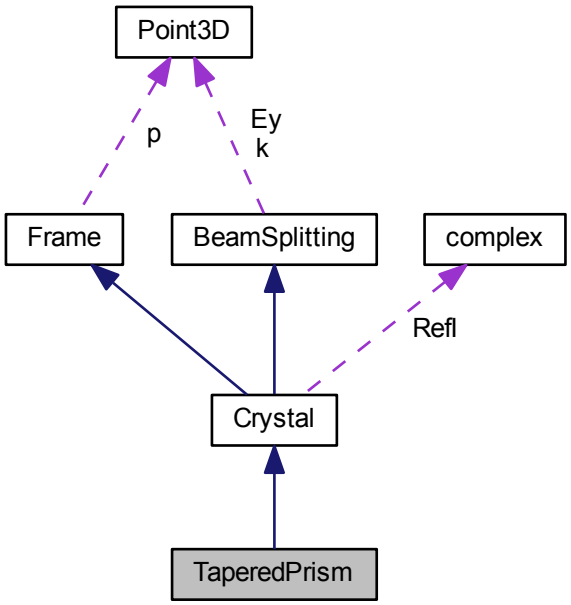
This class defines the hexagonal tapered prism.

`#include <particle.hpp>`

Inheritance diagram for TaperedPrism:



Collaboration diagram for TaperedPrism:



Public Member Functions

- TaperedPrism (const complex &r, double W, double H, double WP, double HP, unsigned int ltr, Point3D _k, Point3D _Ey)
- virtual ~TaperedPrism (void)
- double First (void)

Returns a parameter of the size of the crystal, depends on the realization.

- double Second (void)

  Returns a parameter of the size of the crystal, depends on the realization.
- double Third (void)

  Returns a parameter of the size of the crystal, depends on the realization.
- double Forth (void)

  Returns a parameter of the size of the crystal, depends on the realization.
- void SetVertices (void)

  This function defines the coordinates of the vertexes.
- void SetFacets (void)

Additional Inherited Members

6.24.1 Detailed Description

This class defines the hexagonal tapered prism.

Definition at line 79 of file particle.hpp.

6.24.2 Constructor & Destructor Documentation

6.24.2.1 TaperedPrism::TaperedPrism (const complex & *r,* double *W,* double *H,* double *WP,* double *HP,* unsigned int *Itr,* Point3D *_k,* Point3D *_Ey*) [inline]

Definition at line 87 of file particle.hpp.

6.24.2.2 virtual TaperedPrism::~TaperedPrism (void) [inline], [virtual]

Definition at line 90 of file particle.hpp.

6.24.3 Member Function Documentation

6.24.3.1 double TaperedPrism::First (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented from Frame.

Definition at line 92 of file particle.hpp.

6.24.3.2 double TaperedPrism::Forth (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented from Frame.

Definition at line 95 of file particle.hpp.

6.24.3.3 double TaperedPrism::Second (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented from Frame.

Definition at line 93 of file particle.hpp.

6.24.3.4 void TaperedPrism::SetFacets (void)

Definition at line 136 of file particle.cpp.

6.24.3.5 void TaperedPrism::SetVertices (void) [virtual]

This function defines the coordinates of the vertexes.

Reimplemented from Crystal.

Definition at line 102 of file particle.cpp.

6.24.3.6 double TaperedPrism::Third (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented from Frame.

Definition at line 94 of file particle.hpp.

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

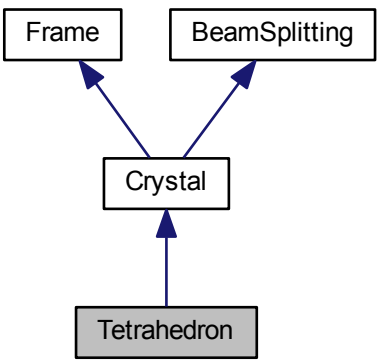
- particle.hpp
- particle.cpp

6.25 Tetrahedron Class Reference

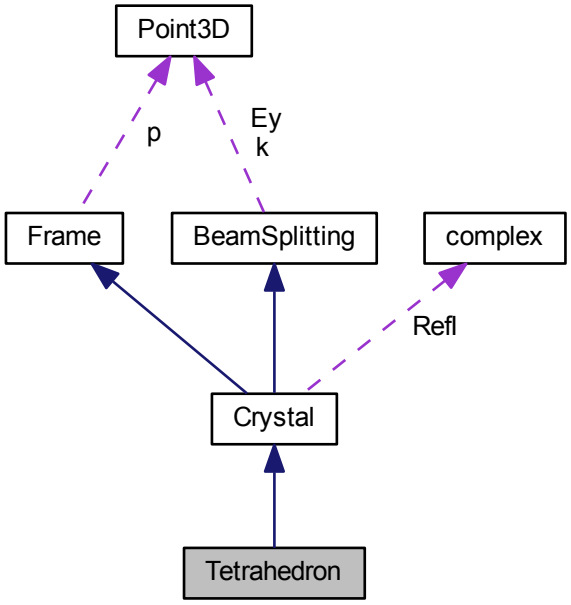
This class defines the tetrahedron.

`#include <particle.hpp>`

Inheritance diagram for Tetrahedron:



Collaboration diagram for Tetrahedron:



Public Member Functions

- Tetrahedron (const complex &r, double W, unsigned int Itr, Point3D _k, Point3D _Ey)
- virtual ~Tetrahedron (void)
- double First (void)

  Returns a parameter of the size of the crystal, depends on the realization.
- void SetVertices (void)

  This function defines the coordinates of the vertexes.
- void SetFacets (void)

Additional Inherited Members

6.25.1 Detailed Description

This class defines the tetrahedron.

Definition at line 179 of file particle.hpp.

6.25.2 Constructor & Destructor Documentation

6.25.2.1 Tetrahedron::Tetrahedron (const complex & *r,* double *W,* unsigned int *Itr,* Point3D *_k,* Point3D *_Ey*) [inline]

Definition at line 187 of file particle.hpp.

6.25.2.2 virtual Tetrahedron::~Tetrahedron (void) [inline], [virtual]

Definition at line 190 of file particle.hpp.

6.25.3 Member Function Documentation

6.25.3.1 double Tetrahedron::First (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented from Frame.

Definition at line 192 of file particle.hpp.

6.25.3.2 void Tetrahedron::SetFacets (void)

Definition at line 302 of file particle.cpp.

6.25.3.3 void Tetrahedron::SetVertices (void) [virtual]

This function defines the coordinates of the vertexes.

Reimplemented from Crystal.

Definition at line 291 of file particle.cpp.

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

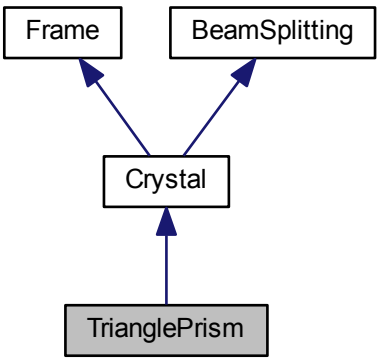
- particle.hpp
- particle.cpp

6.26 TrianglePrism Class Reference

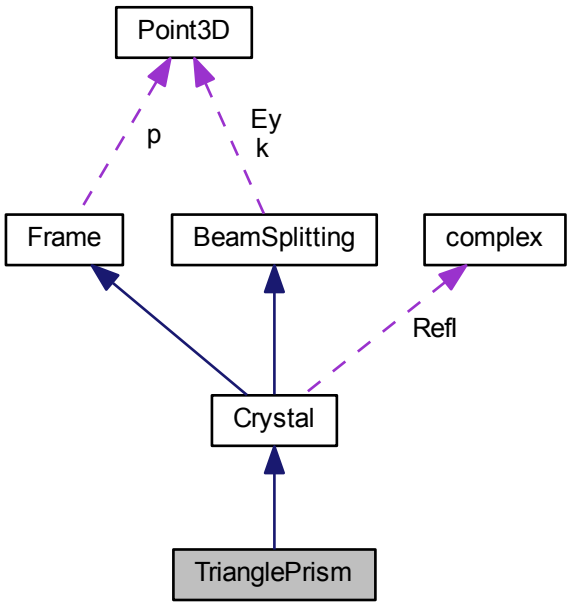
This class defines the triangle prism.

`#include <particle.hpp>`

Inheritance diagram for TrianglePrism:



Collaboration diagram for TrianglePrism:



Public Member Functions

- TrianglePrism (const complex &r, double W, double H, unsigned int ltr, Point3D _k, Point3D _Ey)
- virtual ~TrianglePrism (void)
- double First (void)

  Returns a parameter of the size of the crystal, depends on the realization.
- double Second (void)

  Returns a parameter of the size of the crystal, depends on the realization.
- void SetVertices (void)

  This function defines the coordinates of the vertexes.
- void SetFacets (void)

Additional Inherited Members

6.26.1 Detailed Description

This class defines the triangle prism.

Definition at line 141 of file particle.hpp.

6.26.2 Constructor & Destructor Documentation

6.26.2.1 TrianglePrism::TrianglePrism (const complex & *r,* double *W,* double *H,* unsigned int *ltr,* Point3D *_k,* Point3D *_Ey*) [inline]

Definition at line 148 of file particle.hpp.

6.26.2.2 virtual TrianglePrism::~TrianglePrism (void) [inline], [virtual]

Definition at line 151 of file particle.hpp.

6.26.3 Member Function Documentation

6.26.3.1 double TrianglePrism::First (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented from Frame.

Definition at line 153 of file particle.hpp.

6.26.3.2 double TrianglePrism::Second (void) [inline], [virtual]

Returns a parameter of the size of the crystal, depends on the realization.

Reimplemented from Frame.

Definition at line 154 of file particle.hpp.

6.26.3.3 void TrianglePrism::SetFacets (void)

Definition at line 255 of file particle.cpp.

6.26.3.4 void TrianglePrism::SetVertices (void) [virtual]

This function defines the coordinates of the vertexes.

Reimplemented from Crystal.

Definition at line 237 of file particle.cpp.

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

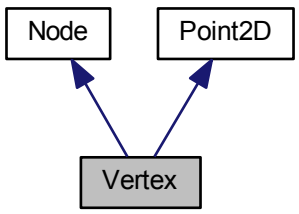
- particle.hpp
- particle.cpp

6.27 Vertex Class Reference

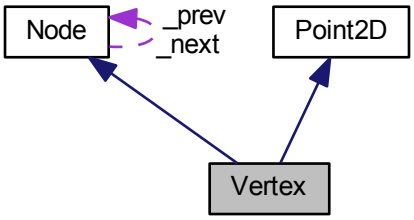
The class contains vertexes of 2D polygon.

`#include <Geometry.hpp>`

Inheritance diagram for Vertex:



Collaboration diagram for Vertex:



Public Member Functions

- Vertex (double x, double y)
- Vertex (const Point2D &p)
- virtual ~Vertex (void)
- Vertex * cw (void) const
- Vertex * ccw (void) const
- Point2D point (void) const
- Vertex * insert (Vertex *v)
- Vertex * remove (void)
- void splice (Vertex *b)
- Vertex * neighbor (rotation r) const
- Vertex * split (Vertex *)

Friends

- class Polyg

Additional Inherited Members

6.27.1 Detailed Description

The class contains vertexes of 2D polygon.

Vertex of the 2D polygon consisits of 2D point and two references to **next** and **previous** vertexes

See M.Laszlo "Computational Geometry and Computer Graphics in C++", page 80.

Definition at line 251 of file Geometry.hpp.

6.27.2 Constructor & Destructor Documentation

6.27.2.1 Vertex::Vertex (double *x*, double *y*) [inline]

Definition at line 253 of file Geometry.hpp.

6.27.2.2 Vertex::Vertex (const Point2D & *p*) [inline]

Definition at line 254 of file Geometry.hpp.

6.27.2.3 virtual Vertex::~Vertex (void) [inline], [virtual]

Definition at line 255 of file Geometry.hpp.

6.27.3 Member Function Documentation

6.27.3.1 Vertex* Vertex::ccw (void) const [inline]

Definition at line 257 of file Geometry.hpp.

6.27.3.2 Vertex* Vertex::cw (void) const [inline]

Definition at line 256 of file Geometry.hpp.

6.27.3.3 Vertex* Vertex::insert (Vertex * *v*) [inline]

Definition at line 259 of file Geometry.hpp.

6.27.3.4 Vertex* Vertex::neighbor (rotation *r*) const [inline]

Definition at line 262 of file Geometry.hpp.

6.27.3.5 Point2D Vertex::point (void) const [inline]

Definition at line 258 of file Geometry.hpp.

6.27.3.6 Vertex* Vertex::remove (void) [inline]

Definition at line 260 of file Geometry.hpp.

6.27.3.7 void Vertex::splice (Vertex * *b*) [inline]

Definition at line 261 of file Geometry.hpp.

6.27.3.8 Vertex * Vertex::split (Vertex * *b*)

Definition at line 208 of file Geometry.cpp.

6.27.4 Friends And Related Function Documentation

6.27.4.1 friend class Polyg [friend]

Definition at line 265 of file Geometry.hpp.

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

- Geometry.hpp
- Geometry.cpp

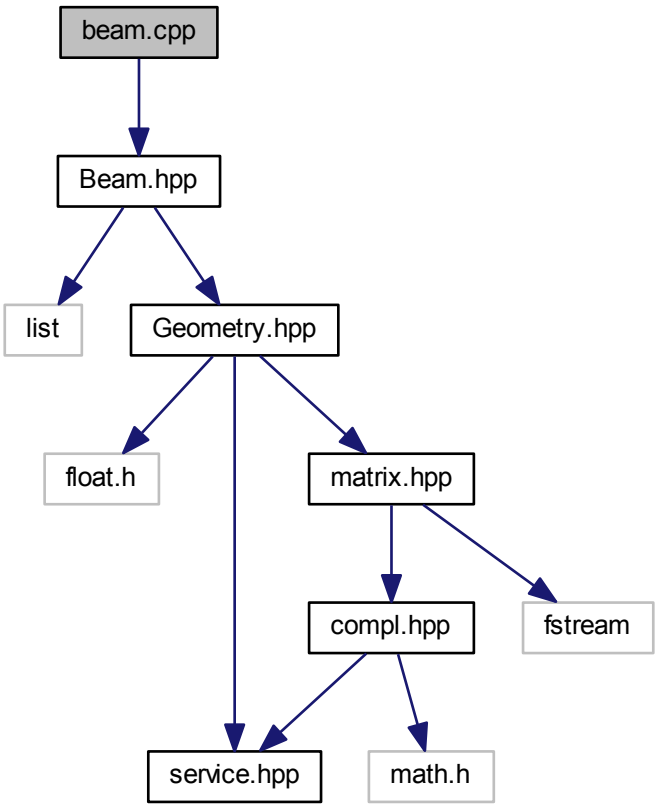
Chapter 7

File Documentation

7.1 beam.cpp File Reference

```
#include "Beam.hpp"
```

Include dependency graph for beam.cpp:



Functions

- void Rot (const Point3D &e, const Point3D &E, const Point3D &v, matrixC &m)
- Point3D CenterOfBeam (const Beam &bm)
- double CrossSection (const Beam &bm)
- double AreaOfBeam (const Beam &bm)
- void OutBeam (char *name, const Beam &bm)
- Point2D Proj_Vertex (const Beam &bm, Point3D &pt)

Variables

- const double Eps = 1e3*DBL_EPSILON

7.1.1 Function Documentation

7.1.1.1 double AreaOfBeam (const Beam & *bm*)

Definition at line 108 of file beam.cpp.

7.1.1.2 Point3D CenterOfBeam (const Beam & *bm*)

Definition at line 74 of file beam.cpp.

7.1.1.3 double CrossSection (const Beam & *bm*)

Definition at line 84 of file beam.cpp.

7.1.1.4 void OutBeam (char * *name*, const Beam & *bm*)

Definition at line 124 of file beam.cpp.

7.1.1.5 Point2D Proj_Vertex (const Beam & *bm*, Point3D & *pt*)

Definition at line 134 of file beam.cpp.

7.1.1.6 void Rot (const Point3D & *e*, const Point3D & *E*, const Point3D & *v*, matrixC & *m*)

Definition at line 12 of file beam.cpp.

7.1.2 Variable Documentation

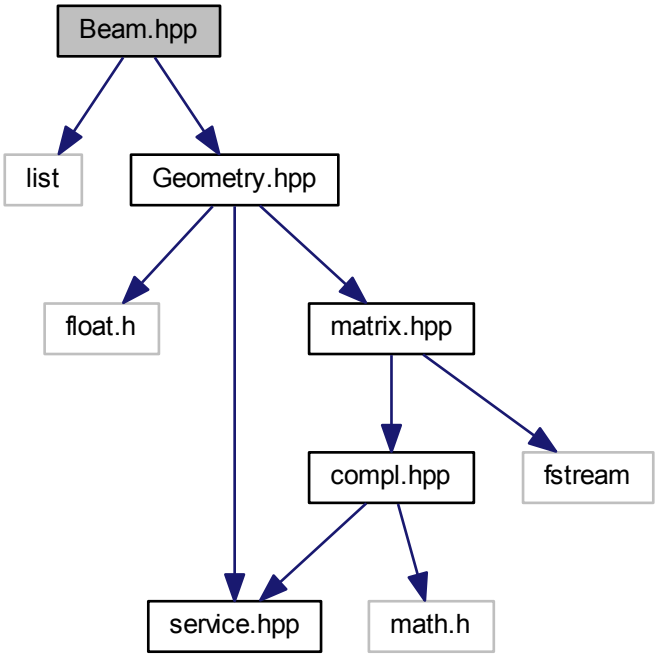
7.1.2.1 const double Eps = 1e3*DBL_EPSILON

Definition at line 3 of file beam.cpp.

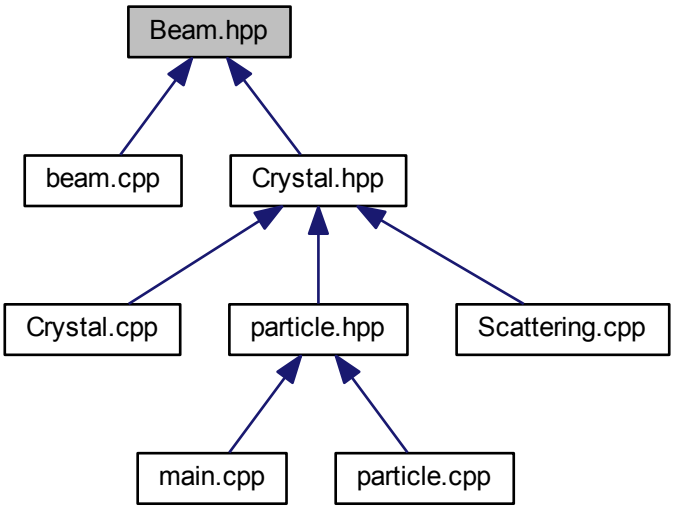
7.2 Beam.hpp File Reference

```
#include <list>
#include "Geometry.hpp"
```

Include dependency graph for Beam.hpp:



This graph shows which files directly or indirectly include this file:



Classes

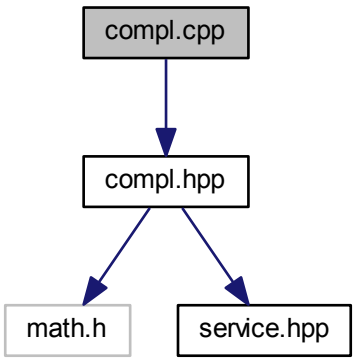
- class Beam

  The Beam class consist of 3D polygon, propagation direction and it's history, Jones matrix and so on.

7.3 compl.cpp File Reference

```
#include "compl.hpp"
```

Include dependency graph for compl.cpp:



Functions

- void SinCos (long double x, long double &sn, long double &cs)
- complex sqrt (const complex &z)
- complex exp_im (double x)

7.3.1 Function Documentation

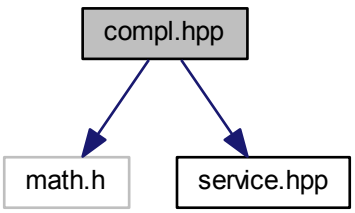
7.3.1.1 complex sqrt (const complex & *z*)

Definition at line 18 of file compl.cpp.

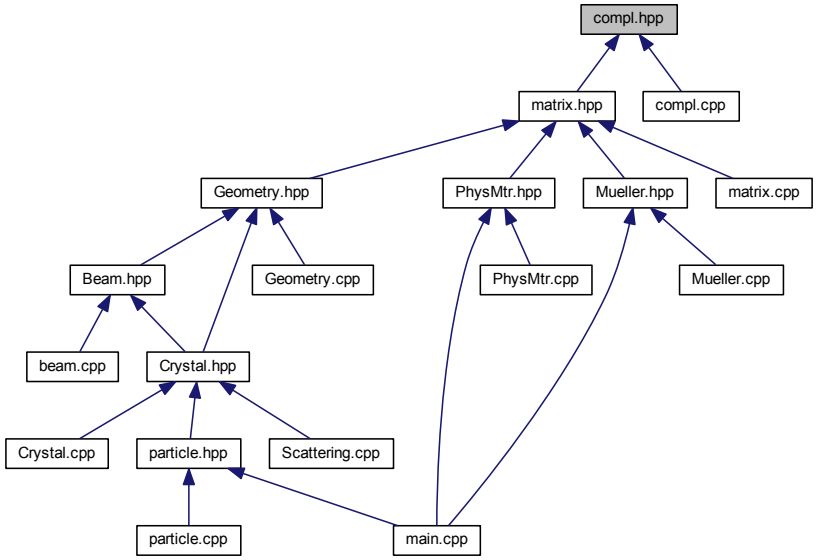
7.4 compl.hpp File Reference

```
#include <math.h>
#include "service.hpp"
```

Include dependency graph for compl.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class complex

  This class provides a complex numbers and operation with them.

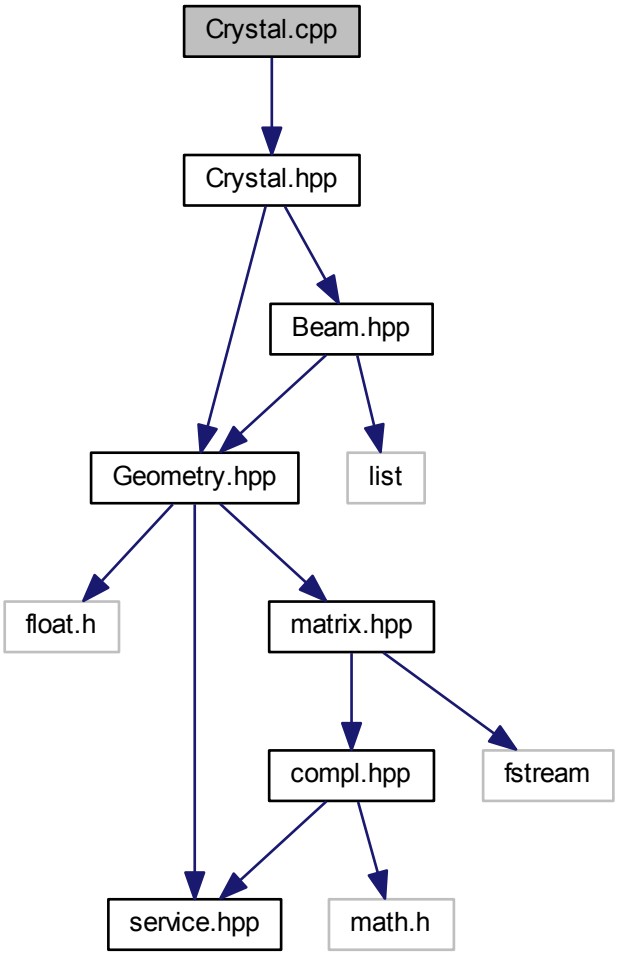
Functions

- complex exp (const complex &z)
- complex exp_im (double x)
- void SinCos (long double, long double &, long double &)

7.5 Crystal.cpp File Reference

```
#include "Crystal.hpp"
```

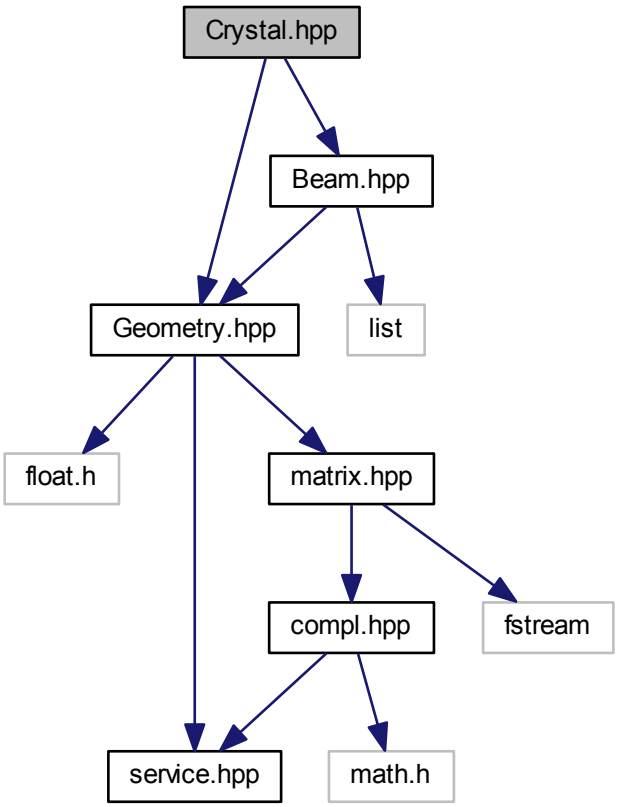
Include dependency graph for Crystal.cpp:



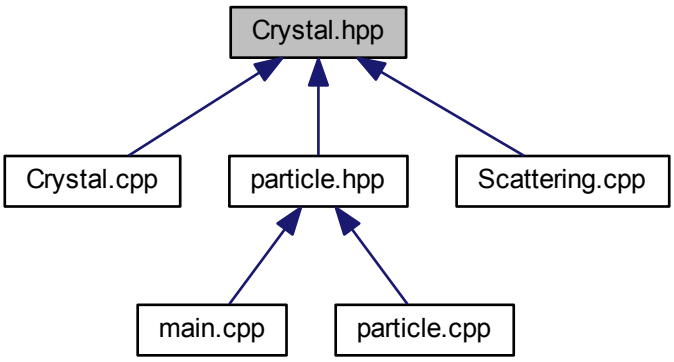
7.6 Crystal.hpp File Reference

```
#include "Geometry.hpp"
#include "Beam.hpp"
```

Include dependency graph for Crystal.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class BeamSplitting

  This class is a prototype for Crystal class. It contains parameters for BeamSplitting algorithm.
- class Frame

  This class is a prototype for Crystal class.
- class Crystal

  This class contains all information about particle's geometry and the parameters of tracing.

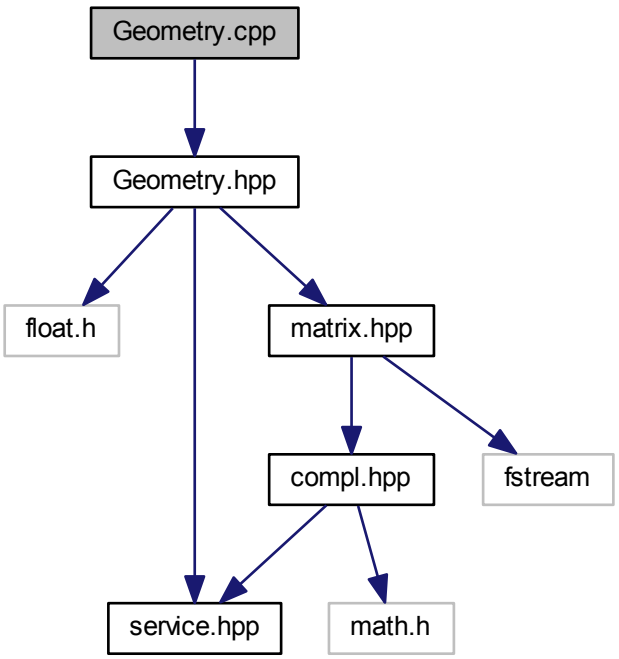
Typedefs

- typedef void(∗ Hand)(Beam &)

  Function prototype witch should be used for handling the outgoing beam in main program.

7.7 Geometry.cpp File Reference

```
#include "Geometry.hpp"
```
Include dependency graph for Geometry.cpp:



Functions

- double polarAngle (const Point2D &a)

  returns polar angle (radians) of the 2D vector a
- double signedAngle (const Point2D &a, const Point2D &org, const Point2D &dst)

returns signed angle

- direction parallel (const Edge &e, const Edge &f)
- double AreaOfConvexPolygon (const Polyg &p)
- void OutPolyg (char *name, char *comment, const Polyg &p)
- bool CheckPolygon (const Polyg &p)
- bool pointInConvexPolygon (const Point2D &s, const Polyg &p)

  *Checks if the 2D point **s** belongs to the 2D convex polygon **p**.*
- state crossingPoint (const Edge &e, const Edge &f, Point2D &p)
- bool aimsAt (const Edge &a, const Edge &b, position aclass, state crossType)

  *this function returns TRUE if and only if edge **a** aims at edge **b**. See p. 156*
- void advance (const Polyg &A, Polyg &R, bool inside)

  moving to the next edge (auxiliary routine for convexPolygonIntersect)
- bool convexPolygonIntersect (const Polyg &P, const Polyg &Q, Polyg &R)
- position base_clsfy (const Point2D &d, const Point2D &n, double eps, bool cmp)
- position clsfy (const Point2D &d, const Point2D &n, double eps, double ld, double ln)
- bool InclusionInsideTest (const Point2D &s, const Polyg &p)
- bool FastPolygonIntersect (const Polyg &P, const Polyg &Q, Polyg &R)
- bool clipPolygonToEdge (const Polyg &s, const Edge &e, Polyg &result)
- bool clipPolygon (const Polyg &s, const Polyg &p, Polyg &result)
- Point3D operator* (const Point3D &a, const matrix &mt)
- SphCrd GetSpherical (const Point3D &r)

7.7.1 Function Documentation

7.7.1.1 void advance (const Polyg & *A*, Polyg & *R*, bool *inside*)

moving to the next edge (auxiliary routine for convexPolygonIntersect)

Definition at line 428 of file Geometry.cpp.

7.7.1.2 bool aimsAt (const Edge & *a*, const Edge & *b*, position *aclass*, state *crossType*)

this function returns TRUE if and only if edge **a** aims at edge **b**. See p. 156

Definition at line 418 of file Geometry.cpp.

7.7.1.3 double AreaOfConvexPolygon (const Polyg & *p*)

Definition at line 265 of file Geometry.cpp.

7.7.1.4 position base_clsfy (const Point2D & *d*, const Point2D & *n*, double *eps*, bool *cmp*) [inline]

Definition at line 545 of file Geometry.cpp.

7.7.1.5 bool CheckPolygon (const Polyg & *p*)

Definition at line 293 of file Geometry.cpp.

7.7.1.6 bool clipPolygon (const Polyg & *s*, const Polyg & *p*, Polyg & *result*)

Definition at line 869 of file Geometry.cpp.

7.7.1.7 bool clipPolygonToEdge (const Polyg & *s,* const Edge & *e,* Polyg & *result*)

Definition at line 836 of file Geometry.cpp.

7.7.1.8 position clsfy (const Point2D & *d,* const Point2D & *n,* double *eps,* double *ld,* double *ln*)

Definition at line 551 of file Geometry.cpp.

7.7.1.9 bool convexPolygonIntersect (const Polyg & *P,* const Polyg & *Q,* Polyg & *R*)

The routine of Laszlo was slightly refined for degenerated cases. It returns true if the intersection R of the input polygons P and Q is not empty.

Definition at line 438 of file Geometry.cpp.

7.7.1.10 state crossingPoint (const Edge & *e,* const Edge & *f,* Point2D & *p*)

Definition at line 368 of file Geometry.cpp.

7.7.1.11 bool FastPolygonIntersect (const Polyg & *P,* const Polyg & *Q,* Polyg & *R*)

It is just the same as previous "convexPolygonIntersect".

The routine was specially optimized for beam-splitting algorithm. It returns true if and only if the intersection R of the input convex polygons P and Q is not singular polyhedron.

Definition at line 592 of file Geometry.cpp.

7.7.1.12 SphCrd GetSpherical (const Point3D & *r*)

Definition at line 890 of file Geometry.cpp.

7.7.1.13 bool InclusionInsideTest (const Point2D & *s,* const Polyg & *p*)

Checks if the 2D point **s** lies inside the 2D convex polygon **p** The routine originally was written by M. Laszlo and slightly refined for our beam-splitting code;

Definition at line 569 of file Geometry.cpp.

7.7.1.14 Point3D operator∗ (const Point3D & *a,* const matrix & *mt*)

Definition at line 880 of file Geometry.cpp.

7.7.1.15 void OutPolyg (char ∗ *name,* char ∗ *comment,* const Polyg & *p*)

Definition at line 282 of file Geometry.cpp.

7.7.1.16 direction parallel (const Edge & *e,* const Edge & *f*)

Definition at line 162 of file Geometry.cpp.

7.7.1.17 bool pointInConvexPolygon (const Point2D & *s*, const Polyg & *p*)

Checks if the 2D point **s** belongs to the 2D convex polygon **p**.

Definition at line 350 of file Geometry.cpp.

7.7.1.18 double polarAngle (const Point2D & *a*)

returns polar angle (radians) of the 2D vector a

Definition at line 54 of file Geometry.cpp.

7.7.1.19 double signedAngle (const Point2D & *a*, const Point2D & *org*, const Point2D & *dst*)

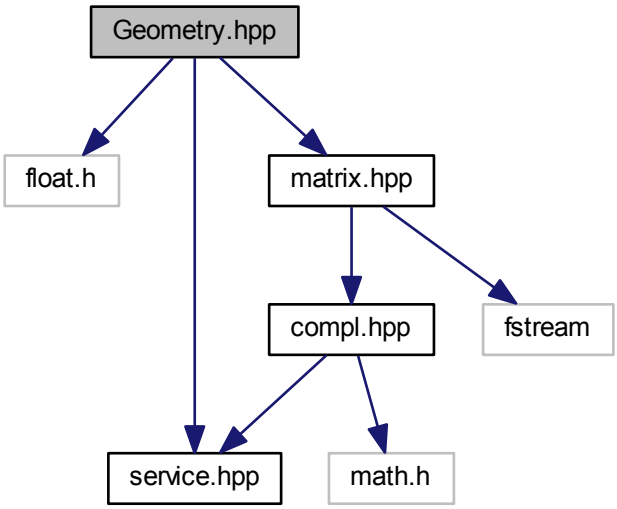
returns signed angle

Definition at line 68 of file Geometry.cpp.

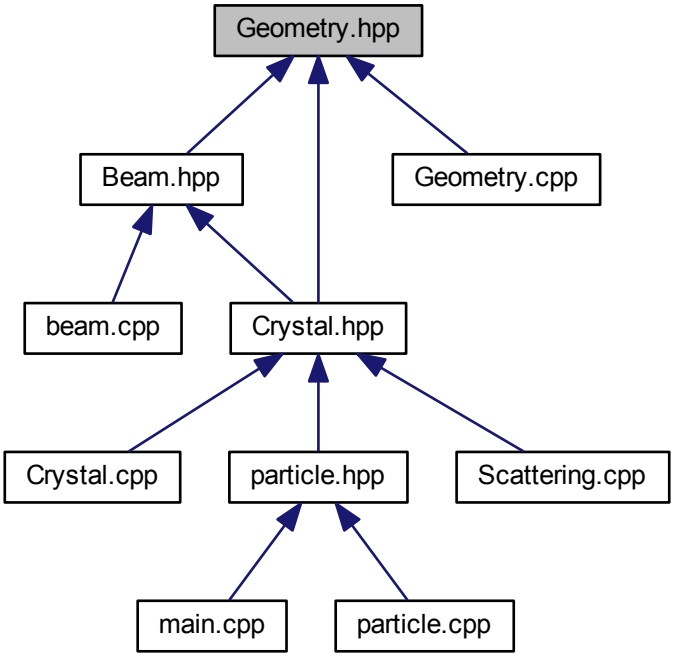
7.8 Geometry.hpp File Reference

```
#include <float.h>
#include "service.hpp"
#include "matrix.hpp"
```

Include dependency graph for Geometry.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Point2D

  The class Point2D contains 2D point in Cartesian coordinates.
- class Edge

  The Edge class represents all forms of lines in 2D.
- class Point3D

  The class Point3D contains 3D point in Cartesian coordinates.
- class SphCrd

  Direction in 3D space (spherical coordinates on unit sphere).
- class Node

  A pointer-based implementation of a linked list.
- class Vertex

  The class contains vertexes of 2D polygon.
- class Polyg

  The Polygon class represents 2D polygon.

Macros

- #define m_2pi 6.28318530717958647692528676655900

Enumerations

- enum direction { NON_PARALLEL, OPPOSITE, ACCORDANT, PERPENDICULAR }

  relative position of two edges
- enum rotation { CLOCKWISE, COUNTER_CLOCKWISE }

  direction of rotation (p.84)
- enum state {
  COLLINEAR, PARALLEL, SKEW, SKEW_CROSS,
  SKEW_NO_CROSS }

  relative state of two edges (p.93)
- enum position {
  LEFT, RIGHT, BEYOND, BEHIND,
  BETWEEN, ORIGIN, DESTINATION }

  position of point regarding to vector (p.76)
- enum crossing { UNKNOWN, P_IS_INSIDE, Q_IS_INSIDE }

  mutual position of two 2D polygons (p.158)

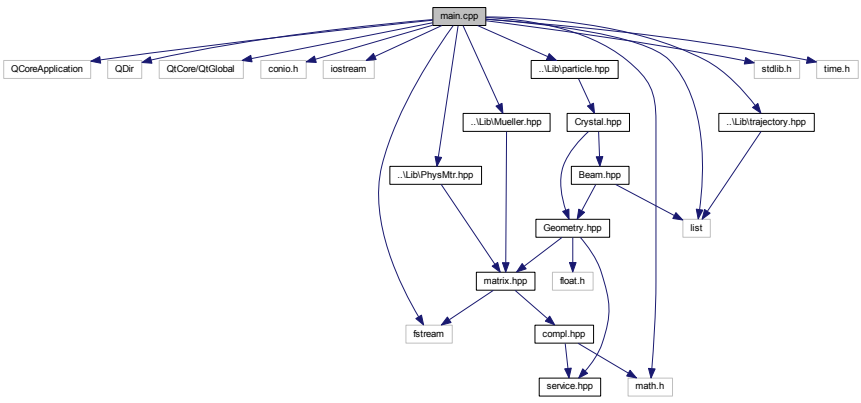
Variables

- const double E_DBL = 1e7∗DBL_EPSILON

7.9 main.cpp File Reference

```
#include <QCoreApplication>
#include <QDir>
#include <QtCore/QtGlobal>
#include <conio.h>
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <list>
#include <time.h>
#include <math.h>
#include "..\Lib\trajectory.hpp"
#include "..\Lib\PhysMtr.hpp"
#include "..\Lib\Mueller.hpp"
#include "..\Lib\particle.hpp"
```

Include dependency graph for main.cpp:



Functions

- void Handler (Beam &bm)

  < handler for the emitted beams
- int ReadFile (char *name, double *params, unsigned int n)

  Reads the parameters from data file.
- void MaskAppend (char s[], unsigned int n)

  *Fill in the **mask** and **Face** from data file.*
- void ShowTitle (void)

  Shows the title.
- void ShowCurrentTime (void)

  Shows current time.
- void DelFace (void)

  *Deletes **Face** structure.*
- int main (int argc, char *argv[])

  Main()

Variables

- Crystal * Body = NULL

  Crystal particle.
- unsigned int KoP

  Kind of particle.
- unsigned int AoP56

  Flag, 1 means angle of tip 56 deg, 0 means sizes of tip defined in data file.
- unsigned int GammaNumber

  Number of steps for Gamma rotation.
- unsigned int BettaNumber

  Number of steps for Betta rotation.
- unsigned int ThetaNumber

  Total number of bins in output file.
- unsigned int Itr

  Maximal number of internal reflection for every beam.
- unsigned int EDF

  EDF = 1 means to extract the delta-function in forward direction, EDF = 0 otherwise.
- complex _Refl (0, 0)

  Refraction index.
- unsigned int Sorting =0

  Sorting = 0 if all trajectories are taking into account, otherwise Sorting is equal to number of trajectories to calculate.
- unsigned int _NoF

  Number of facets of the crystal.
- unsigned int ** Face

  An array for masking the trajectories which are out of interest.
- double Radius

  Radius of the particle.
- double Halh_Height

  Half height of the particle.
- double TipRadius

  Radius of the tip (if AoP56 = 0)
- double TipHeight

  Height of the tip (if AoP56 = 0)

- double SizeBin

  The size of the bin for Theta angle (radians)
- double P = 0

  Probability distribution for Betta angle.
- Arr2D mxd (0, 0, 0, 0)

  An array of output Mueller matrixes.
- matrix back (4, 4)
- matrix forw (4, 4)

  < *Mueller matrix in backward direction*
- list< Chain > mask

  List of trajectories to take into account.
- Point3D k (0, 0, 1)
- Point3D Ey (0, 1, 0)

  < *Direction on incident wave*
- const int size = 256

7.9.1 Function Documentation

7.9.1.1 void DelFace (void)

Deletes **Face structure.

Definition at line 361 of file main.cpp.

7.9.1.2 void Handler (Beam & *bm*)

< handler for the emitted beams

Definition at line 262 of file main.cpp.

7.9.1.3 int main (int *argc*, char * *argv[]*)

Main()

< QT needs it

< Number of lines in data files, except trajectories

< array of input data

Definition at line 65 of file main.cpp.

7.9.1.4 void MaskAppend (char *s[]*, unsigned int *n*)

Fill in the **mask** and **Face** from data file.

Definition at line 337 of file main.cpp.

7.9.1.5 int ReadFile (char * *name*, double * *params*, unsigned int *n*)

Reads the parameters from data file.

Definition at line 291 of file main.cpp.

7.9.1.6 void ShowCurrentTime (void)

Shows current time.

Definition at line 379 of file main.cpp.

7.9.1.7 void ShowTitle (void)

Shows the title.

Definition at line 368 of file main.cpp.

7.9.2 Variable Documentation

7.9.2.1 unsigned int _NoF

Number of facets of the crystal.

Definition at line 32 of file main.cpp.

7.9.2.2 complex _Refl(0, 0)

Refraction index.

7.9.2.3 unsigned int AoP56

Flag, 1 means angle of tip 56 deg, 0 means sizes of tip defined in data file.

Definition at line 24 of file main.cpp.

7.9.2.4 matrix back(4, 4)

7.9.2.5 unsigned int BettaNumber

Number of steps for Betta rotation.

Definition at line 24 of file main.cpp.

7.9.2.6 Crystal* Body = NULL

Crystal particle.

Definition at line 23 of file main.cpp.

7.9.2.7 unsigned int EDF

EDF = 1 means to extract the delta-function in forward direction, EDF = 0 otherwise.

Definition at line 24 of file main.cpp.

7.9.2.8 Point3D Ey(0, 1, 0)

< Direction on incident wave

Basis for polarization characteristic of light

7.9.2.9 unsigned int ∗∗ Face

An array for masking the trajectories which are out of interest.

Definition at line 32 of file main.cpp.

7.9.2.10 matrix forw(4, 4)

< Mueller matrix in backward direction

Mueller matrix in forward direction

7.9.2.11 unsigned int GammaNumber

Number of steps for Gamma rotation.

Definition at line 24 of file main.cpp.

7.9.2.12 double Halh_Height

Half height of the particle.

Definition at line 35 of file main.cpp.

7.9.2.13 unsigned int ltr

Maximal number of internal reflection for every beam.

Definition at line 24 of file main.cpp.

7.9.2.14 Point3D k(0, 0, 1)

7.9.2.15 unsigned int KoP

Kind of particle.

Definition at line 24 of file main.cpp.

7.9.2.16 list<Chain> mask

List of trajectories to take into account.

Definition at line 44 of file main.cpp.

7.9.2.17 Arr2D mxd(0, 0, 0, 0)

An array of output Mueller matrixes.

7.9.2.18 double P = 0

Probability distribution for Betta angle.

Definition at line 40 of file main.cpp.

7.9.2.19 double Radius

Radius of the particle.

Definition at line 35 of file main.cpp.

7.9.2.20 const int size = 256

Definition at line 289 of file main.cpp.

7.9.2.21 double SizeBin

The size of the bin for Theta angle (radians)

Definition at line 35 of file main.cpp.

7.9.2.22 unsigned int Sorting =0

Sorting = 0 if all trajectories are taking into account, otherwise Sorting is equal to number of trajectories to calculate.

Definition at line 32 of file main.cpp.

7.9.2.23 unsigned int ThetaNumber

Total number of bins in output file.

Definition at line 24 of file main.cpp.

7.9.2.24 double TipHeight

Height of the tip (if AoP56 = 0)

Definition at line 35 of file main.cpp.

7.9.2.25 double TipRadius

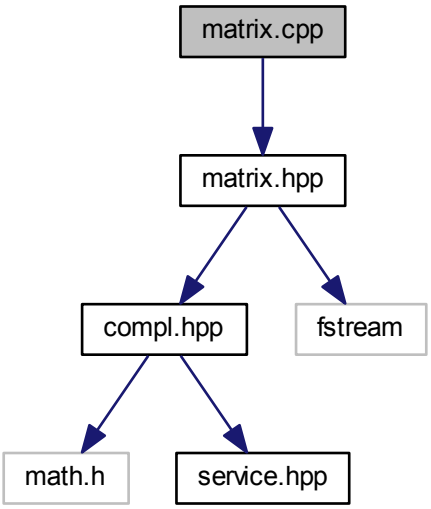
Radius of the tip (if AoP56 = 0)

Definition at line 35 of file main.cpp.

7.10 matrix.cpp File Reference

```
#include "matrix.hpp"
```

Include dependency graph for matrix.cpp:



Functions

- double norm (const matrix &mt)
- double Max (const matrix &mt)
- std::ofstream & operator<< (std::ofstream &out, const matrix &m)
- double norm (const matrixC &mt)
- double Max (const matrixC &mt)
- std::ofstream & operator<< (std::ofstream &out, const matrixC &m)

7.10.1 Function Documentation

7.10.1.1 double Max (const matrix & *mt*)

Definition at line 137 of file matrix.cpp.

7.10.1.2 double Max (const matrixC & *mt*)

Definition at line 295 of file matrix.cpp.

7.10.1.3 double norm (const matrix & *mt*)

Definition at line 129 of file matrix.cpp.

7.10.1.4 double norm (const matrixC & *mt*)

Definition at line 287 of file matrix.cpp.

7.10.1.5 std::ofstream& operator<< (std::ofstream & *out*, const matrix & *m*)

Definition at line 149 of file matrix.cpp.

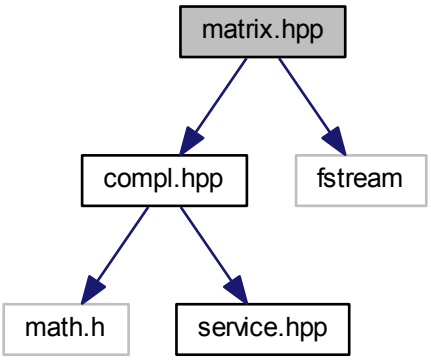
7.10.1.6 std::ofstream& operator<< (std::ofstream & *out*, const matrixC & *m*)

Definition at line 307 of file matrix.cpp.

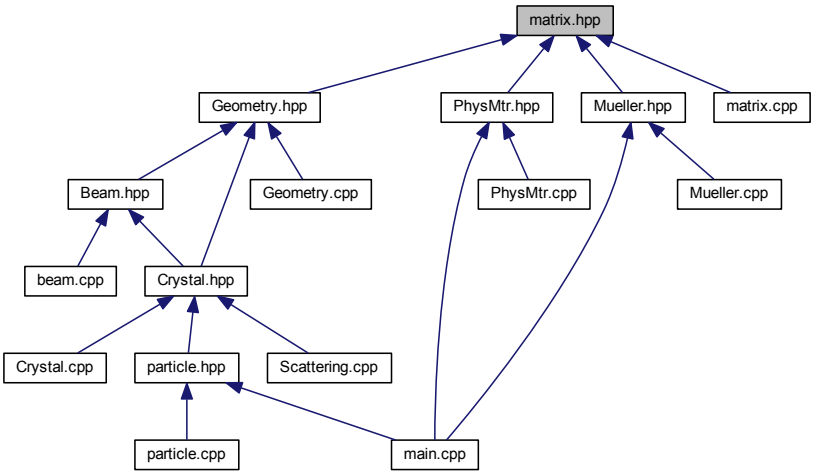
7.11 matrix.hpp File Reference

```
#include "compl.hpp"
#include <fstream>
```

Include dependency graph for matrix.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class matrix

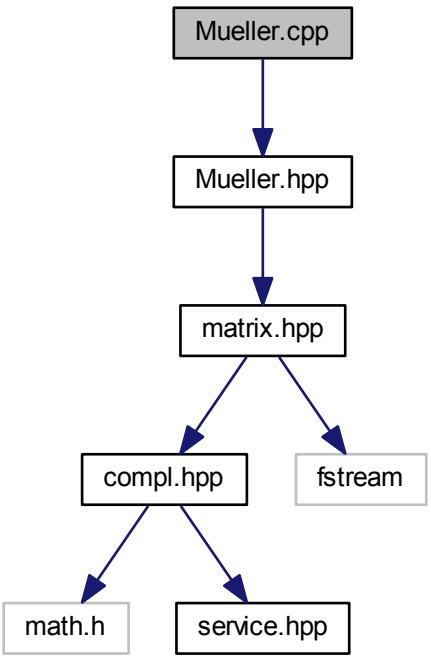
  *The array with (n-rows x m-columns) dimensions of **real** values. Size of the array can't be changed.*
- class matrixC

  *The array with (n-rows x m-columns) dimensions of **complex** values. Size of the array can't be changed.*

7.12 Mueller.cpp File Reference

```
#include "Mueller.hpp"
```

Include dependency graph for Mueller.cpp:



Functions

- matrix Mueller (const matrixC &in)

  The function returns Mueller matrix calculated from Jones matrix.
- void RightRotateMueller (matrix &m, double cs, double sn)

  *Right multiplication of matrix **m** by rotation matrix with cos(f)=cs, sin(f)=sn.*
- void LeftRotateMueller (matrix &m, double cs, double sn)

  *Left multiplication of matrix **m** by rotation matrix with cos(f)=cs, sin(f)=sn.*
- void RotateMueller (matrix &m, double _cs, double _sn, double cs, double sn)

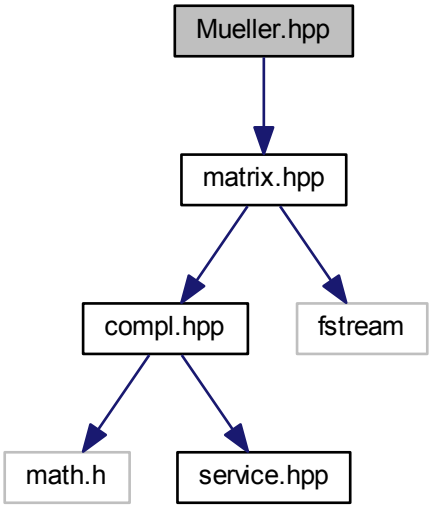
  *Multiplication of matrix **m** by left and right rotation matrixes*
- void ForwardScattering (matrix &m)

  *Calculate Mueller matrix in forward direction for randomly oriented particle. **See equation below.***
- void BackwardScattering (matrix &m)

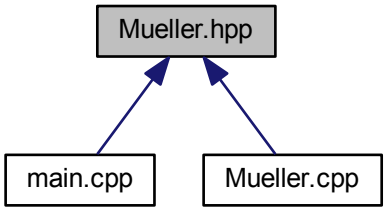
  *Calculate Mueller matrix in backward direction for randomly oriented particle. **See equation below.***

7.13 Mueller.hpp File Reference

`#include "matrix.hpp"`
Include dependency graph for Mueller.hpp:



This graph shows which files directly or indirectly include this file:



Functions

- matrix Mueller (const matrixC &in)

  The function returns Mueller matrix calculated from Jones matrix.
- void RightRotateMueller (matrix &, double, double)

  *Right multiplication of matrix **m** by rotation matrix with cos(f)=cs, sin(f)=sn.*
- void LeftRotateMueller (matrix &, double, double)

  *Left multiplication of matrix **m** by rotation matrix with cos(f)=cs, sin(f)=sn.*
- void RotateMueller (matrix &m, double _cs, double _sn, double cs, double sn)

*Multiplication of matrix **m** by left and right rotation matrixes*

- void ForwardScattering (matrix &)

  Calculate Mueller matrix in forward direction for randomly oriented particle. ***See equation below.***

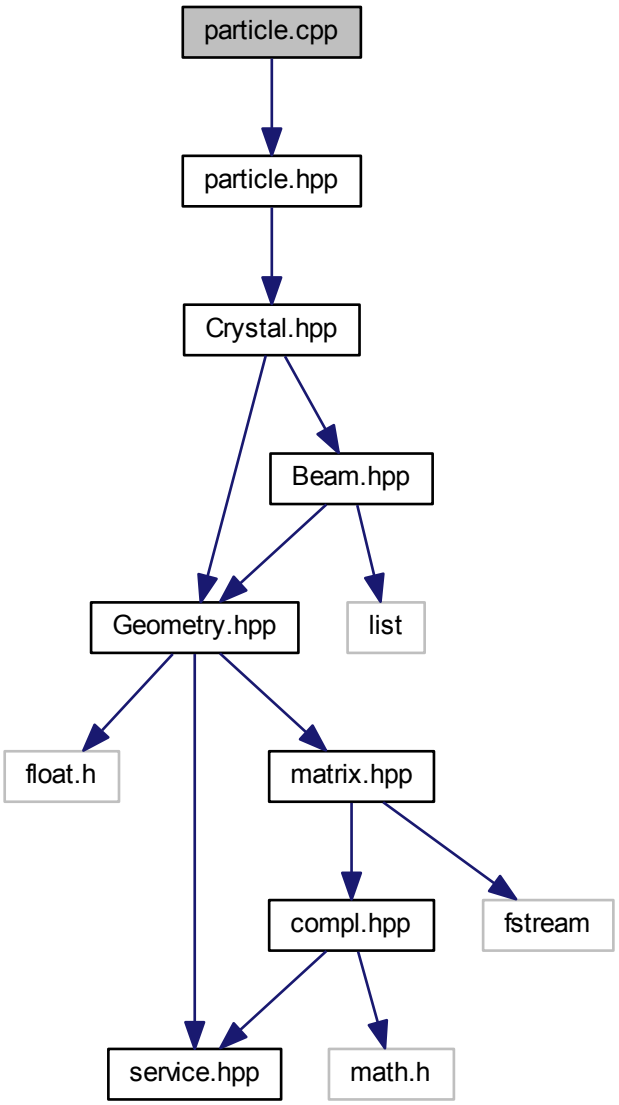
- void BackwardScattering (matrix &)

  Calculate Mueller matrix in backward direction for randomly oriented particle. ***See equation below.***

7.14 particle.cpp File Reference

```
#include "particle.hpp"
```

Include dependency graph for particle.cpp:



Macros

- #define Sqrt3_2 0.866025403784438646763723170752 94
- #define Sqrt3 1.732050807568877293527446341505 9
- #define Sqrt6 2.449489742783178098197284074705 9
- #define Sqrt2 1.414213562373095048801688724209 7
- #define Sqrt1_3 0.577350269189625764509148780501 96

7.14.1 Macro Definition Documentation

7.14.1.1 #define Sqrt1_3 0.57735026918962576450914878050196

Definition at line 7 of file particle.cpp.

7.14.1.2 #define Sqrt2 1.41421356237309504880168872420 97

Definition at line 6 of file particle.cpp.

7.14.1.3 #define Sqrt3 1.73205080756887729352744634150 59

Definition at line 4 of file particle.cpp.

7.14.1.4 #define Sqrt3_2 0.86602540378443864676372317075294

Definition at line 3 of file particle.cpp.

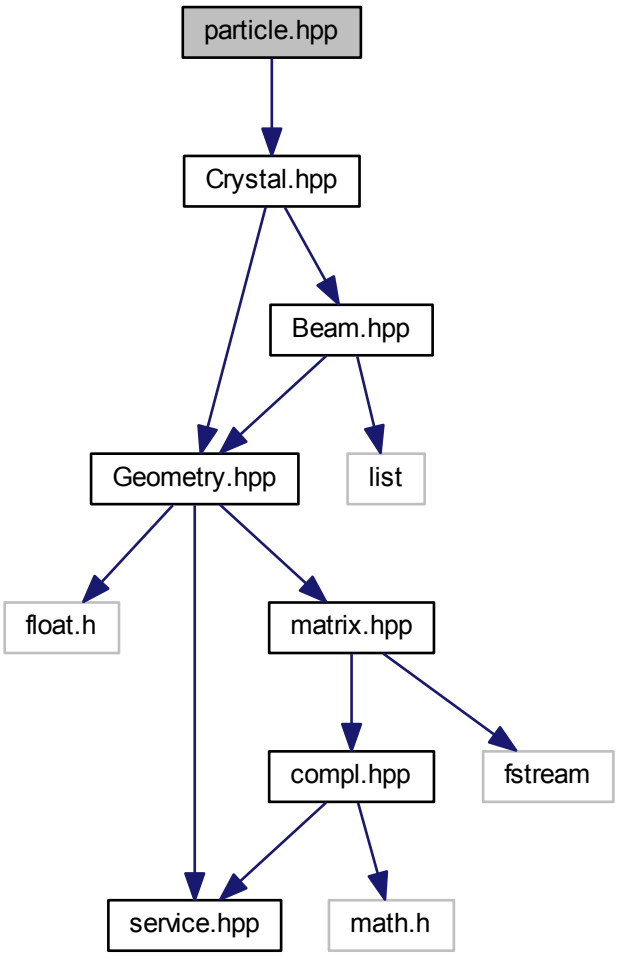
7.14.1.5 #define Sqrt6 2.44948974278317809819728407470 59

Definition at line 5 of file particle.cpp.

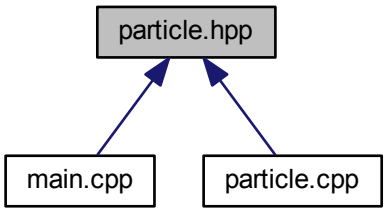
7.15 particle.hpp File Reference

```
#include "Crystal.hpp"
```

Include dependency graph for particle.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Prism

  This class defines hexagonal prism.
- class Pyramid

  This class defines the pyramid.
- class Bullet

  This class defines the hexagonal bullet.
- class TaperedPrism

  This class defines the hexagonal tapered prism.
- class CavityPrism

  This class defines the hexagonal prism with cavity on bottom.
- class Cup

  This class defines the cupped prism.
- class TrianglePrism

  This class defines the triangle prism.
- class Parallelepiped

  This class defines the parallelepiped.
- class Tetrahedron

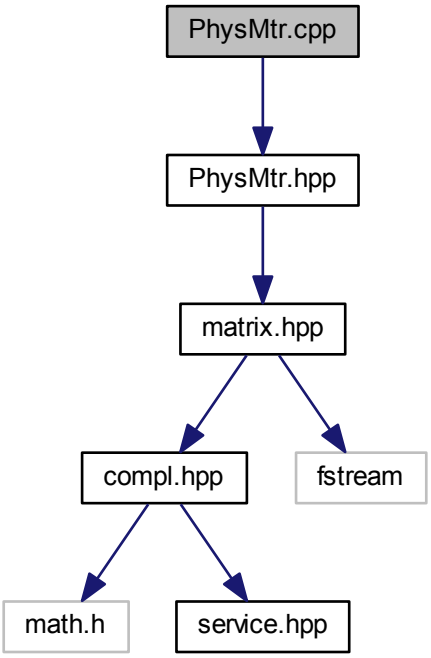
  This class defines the tetrahedron.
- class Dodecahedron

  This class defines the dodecahedron.

7.16 PhysMtr.cpp File Reference

```
#include "PhysMtr.hpp"
```

Include dependency graph for PhysMtr.cpp:



Functions

- double Max (const Arr2D &Arr)
- matrix SumArr (const Arr2D &Arr)
- matrixC SumArr (const Arr2DC &Arr)

7.16.1 Function Documentation

7.16.1.1 double Max (const Arr2D & *Arr*)

Parameters

Arr	pointer to the array

Returns

maximal element of the array

Definition at line 106 of file PhysMtr.cpp.

7.16.1.2 matrix SumArr (const Arr2D & *Arr*)

Parameters

Arr	pointer to the array

Returns

summed matrix

Definition at line 119 of file PhysMtr.cpp.

7.16.1.3 matrixC SumArr (const Arr2DC & *Arr*)

Parameters

Arr	pointer to the array

Returns

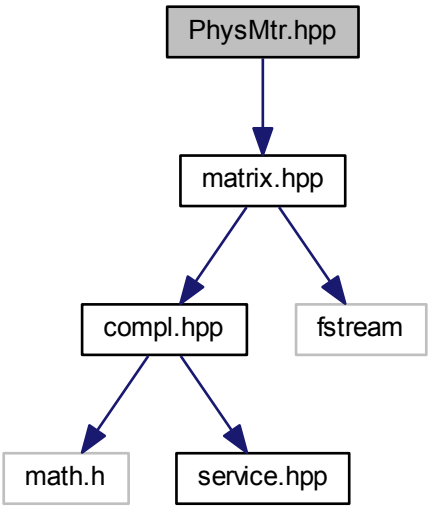
summed matrix

Definition at line 245 of file PhysMtr.cpp.

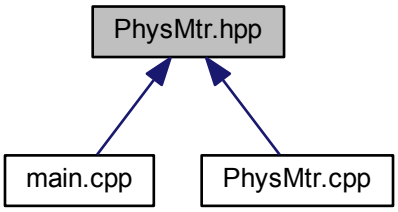
7.17 PhysMtr.hpp File Reference

```
#include "matrix.hpp"
```

Include dependency graph for PhysMtr.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Arr2D

  *The array with (N-rows x M-columns) dimensions of small **real-value** matrixes with (n x m) dimensions.*

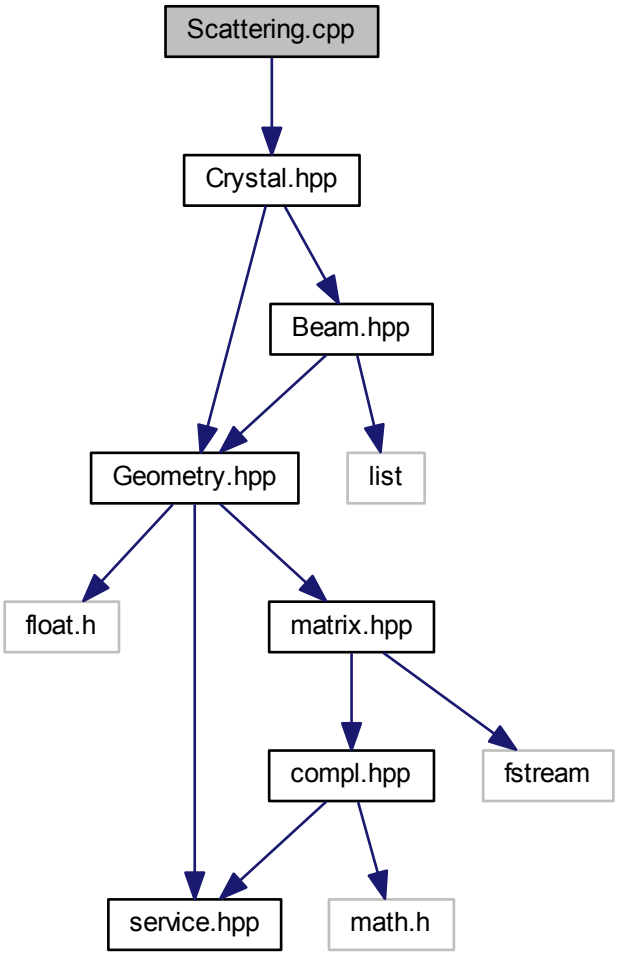
- class Arr2DC

  *The array with (N-rows x M-columns) dimensions of small **complex-value** matrixes with (n x m) dimensions.*

7.18 Scattering.cpp File Reference

```
#include "Crystal.hpp"
```

Include dependency graph for Scattering.cpp:



Variables

- const double Eps1 = 1.7453292431333680334067268304459e-4
- const double Eps2 = 0.99999999998254670756866631966593

7.18.1 Variable Documentation

7.18.1.1 const double Eps1 = 1.7453292431333680334067268304459e-4

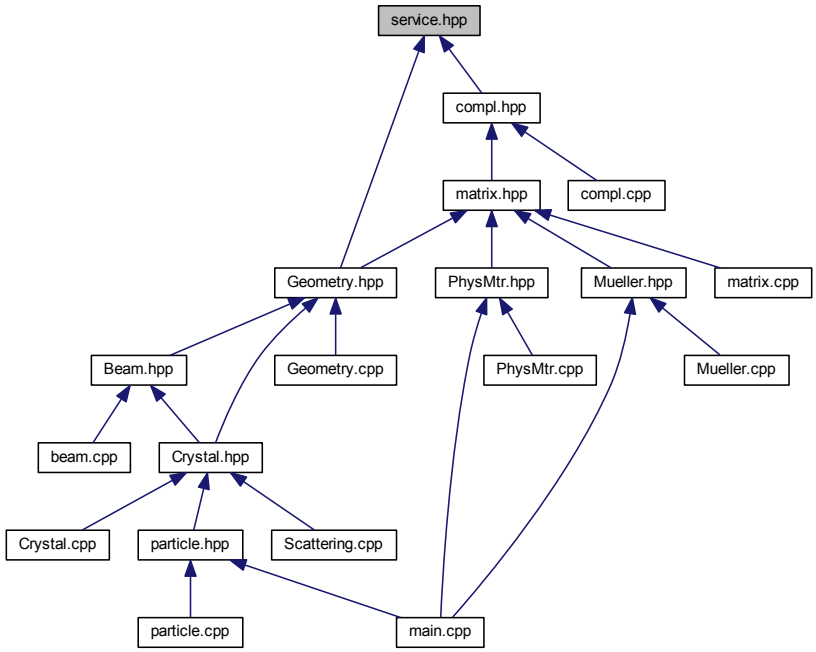
Definition at line 6 of file Scattering.cpp.

7.18.1.2 const double Eps2 = 0.99999999998254670756866631966593

Definition at line 7 of file Scattering.cpp.

7.19 service.hpp File Reference

This graph shows which files directly or indirectly include this file:



Functions

- template<class T >
  T SQR (const T &x)
- template<class T >
  T CUB (const T &x)
- template<class T >
  T ABS (const T &x)
- template<class T >
  int SGN (const T &x)
- template<class T >
  void SWAP (T &x, T &y)
- template<class T >
  int SCAL (const T &x)
- template<class T >
  T MIN (const T &x, const T &y)
- template<class T >
  T MAX (const T &x, const T &y)
- template<class T >
  T MAX (const T &x, const T &y, const T &z)
- template<class T >
  T MIN (const T &x, const T &y, const T &z)
- template<class T >
  int nMX (const T &x, const T &y)
- template<class T >
  int nMN (const T &x, const T &y)

- template<class T >
  int nMX (const T &x, const T &y, const T &z)
- template<class T >
  int nMN (const T &x, const T &y, const T &z)

7.19.1 Function Documentation

7.19.1.1 template<class T > T ABS (const T & *x*)

Definition at line 13 of file service.hpp.

7.19.1.2 template<class T > T CUB (const T & *x*)

Definition at line 10 of file service.hpp.

7.19.1.3 template<class T > T MAX (const T & *x*, const T & *y*)

Definition at line 31 of file service.hpp.

7.19.1.4 template<class T > T MAX (const T & *x*, const T & *y*, const T & *z*)

Definition at line 34 of file service.hpp.

7.19.1.5 template<class T > T MIN (const T & *x*, const T & *y*)

Definition at line 28 of file service.hpp.

7.19.1.6 template<class T > T MIN (const T & *x*, const T & *y*, const T & *z*)

Definition at line 38 of file service.hpp.

7.19.1.7 template<class T > int nMN (const T & *x*, const T & *y*)

Definition at line 45 of file service.hpp.

7.19.1.8 template<class T > int nMN (const T & *x*, const T & *y*, const T & *z*)

Definition at line 52 of file service.hpp.

7.19.1.9 template<class T > int nMX (const T & *x*, const T & *y*)

Definition at line 42 of file service.hpp.

7.19.1.10 template<class T > int nMX (const T & *x*, const T & *y*, const T & *z*)

Definition at line 48 of file service.hpp.

7.19.1.11 template<class T > int SCAL (const T & *x*)

Definition at line 24 of file service.hpp.

7.19.1.12 template<class T > int SGN (const T & *x*)

Definition at line 16 of file service.hpp.

7.19.1.13 template<class T > T SQR (const T & *x*)

Definition at line 7 of file service.hpp.

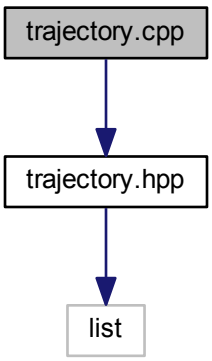
7.19.1.14 template<class T > void SWAP (T & *x*, T & *y*)

Definition at line 20 of file service.hpp.

7.20 trajectory.cpp File Reference

```
#include "trajectory.hpp"
```

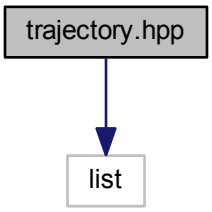
Include dependency graph for trajectory.cpp:



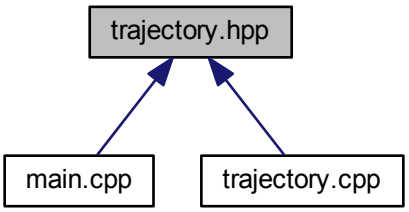
7.21 trajectory.hpp File Reference

```
#include <list>
```

Include dependency graph for trajectory.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Chain

  The class represents chain of number of facets.