DD2D - Matryoshka approach

0

Generated by Doxygen 1.8.3.1

Mon Jun 3 2013 14:47:51

Contents

Chapter 1

Main Page

The files in this program provide a heirarchical data structure system for carrying out dislocation dynamics simulations in two dimensions. The base class is Defect, which represents a generic defect in a metallic crystal. All other defects, such as dislocations, dislocation sources, precipitates, etc., are represented by their own classes which inherit certain functions from the Defect class.

The goal of carrying out these simulations in two dimensions is to be able to simulate plastic deformation of up to a few percent. Current three dimensional dislocation dynamics simulations are computationally expensive. This approach hopes to sacrifice some of the precision in order to gain in speed and flexibility.

The program is under development now, with the data structures being defined. When it will be complete, it is intended to have data structures nested within each other, hence the name `Matryoshka`. For example, a polycrystal is a collection of grains; a grain is a collection of slip systems; a slip system is a collection of slip planes; a slip plane is a collection of dislocations, dislocation sources and other defects. This program will also take advantage of the functionality provided by the C++ STL to manage lists of various objects in the simulation. Once the base simulations execute successfully, other defects will be introduced.

To view the hierarchical structure, go to the section labeled Data Structures > Class Hierarchy. A good place to start would be the Defect class, which is the generic base class for most of the entities present in the simulation.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

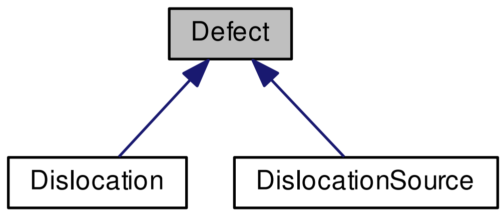
Data Structure Documentation

5.1 Defect Class Reference

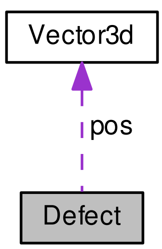
Class Defect representing a generic defect in a material.

`#include <defect.h>`

Inheritance diagram for Defect:



Collaboration diagram for Defect:



Public Member Functions

- Defect ()

    Default constructor.

- Defect (double x, double y, double z)

    Constructor specifying the position.

- Defect (double ∗p)

    Constructor specifying the position.

- void setPosition (double ∗a)

    Sets the position of the defect.

- void setPosition (double x, double y, double z)

    Sets the position of the defect.

- void setPosition (Vector3d a)

    Sets the position of the defect.

- void setX (double x)

    Sets the X-coordinate of the defect.

- void setY (double y)

    Sets the Y-coordinate of the defect.

- void setZ (double z)

    Sets the Z-coordinate of the defect.

- double ∗ getPosition ()

    Returns in an array the position.

- void getPosition (double ∗a)

    Returns the array position in a pre-allocated array.

- Vector3d getPosition ()

    Returns the position vector of the defect.

- double getX ()

    Returns the X-coordinate of the defect.

- double getY ()

    Returns the Y-coordinate of the defect.

- double getZ ()

    Returns the Z-coordinate of the defect.

- virtual Stress stressField (Vector3d p, double mu, double nu)

    Virtual function for calculating the stress field.

Protected Attributes

- Vector3d pos

    Position vector of the defect in 2D space.

5.1.1 Detailed Description

Class Defect representing a generic defect in a material.

Defines the Defect class representing an defect in the simulation. This is simply a generic description class with virtual functions. Later classes like dislocations, precipitates, boundaries etc will inherit from this class.

Definition at line 20 of file defect.h.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Defect::Defect ( )

Default constructor.

Creates the object with position (0.0, 0.0, 0.0).

Definition at line 17 of file defect.cpp.

```
{
  for (int i=0; i<3; i++)
    {
      this->pos.setValue(i, 0.0);
    }
}
```

5.1.2.2 Defect::Defect (double *x*, double *y*, double *z*)

Constructor specifying the position.

The object is initialized with the position specified by the arguments (x, y, z).

Parameters

x	X-coordinate of the defect.
y	Y-coordinate of the defect
z	Z-coordinate of the defect.

Definition at line 32 of file defect.cpp.

```
{
  this->pos.setValue (0, x);
  this->pos.setValue (1, y);
  this->pos.setValue (2, z);
}
```

5.1.2.3 Defect::Defect (double ∗ *p*)

Constructor specifying the position.

The object is initialized with the position specified in the array pointed to by the argument.

Parameters

p	Pointer to the array containing the coordinates of the defect.

Definition at line 44 of file defect.cpp.

```
{
  this->pos.setValue (p);
}
```

5.1.3 Member Function Documentation

5.1.3.1 Vector3d Defect::getPosition ( )

Returns in an array the position.

Returns the position vector of the defect.

The position of the defect is saved in an array and a pointer to its first term is returned.

Returns

Pointer to the first term of the array containing the position of the defect.
The position vector of the defect, in a variable of type Vector3d.

Definition at line 119 of file defect.cpp.

```
120 {
121   return (this->pos.getVector ());
122 }
```

5.1.3.2 void Defect::getPosition (double * a)

Returns the array position in a pre-allocated array.

Returns in the array provided in the argument the position of the defect. The array must be pre-allocated.

Parameters

a	Pointer to the location where the defect coordinates are to be populated.

Definition at line 129 of file defect.cpp.

```
130 {
131   a = this->pos.getVector ();
132 }
```

5.1.3.3 Vector3d Defect::getPosition ( )

Returns the position vector of the defect.

Returns

The position vector of the defect, in a variable of type Vector3d.

5.1.3.4 double Defect::getX ( )

Returns the X-coordinate of the defect.

Returns

X-coordinate of the defect.

Definition at line 147 of file defect.cpp.

```
148 {
149   return (this->getValue (0));
150 }
```

5.1.3.5 double Defect::getY ( )

Returns the Y-coordinate of the defect.

Returns

Y-coordinate of the defect.

Definition at line 156 of file defect.cpp.

```
{
  return (this->pos.getValue (1));
}
```

5.1.3.6 double Defect::getZ ( )

Returns the Z-coordinate of the defect.

Returns

Z-coordinate of the defect.

Definition at line 165 of file defect.cpp.

```
{
  return (this->pos.getValue (2));
}
```

5.1.3.7 void Defect::setPosition (double ∗ *a*)

Sets the position of the defect.

The position of the defect is set to the co-ordinates present in the array pointed to by the argument. Sets the position of the defect as the values in the array pointed to by the argument.

Parameters

a	Pointer to the array containing the coordinates of the defect.

Definition at line 56 of file defect.cpp.

```
{
  this->pos.setVector (a);
}
```

5.1.3.8 void Defect::setPosition (double *x*, double *y*, double *z*)

Sets the position of the defect.

The position of the defect is set to the co-ordinates specified by the arguments (x, y, z). Sets the position of the defect as the coordinates provided as arguments.

Parameters

x	X-coordinate of the defect.
y	Y-coordinate of the defect.
z	Z-coordinate of the defect.

Definition at line 69 of file defect.cpp.

```
{
  this->pos.setValue (0, x);
```

```
  this->pos.setValue (1, y);
  this->pos.setValue (2, z);
}
```

5.1.3.9 void Defect::setPosition (Vector3d *a*)

Sets the position of the defect.

The position of the defect is set to the position vector fiven by the argument a.

Parameters

a	Position vector of the defect.

Definition at line 81 of file defect.cpp.

```
{
  this->position = a;
}
```

5.1.3.10 void Defect::setX (double *x*)

Sets the X-coordinate of the defect.

Parameters

x	X-coordinate of the defect.

Definition at line 90 of file defect.cpp.

```
{
  this->pos.setValue (0, x);
}
```

5.1.3.11 void Defect::setY (double *y*)

Sets the Y-coordinate of the defect.

Parameters

y	Y-coordinate of the defect.

Definition at line 99 of file defect.cpp.

```
{
  this->pos.setValue (1, y);
}
```

5.1.3.12 void Defect::setZ (double *z*)

Sets the Z-coordinate of the defect.

Parameters

z	Z-coordinate of the defect.

Definition at line 108 of file defect.cpp.

```
{
  this->pos.setValue (2, z);
}
```

5.1.3.13 virtual Stress Defect::stressField (Vector3d *p*, double *mu*, double *nu*) [inline], [virtual]

Virtual function for calculating the stress field.

Returns the value of the stress field of the given defect at the position given by the argument. This is a virtual function and always returns a zero matrix. Classes which inherit this function should have their own implementations of this function to override its behaviour.

Parameters

p	Position vector of the the point where the stress field is to be calculated.
mu	Shear modulus in Pascals.
nu	Poisson's ratio.

Returns

Stress field value at the position p.

Reimplemented in Dislocation.

Definition at line 142 of file defect.h.

```
  {
    // This virtual function returns a zero matrix.
    // Inheriting classes will have functions implementing this in their own way
    // They will override this behaviour.
    Stress s;
    return (s);
  }
```

5.1.4 Field Documentation

5.1.4.1 Vector3d Defect::pos [protected]

Position vector of the defect in 2D space.

Definition at line 26 of file defect.h.

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

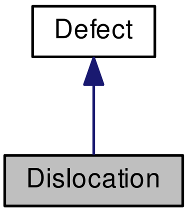
- defect.h
- defect.cpp

5.2 Dislocation Class Reference

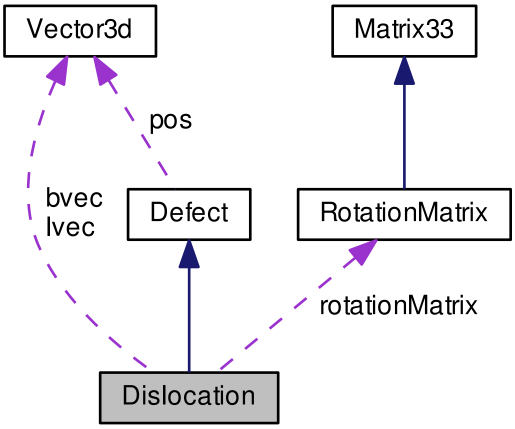
Dislocation class representing a dislocation in the simulation.

`#include <dislocation.h>`

Inheritance diagram for Dislocation:



Collaboration diagram for Dislocation:



Public Member Functions

- Dislocation ()

    Default constructor.
- Dislocation (Vector3d burgers, Vector3d line, Vector3d position, double bm, bool m)

    Constructor that explicitly specifies all parameters.
- void setBurgers (Vector3d burgers)

    Sets the Burgers vector of the dislocation.
- void setLineVector (Vector3d line)

    Sets the line vector of the dislocation.
- void setMobile ()

    Sets the dislocation as mobile.
- void setPinned ()

    Sets the dislocation as pinned.

- Vector3d getBurgers ()

  Gets the Burgers vector of the dislocation.
- Vector3d getLineVector ()

  Gets the line vector of the dislocation.
- bool isMobile ()

  Returns whether the dislocation is mobile or pinned.
- void calculateRotationMatrix ()

  Calculate the roation matrix.
- Stress stressField (Vector3d p, double mu, double nu)

  Calculates the stress field due to this dislocation at the position given as argument.
- Stress stressFieldLocal (Vector3d p, double mu, double nu)

  Calculates the stress field due to the dislocation in the local co-ordinate system.
- Vector3d forcePeachKoehler (Stress sigma, double tau_crss)

  Calculate the Peach-Koehler force acting on the dislocation due the stress.
- double idealTimeIncrement (Vector3d v0, double minDistance, Defect d, Vector3d v1)

  Returns the ideal time increment for the dislocation.
- void setPosition (double ∗a)

  Sets the position of the defect.
- void setPosition (double x, double y, double z)

  Sets the position of the defect.
- void setPosition (Vector3d a)

  Sets the position of the defect.
- void setX (double x)

  Sets the X-coordinate of the defect.
- void setY (double y)

  Sets the Y-coordinate of the defect.
- void setZ (double z)

  Sets the Z-coordinate of the defect.
- double ∗ getPosition ()

  Returns in an array the position.
- void getPosition (double ∗a)

  Returns the array position in a pre-allocated array.
- Vector3d getPosition ()

  Returns the position vector of the defect.
- double getX ()

  Returns the X-coordinate of the defect.
- double getY ()

  Returns the Y-coordinate of the defect.
- double getZ ()

  Returns the Z-coordinate of the defect.

Protected Attributes

- Vector3d bvec

  Burgers vector of the dislocation.
- Vector3d lvec

  Line vector if the dislocation.
- bool mobile

  Boolean term indicating mobility.
- double bmag

Magnitude of the Burgers vector in metres.

- RotationMatrix rotationMatrix

  The rotation matrix for rotating from the global to the local co-ordinate system and vice-versa.
- Vector3d pos

  Position vector of the defect in 2D space.

5.2.1 Detailed Description

Dislocation class representing a dislocation in the simulation.

The Dislocation class represents a dislocation in the simulation. The class inherits from the Defect class. A dislocation has several properties like a Burgers vector, line vector, etc. which will all be declared here.

Definition at line 21 of file dislocation.h.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Dislocation::Dislocation ( )

Default constructor.

Initializes the dislocation with the following default parameters: Position: (0.0, 0.0, 0.0) Burgers vector: Default value set in defaults file. Line vector: Default value set in defaults file. Burgers vector magnitude: Default value set in the defaults file. Mobile: true.

Initializes the dislocation with the following default parameters: Position: (0.0, 0.0, 0.0) Burgers vector: Default value set in defaults file. Line vector: Default value set in defaults file. Burgers vector magnitude: Default value set in teh defaults file. Mobile: true.

Definition at line 21 of file dislocation.cpp.

```
{
  this->setPosition ( 0.0, 0.0, 0.0 );
  this->setBurgers ( Vector3d ( DEFAULT_BURGERS_0,
   DEFAULT_BURGERS_1, DEFAULT_BURGERS_2 ) );
  this->setLineVector ( Vector3d ( DEFAULT_LINEVECTOR_0,
   DEFAULT_LINEVECTOR_1, DEFAULT_LINEVECTOR_2) );
  this->bmag = DEFAULT_BURGERS_MAGNITUDE;
  this->mobile = true;
  this->calculateRotationMatrix ();
}
```

5.2.2.2 Dislocation::Dislocation (Vector3d *burgers,* Vector3d *line,* Vector3d *position,* double *bm,* bool *m*)

Constructor that explicitly specifies all parameters.

All parameters: Burgers vector, line vector, position, are specified.

Parameters

burgers	Burgers vector.
line	Line vector.
position	Position of the dislocation.
bm	Magnitude of the Burgers vector in metres.
m	Mobility (true/false).

Definition at line 40 of file dislocation.cpp.

```
{
  this->bvec   = burgers;
  this->lvec   = line;
```

```
  this->pos     = position;
  this->mobile = m;
  this->bmag    = bm;
  this->calculateRotationMatrix ();
}
```

5.2.3 Member Function Documentation

5.2.3.1 void Dislocation::calculateRotationMatrix ( )

Calculate the roation matrix.

This function calculates the rotation matrix for this dislocation using the global and local co-ordinate systems. The matrix rotationMatrix is for rotation from the old (unprimed, global) to the new (primed, dislocation) system.

Definition at line 118 of file dislocation.cpp.

```
{
  Vector3d globalSystem[3];      // Global co-ordinate systems
  Vector3d localSystem[3];       // Dislocation co-ordinate system

  // Vectors of the global co-ordinate system
  globalSystem[0] = Vector3d (1.0, 0.0, 0.0);
  globalSystem[1] = Vector3d (0.0, 1.0, 0.0);
  globalSystem[2] = Vector3d (0.0, 0.0, 1.0);

  // Vectors of the dislocation co-ordinate system
  localSystem[0] = bvec.normalize ();
  localSystem[2] = lvec.normalize ();
  localSystem[1] = (lvec ^ bvec).normalize ();

  // Calculate rotation matrix
  this->rotationMatrix = RotationMatrix (globalSystem, localSystem);
}
```

5.2.3.2 Vector3d Dislocation::forcePeachKoehler (Stress *sigma*, double *tau_crss*)

Calculate the Peach-Koehler force acting on the dislocation due the stress.

This function calculates the Peach-Koehler force in the dislocation due to the stress (expressed in the global co-ordinate system) provided as argument. The force returned is also in the global co-ordinate system. This function checks if the xy component of the stress tensorm expressed in the dislocation's local co-ordinate system, is greater than tau_crss. If it is, the force is calculated using the Peach-Koehler equation, otherwise, the force on the dislocation is zero.

Parameters

sigma	The stress tensor, expressed in the global co-ordinate system.
tau_crss	Critical Resolved Shear Stress in Pa.

Returns

The Peach-Koehler force on the dislocation, expressed in the global co-ordinate system.

Definition at line 201 of file dislocation.cpp.

```
{
  // Stress in the local co-ordinate system
  Stress sigmaLocal = (this->rotationMatrix) * (sigma) * (this->
  rotationMatrix)^;
  Vector3d force;

  // Check for CRSS condition
  if (sigmaLocal.getValue(0,1) >= tau_crss)
    {
      Vector3d force = sigma * ((this->bvec)^(this->lvec));
    }

```

```
  return (force);
}
```

5.2.3.3 Vector3d Dislocation::getBurgers ( )

Gets the Burgers vector of the dislocation.

Returns

Burgers vector in a variable of type Vector3d.

Definition at line 90 of file dislocation.cpp.

```
{
  return ( this->bvec );
}
```

5.2.3.4 Vector3d Dislocation::getLineVector ( )

Gets the line vector of the dislocation.

Returns

Line vector in a variable of type Vector3d.

Definition at line 99 of file dislocation.cpp.

```
{
  return ( this->lvec );
}
```

5.2.3.5 Vector3d Defect::getPosition ( ) [inherited]

Returns in an array the position.

Returns the position vector of the defect.

The position of the defect is saved in an array and a pointer to its first term is returned.

Returns

Pointer to the first term of the array containing the position of the defect.
The position vector of the defect, in a variable of type Vector3d.

Definition at line 119 of file defect.cpp.

```
{
  return (this->pos.getVector ());
}
```

5.2.3.6 void Defect::getPosition (double ∗ *a*) [inherited]

Returns the array position in a pre-allocated array.

Returns in the array provided in the argument the position of the defect. The array must be pre-allocated.

Parameters

a	Pointer to the location where the defect coordinates are to be populated.

Definition at line 129 of file defect.cpp.

```
{
  a = this->pos.getVector ();
}
```

5.2.3.7 Vector3d Defect::getPosition ( ) [inherited]

Returns the position vector of the defect.

Returns

The position vector of the defect, in a variable of type Vector3d.

5.2.3.8 double Defect::getX ( ) [inherited]

Returns the X-coordinate of the defect.

Returns

X-coordinate of the defect.

Definition at line 147 of file defect.cpp.

```
{
  return (this->getValue (0));
}
```

5.2.3.9 double Defect::getY ( ) [inherited]

Returns the Y-coordinate of the defect.

Returns

Y-coordinate of the defect.

Definition at line 156 of file defect.cpp.

```
{
  return (this->pos.getValue (1));
}
```

5.2.3.10 double Defect::getZ ( ) [inherited]

Returns the Z-coordinate of the defect.

Returns

Z-coordinate of the defect.

Definition at line 165 of file defect.cpp.

```
{
  return (this->pos.getValue (2));
}
```

5.2.3.11 double Dislocation::idealTimeIncrement (Vector3d *v0*, double *minDistance*, Defect *d*, Vector3d *v1*)

Returns the ideal time increment for the dislocation.

A dislocation is not allowed to approach another defect beyond a certain distance, specified by the argument minDistance. This function calculates the ideal time increment for this dislocation to not collide with the defect.

Parameters

v0	Velocity of the dislocation.
minDistance	Minimum distance of approach to the defect.
d	The defect for which the present dislocation's time increment is to be calculated.
v1	Velocity of the other defect.

Returns

The ideal time increment for this dislocation.

Definition at line 225 of file dislocation.cpp.

```
{
  double norm_v0 = v0.magnitude();
  if (norm_v0 == 0.0)
    {
      // This dislocation is not moving
      return (1000.0);
    }

  // Positions
  Vector3d p0 = this->getPosition();
  Vector3d p1 = d.getPosition();
  Vector3d p01 = p1 - p0;
  double norm_p01 = p01.magnitude();

  if (norm_p01 == 0.0)
    {
      // The dislocation is lying on top of the obstacle - so it should not move
      return (0.0);
    }
  else
    {
      // Find out if the dislocation is approaching the defect or not

      // Velocities
      Vector3d v01 = v1 - v0;
      double norm_v01 = v01.magnitude();
      double dotProduct = v01 * p01;
      double cosine = dotProduct/(norm_v01 * norm_p01);
      if (cosine < 0.0)
        {
          // The dislocation is approaching the other defect
          return ( (norm_p01 - minDistance)/norm_v01 );
        }
      else
        {
          // They are diverging
          // So any time increment will do
          return (1000.0);
        }
    }
}
```

5.2.3.12 bool Dislocation::isMobile ( )

Returns whether the dislocation is mobile or pinned.

Returns

Returns true if the dislocation is mobile, false if pinned.

Definition at line 108 of file dislocation.cpp.

```
{
  return (this->mobile);
}
```

5.2.3.13 void Dislocation::setBurgers (Vector3d *burgers*)

Sets the Burgers vector of the dislocation.

Parameters

burgers	Bergers vector of the dislocation.

Definition at line 54 of file dislocation.cpp.

```
{
  this->bvec = burgers;
}
```

5.2.3.14 void Dislocation::setLineVector (Vector3d *line*)

Sets the line vector of the dislocation.

Parameters

line	Line vector of the dislocation.

Definition at line 62 of file dislocation.cpp.

```
{
  this->lvec = line;
}
```

5.2.3.15 void Dislocation::setMobile ( )

Sets the dislocation as mobile.

Sets the flag mobile to true.

Definition at line 71 of file dislocation.cpp.

```
{
  this->mobile = true;
}
```

5.2.3.16 void Dislocation::setPinned ( )

Sets the dislocation as pinned.

Sets the flag mobile to false.

Definition at line 80 of file dislocation.cpp.

```
{
  this->mobile = false;
}
```

5.2.3.17 void Defect::setPosition (double * *a*) [inherited]

Sets the position of the defect.

The position of the defect is set to the co-ordinates present in the array pointed to by the argument. Sets the position of the defect as the values in the array pointed to by the argument.

Parameters

a	Pointer to the array containing the coordinates of the defect.

Definition at line 56 of file defect.cpp.

```
57 {
58   this->pos.setVector (a);
59 }
```

5.2.3.18 void Defect::setPosition (double *x*, double *y*, double *z*) [inherited]

Sets the position of the defect.

The position of the defect is set to the co-ordinates specified by the arguments (x, y, z). Sets the position of the defect as the coordinates provided as arguments.

Parameters

x	X-coordinate of the defect.
y	Y-coordinate of the defect.
z	Z-coordinate of the defect.

Definition at line 69 of file defect.cpp.

```
70 {
71   this->pos.setValue (0, x);
72   this->pos.setValue (1, y);
73   this->pos.setValue (2, z);
74 }
```

5.2.3.19 void Defect::setPosition (Vector3d *a*) [inherited]

Sets the position of the defect.

The position of the defect is set to the position vector fiven by the argument a.

Parameters

a	Position vector of the defect.

Definition at line 81 of file defect.cpp.

```
82 {
83   this->position = a;
84 }
```

5.2.3.20 void Defect::setX (double *x*) [inherited]

Sets the X-coordinate of the defect.

Parameters

x	X-coordinate of the defect.

Definition at line 90 of file defect.cpp.

```
{
  this->pos.setValue (0, x);
}
```

5.2.3.21 void Defect::setY (double *y*) [inherited]

Sets the Y-coordinate of the defect.

Parameters

y	Y-coordinate of the defect.

Definition at line 99 of file defect.cpp.

```
{
  this->pos.setValue (1, y);
}
```

5.2.3.22 void Defect::setZ (double *z*) [inherited]

Sets the Z-coordinate of the defect.

Parameters

z	Z-coordinate of the defect.

Definition at line 108 of file defect.cpp.

```
{
  this->pos.setValue (2, z);
}
```

5.2.3.23 Stress Dislocation::stressField (Vector3d *p*, double *mu*, double *nu*) [virtual]

Calculates the stress field due to this dislocation at the position given as argument.

The stress field of the dislocation is calculated at the position indicated by the argument.

Parameters

p	Position vector of the point where the stress field is to be calculated.
mu	Shear modulus in Pascals.
nu	Poisson's ratio.

Returns

Stress tensor, expressed in the global co-ordinate system, giving the value of the stress field at position p.

Reimplemented from Defect.

Definition at line 146 of file dislocation.cpp.

```
{
  double principalStresses[3];
  double shearStresses[3];
  Vector3d r;  // Vector joining the present dislocation to the point p

  r = p - this->pos; // Still in global coordinate system
  Vector3d rLocal = this->rotationMatrix * r;     // Rotated to local co-ordinate system

  // Calculate the stress field in the local co-ordinate system
  Stress sLocal = this->stressFieldLocal (rLocal, mu, nu);

  // Calculate the stress field in the global co-ordinate system
  Stress sGlobal = (this->rotationMatrix) * sLocal * (^(this->rotationMatrix));

  return (sGlobal);
}
```

5.2.3.24 Stress Dislocation::stressFieldLocal (Vector3d *p*, double *mu*, double *nu*)

Calculates the stress field due to the dislocation in the local co-ordinate system.

The stress field due to the dislocation is calculated at the position indicated by the argument. The stress tensor is expressed in the dislocation's local co-ordinate system.

Parameters

p	Position vector of the point where the stress field is to be calculated. This position vector is calculated in the local co-ordinate system, taking the dislocation as the origin.
mu	Shear modulus in Pascals.
nu	Poisson's ratio.

Returns

Stress tensor, expressed in the dislocation's local co-ordinate system.

Definition at line 172 of file dislocation.cpp.

```
{
  double D = ( mu * this->bm ) / ( 2.0 * PI * ( 1.0 - nu ) ); // Constant for all components of the stress tensor

  double x, y, denominator;     // Terms that appear repeatedly in the stress tensor

  x = p.getValue (0);
  y = p.getValue (1);
  denominator = pow ( ((x*x) + (y*y)), 2);

  principalStresses[0] = -1.0 * D * y * ( (3.0*x*x) + (y*y) ) / denominator;
  principalStresses[1] = D * y * ( (x*x) - (y*y) ) / denominator;
  principalStresses[2] = nu * ( principalStresses[0] + principalStresses[1] );

  shearStresses[0] = D * x * ( (x*x) - (y*y) ) / denominator;
  shearStresses[1] = 0.0;
  shearStresses[2] = 0.0;

  return (Stress(principalStresses, shearStresses));
}
```

5.2.4 Field Documentation

5.2.4.1 double Dislocation::bmag [protected]

Magnitude of the Burgers vector in metres.

The magnitude of the Burgers vector is useful for several calculations such as stress field around the dislocation.

Definition at line 44 of file dislocation.h.

5.2.4.2 Vector3d Dislocation::bvec [protected]

Burgers vector of the dislocation.

Definition at line 27 of file dislocation.h.

5.2.4.3 Vector3d Dislocation::lvec [protected]

Line vector if the dislocation.

Definition at line 32 of file dislocation.h.

5.2.4.4 bool Dislocation::mobile [protected]

Boolean term indicating mobility.

For mobile dislocations this term is true and for pinned dislocations it is false.

Definition at line 38 of file dislocation.h.

5.2.4.5 Vector3d Defect::pos [protected], [inherited]

Position vector of the defect in 2D space.

Definition at line 26 of file defect.h.

5.2.4.6 RotationMatrix Dislocation::rotationMatrix [protected]

The rotation matrix for rotating from the global to the local co-ordinate system and vice-versa.

This is the rotation matrix that represents the relationship between the global and local co-ordinate systems. It is used to convert tensors and vectors between the two systems. The rotation matrix needs to be calculated once and may be refreshed periodically if lattice rotation is implemented. In the absence of lattice rotation, the matrix will remain invariant.

Definition at line 50 of file dislocation.h.

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

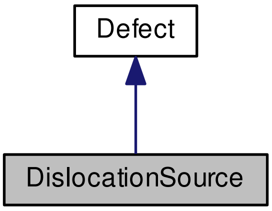
- dislocation.h
- dislocation.cpp

5.3 DislocationSource Class Reference

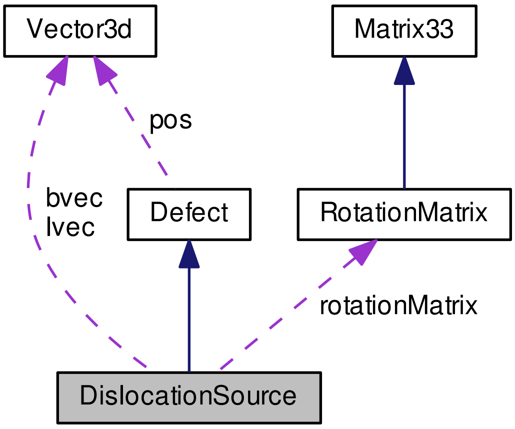
DislocationSource class representing a source of dislocations in the simulation.

`#include <dislocationSource.h>`

Inheritance diagram for DislocationSource:



Collaboration diagram for DislocationSource:



Public Member Functions

- DislocationSource ()

  Default constructor.
- DislocationSource (Vector3d burgers, Vector3d line, Vector3d position, double bm, double tau, int nIter)

  Constructor that explicitly specifies all parameters.
- void setBurgers (Vector3d burgers)

  Sets the Burgers vector of the dislocation.
- void setLineVector (Vector3d line)

  Sets the line vector of the dislocation.
- void setBurgersMagnitude (double bm)

  Set the magnitude of the Burgers vector.
- void setTauCritical (double tauC)

  Set the critical shear stres for dipole emission.

- void setNumIterations (int nIter)

    Set the number of iterations before a dipole is emitted.
- void resetIterationCounter ()

    Sets the iteration counter to zero.
- Vector3d getBurgers ()

    Returns the Burgers vector of the dislocations in the dipole.
- Vector3d getLineVector ()

    Returns the line vector of the dislocations in the dipole.
- double getBurgersMag ()

    Returns the magnitude of the Burgers vector.
- double getTauCritical ()

    Returns the critical shear stress value for dipole emission.
- int getNumIterations ()

    Returns the number if iterations that the dislocation source must spend experiencing a shear stress greater than the critical value before it can emit a dislocation dipole.
- int getIterationCount ()

    Get the count of the iterations spent at higher than critical shear stress.
- double dipoleNucleationLength (double tau, double mu, double nu)

    The nucleation length of the dipole.
- void incrementIterationCount ()

    Increments the variable countIterations by 1.
- bool ifEmitDipole ()

    Checks if the dislocation source has experienced higher than critical shear stress for long enough to emit a dipole.
- void setPosition (double ∗a)

    Sets the position of the defect.
- void setPosition (double x, double y, double z)

    Sets the position of the defect.
- void setPosition (Vector3d a)

    Sets the position of the defect.
- void setX (double x)

    Sets the X-coordinate of the defect.
- void setY (double y)

    Sets the Y-coordinate of the defect.
- void setZ (double z)

    Sets the Z-coordinate of the defect.
- double ∗ getPosition ()

    Returns in an array the position.
- void getPosition (double ∗a)

    Returns the array position in a pre-allocated array.
- Vector3d getPosition ()

    Returns the position vector of the defect.
- double getX ()

    Returns the X-coordinate of the defect.
- double getY ()

    Returns the Y-coordinate of the defect.
- double getZ ()

    Returns the Z-coordinate of the defect.
- virtual Stress stressField (Vector3d p, double mu, double nu)

    Virtual function for calculating the stress field.

Protected Attributes

- Vector3d bvec

    Burgers vector of the dislocation.
- Vector3d lvec

    Line vector if the dislocation.
- bool mobile

    Boolean term indicating mobility.
- double bmag

    Magnitude of the Burgers vector in metres.
- RotationMatrix rotationMatrix

    The rotation matrix for rotating from the global to the local co-ordinate system and vice-versa.
- double tauCritical

    Critical stress for the emission of a dislocation dipole.
- int nIterations

    Number of iterations before a dipole is emitted.
- int countIterations

    Counter variable for the number of consecutive iterations the dislocation source has experienced a shear stress greater than its critical value.
- Vector3d pos

    Position vector of the defect in 2D space.

5.3.1 Detailed Description

DislocationSource class representing a source of dislocations in the simulation.

This class inherits from the Defect class. This object is basically the representation of a Frank-Read source emitting dislocation dipoles. When the dislocation source experiences a shear stress greater than a critical value for a certain amount of time (or number of iterations), it emits a dislocation dipole with a length that is a function of the applied stress. The properties of this class and the member functions will be declared here.

Definition at line 22 of file dislocationSource.h.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 DislocationSource::DislocationSource ( )

Default constructor.

Initializes the dislocation with the default parameters provided in the files dislocationDefaults.h and dislocationSourceDefaults.h.

Definition at line 17 of file dislocationSource.cpp.

```
{
  this->setPosition ( Vector3d ( DEFAULT_POSITION_0,
   DEFAULT_POSITION_1, DEFAULT_POSITION_2 ) );
  this->setBurgers ( Vector3d ( DEFAULT_BURGERS_0,
   DEFAULT_BURGERS_1, DEFAULT_BURGERS_2 ) );
  this->setLineVector ( Vector3d ( DEFAULT_LINEVECTOR_0,
   DEFAULT_LINEVECTOR_1, DEFAULT_LINEVECTOR_2) );
  this->bmag = DEFAULT_BURGERS_MAGNITUDE;
  this->tauCritical = DEFAULT_TAU_CRITICAL;
  this->nIterations = DEFAULT_NITERATIONS;
  this->countIterations = 0;
}
```

5.3.2.2 DislocationSource::DislocationSource (Vector3d *burgers*, Vector3d *line*, Vector3d *position*, double *bm*, double *tau*, int *nIter*)

Constructor that explicitly specifies all parameters.

All parameters: Burgers vector, line vector, position, are specified.

Parameters

burgers	Burgers vector.
line	Line vector.
position	Position of the dislocation source.
bm	Magnitude of the Burgers vector in metres.
tau	Critical shear stress value.
nIter	Number of iterations of experiencing critical stress before a dipole is emitted.

All parameters: Burgers vector, line vector, position, are specified.

Parameters

burgers	Burgers vector.
line	Line vector.
position	Position of the dislocation.
bm	Magnitude of the Burgers vector in metres.
tau	Critical shear stress value.
nIter	Number of iterations of experiencing critical stress before a dipole is emitted.

Definition at line 38 of file dislocationSource.cpp.

```
{
  this->bvec    = burgers;
  this->lvec    = line;
  this->pos     = position;
  this->bmag    = bm;
  this->tauCritical = tau;
  this->nIterations = nIter;
  this->countIterations = 0;
}
```

5.3.3 Member Function Documentation

5.3.3.1 double DislocationSource::dipoleNucleationLength (double *tau*, double *mu*, double *nu*)

The nucleation length of the dipole.

When a dislocation source has experienced a shear stress greater than the critical value for a certain amount of time, it emits a dislocation dipole. In three dimensions, this is equivalent to a dislocation loop emitted by a Frank--Read source. The length of the dipole (or diameter of the loop in 3D) is such that the interaction force between the two dislocations (or line tension in 3D) balances out the applied shear stress.

Parameters

tau	The shear stress experienced by the dislocation source.
mu	Shear modulus of the material, in Pa.
nu	Poisson's ratio.

Returns

The length of the dislocation dipole.

Definition at line 167 of file dislocationSource.cpp.

```
{
  double L = 0.0;

  if (tau >= tauCritical)
  {
    L = (mu * this->bmag) / ( 2.0 * PI * (1.0 - nu) * this->tauCritical );
  }

  return (L);
}
```

5.3.3.2 Vector3d DislocationSource::getBurgers ( )

Returns the Burgers vector of the dislocations in the dipole.

Returns

The Burgers vector of the dislocations in the dipole.

Definition at line 108 of file dislocationSource.cpp.

```
{
  return (this->bvec);
}
```

5.3.3.3 double DislocationSource::getBurgersMag ( )

Returns the magnitude of the Burgers vector.

Returns

The magnitude of the Burgers vector.

Definition at line 126 of file dislocationSource.cpp.

```
{
  return (this->bmag);
}
```

5.3.3.4 int DislocationSource::getIterationCount ( )

Get the count of the iterations spent at higher than critical shear stress.

Returns

Number of iterations spent at higher than critical shear stress.

Definition at line 153 of file dislocationSource.cpp.

```
{
  return (this->countIterations);
}
```

5.3.3.5 Vector3d DislocationSource::getLineVector ( )

Returns the line vector of the dislocations in the dipole.

Returns

The line vector of the dislocations in the dipole.

Definition at line 117 of file dislocationSource.cpp.

```
{
  return (this->lvec);
}
```

5.3.3.6 int DislocationSource::getNumIterations ( )

Returns the number if iterations that the dislocation source must spend experiencing a shear stress greater than the critical value before it can emit a dislocation dipole.

Returns

The number if iterations that the dislocation source must spend experiencing a shear stress greater than the critical value before it can emit a dislocation dipole.

Definition at line 144 of file dislocationSource.cpp.

```
{
  return (this->nIterations);
}
```

5.3.3.7 Vector3d Defect::getPosition ( ) [inherited]

Returns in an array the position.

Returns the position vector of the defect.

The position of the defect is saved in an array and a pointer to its first term is returned.

Returns

Pointer to the first term of the array containing the position of the defect.
The position vector of the defect, in a variable of type Vector3d.

Definition at line 119 of file defect.cpp.

```
{
  return (this->pos.getVector ());
}
```

5.3.3.8 void Defect::getPosition (double ∗ a) [inherited]

Returns the array position in a pre-allocated array.

Returns in the array provided in the argument the position of the defect. The array must be pre-allocated.

Parameters

a	Pointer to the location where the defect coordinates are to be populated.

Definition at line 129 of file defect.cpp.

```
{
  a = this->pos.getVector ();
```

```
}
```

5.3.3.9 Vector3d Defect::getPosition ( ) [inherited]

Returns the position vector of the defect.

Returns

The position vector of the defect, in a variable of type Vector3d.

5.3.3.10 double DislocationSource::getTauCritical ( )

Returns the critical shear stress value for dipole emission.

Returns

The critical shear stress value for dipole emission.

Definition at line 135 of file dislocationSource.cpp.

```
{
  return (this->tauCritical);
}
```

5.3.3.11 double Defect::getX ( ) [inherited]

Returns the X-coordinate of the defect.

Returns

X-coordinate of the defect.

Definition at line 147 of file defect.cpp.

```
{
  return (this->getValue (0));
}
```

5.3.3.12 double Defect::getY ( ) [inherited]

Returns the Y-coordinate of the defect.

Returns

Y-coordinate of the defect.

Definition at line 156 of file defect.cpp.

```
{
  return (this->pos.getValue (1));
}
```

5.3.3.13 double Defect::getZ ( ) [inherited]

Returns the Z-coordinate of the defect.

Returns

Z-coordinate of the defect.

Definition at line 165 of file defect.cpp.

```
{
  return (this->pos.getValue (2));
}
```

5.3.3.14 bool DislocationSource::ifEmitDipole ( )

Checks if the dislocation source has experienced higher than critical shear stress for long enough to emit a dipole.

The number of iterations for which the dislocation source must experience a shear stress higher than the critical value is given in the member nIterations. When the counter variable countIterations reaches this value, the source is ready to emit a dipole, so a true value is returned. In other cases, false is returned.

Returns

The boolean result of whether the count of iterations is greater than the limiting number of iterations provided at input.

Definition at line 192 of file dislocationSource.cpp.

```
{
  return ( this->countIterations >= this->nIterations );
}
```

5.3.3.15 void DislocationSource::incrementIterationCount ( )

Increments the variable countIterations by 1.

Definition at line 182 of file dislocationSource.cpp.

```
{
  this->countIterations++;
}
```

5.3.3.16 void DislocationSource::resetIterationCounter ( )

Sets the iteration counter to zero.

Definition at line 98 of file dislocationSource.cpp.

```
{
  this->countIterations = 0;
}
```

5.3.3.17 void DislocationSource::setBurgers (Vector3d *burgers*)

Sets the Burgers vector of the dislocation.

Parameters

burgers	Burgers vector of the dislocation.

Definition at line 54 of file dislocationSource.cpp.

```
{
  this->bvec = burgers;
}
```

5.3.3.18 void DislocationSource::setBurgersMagnitude (double *bm*)

Set the magnitude of the Burgers vector.

Parameters

bm	Magnitude of the Burgers vector.

Definition at line 72 of file dislocationSource.cpp.

```
{
  this->bmag = bm;
}
```

5.3.3.19 void DislocationSource::setLineVector (Vector3d *line*)

Sets the line vector of the dislocation.

Parameters

line	Line vector of the dislocation.

Definition at line 63 of file dislocationSource.cpp.

```
{
  this->lvec = line;
}
```

5.3.3.20 void DislocationSource::setNumIterations (int *nIter*)

Set the number of iterations before a dipole is emitted.

Parameters

nIter	Number of iterations spent at a high shear stress value before a dislocation dipole is emitted.

Definition at line 90 of file dislocationSource.cpp.

```
{
  this->nIterations = nIter;
}
```

5.3.3.21 void Defect::setPosition (double * *a*) [inherited]

Sets the position of the defect.

The position of the defect is set to the co-ordinates present in the array pointed to by the argument. Sets the position of the defect as the values in the array pointed to by the argument.

Parameters

a	Pointer to the array containing the coordinates of the defect.

Definition at line 56 of file defect.cpp.

```
{
  this->pos.setVector (a);
}
```

5.3.3.22 void Defect::setPosition (double *x*, double *y*, double *z*) [inherited]

Sets the position of the defect.

The position of the defect is set to the co-ordinates specified by the arguments (x, y, z). Sets the position of the defect as the coordinates provided as arguments.

Parameters

x	X-coordinate of the defect.
y	Y-coordinate of the defect.
z	Z-coordinate of the defect.

Definition at line 69 of file defect.cpp.

```
{
  this->pos.setValue (0, x);
  this->pos.setValue (1, y);
  this->pos.setValue (2, z);
}
```

5.3.3.23 void Defect::setPosition (Vector3d *a*) [inherited]

Sets the position of the defect.

The position of the defect is set to the position vector fiven by the argument a.

Parameters

a	Position vector of the defect.

Definition at line 81 of file defect.cpp.

```
{
  this->position = a;
}
```

5.3.3.24 void DislocationSource::setTauCritical (double *tauC*)

Set the critical shear stres for dipole emission.

Parameters

tauC	Critical shear stress for dipole emission.

Definition at line 81 of file dislocationSource.cpp.

```
{
  this->tauCritical = tauC;
}
```

5.3.3.25 void Defect::setX (double *x*) [inherited]

Sets the X-coordinate of the defect.

Parameters

x	X-coordinate of the defect.

Definition at line 90 of file defect.cpp.

```
{
  this->pos.setValue (0, x);
}
```

5.3.3.26 void Defect::setY (double *y*) [inherited]

Sets the Y-coordinate of the defect.

Parameters

y	Y-coordinate of the defect.

Definition at line 99 of file defect.cpp.

```
{
  this->pos.setValue (1, y);
}
```

5.3.3.27 void Defect::setZ (double *z*) [inherited]

Sets the Z-coordinate of the defect.

Parameters

z	Z-coordinate of the defect.

Definition at line 108 of file defect.cpp.

```
{
  this->pos.setValue (2, z);
}
```

5.3.3.28 virtual Stress Defect::stressField (Vector3d *p*, double *mu*, double *nu*) [inline], [virtual], [inherited]

Virtual function for calculating the stress field.

Returns the value of the stress field of the given defect at the position given by the argument. This is a virtual function and always returns a zero matrix. Classes which inherit this function should have their own implementations of this function to override its behaviour.

Parameters

p	Position vector of the the point where the stress field is to be calculated.
mu	Shear modulus in Pascals.
nu	Poisson's ratio.

Returns

Stress field value at the position p.

Reimplemented in Dislocation.

Definition at line 142 of file defect.h.

```
{
  // This virtual function returns a zero matrix.
  // Inheriting classes will have functions implementing this in their own way
  // They will override this behaviour.
  Stress s;
  return (s);
}
```

5.3.4 Field Documentation

5.3.4.1 double DislocationSource::bmag [protected]

Magnitude of the Burgers vector in metres.

The magnitude of the Burgers vector is useful for several calculations such as stress field around the dislocation.

Definition at line 45 of file dislocationSource.h.

5.3.4.2 Vector3d DislocationSource::bvec [protected]

Burgers vector of the dislocation.

Definition at line 28 of file dislocationSource.h.

5.3.4.3 int DislocationSource::countIterations [protected]

Counter variable for the number of consecutive iterations the dislocation source has experienced a shear stress greater than its critical value.

A dislocation source needs to experience a shear stress higher than a critical value, given by tauCritical, for a certain amount of time before it is triggered and it emits a dislocation dipole. This limiting number of iterations is given by the variable nIterations, and this variable countIterations is a counter variable. Once this limit is reached, a dipole is emitted and this counter variable is set to zero.

Definition at line 69 of file dislocationSource.h.

5.3.4.4 Vector3d DislocationSource::lvec [protected]

Line vector if the dislocation.

Definition at line 33 of file dislocationSource.h.

5.3.4.5 bool DislocationSource::mobile [protected]

Boolean term indicating mobility.

For mobile dislocations this term is true and for pinned dislocations it is false.

Definition at line 39 of file dislocationSource.h.

5.3.4.6 int DislocationSource::nIterations [protected]

Number of iterations before a dipole is emitted.

A dislocation dipole source needs to experience a certain critical level of shear stress for a certain amount of time before it can emit a dipole. The amount of time is represented instead by a number of iterations nIterations.

Definition at line 63 of file dislocationSource.h.

5.3.4.7 Vector3d Defect::pos [protected], [inherited]

Position vector of the defect in 2D space.

Definition at line 26 of file defect.h.

5.3.4.8 RotationMatrix DislocationSource::rotationMatrix [protected]

The rotation matrix for rotating from the global to the local co-ordinate system and vice-versa.

This is the rotation matrix that represents the relationship between the global and local co-ordinate systems. It is used to convert tensors and vectors between the two systems. The rotation matrix needs to be calculated once and may be refreshed periodically if lattice rotation is implemented. In the absence of lattice rotation, the matrix will remain invariant.

Definition at line 51 of file dislocationSource.h.

5.3.4.9 double DislocationSource::tauCritical [protected]

Critical stress for the emission of a dislocation dipole.

A dislocation dipole source needs to experience a certain critical level of shear stress for a certain amount of time before it can emit a dipole. This critical stress is given by tauCritical.

Definition at line 57 of file dislocationSource.h.

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

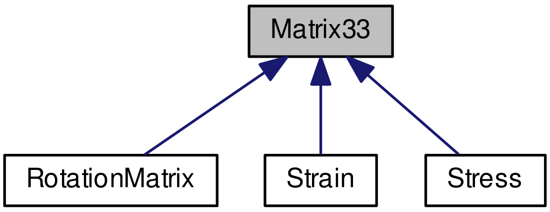
- dislocationSource.h
- dislocationSource.cpp

5.4 Matrix33 Class Reference

Matrix33 class representing a 3x3 square matrix.

`#include <matrix33.h>`

Inheritance diagram for Matrix33:



Public Member Functions

- Matrix33 ()

  Default constructor.
- Matrix33 (double **a)

  Constructor with the values provided in a 3x3 matrix.
- Matrix33 (Vector3d a)

  Constructor to create the matrix from the dyadic product of a vector with itself.
- Matrix33 (Vector3d a, Vector3d b)

  Constructor with the vectors, the product of which will result in the matrix.
- void setValue (int row, int column, double value)

  Function to set the value of an element indicated by its position.
- double getValue (int row, int column)

  Returns the value of the element located by the row and column indices provided.
- Matrix33 adjugate ()

  Returns the adjugate matrix of the present matrix.
- Matrix33 operator+ (const Matrix33 &) const

  Operator for addition of two matrices.
- void operator+= (const Matrix33 &)

  Operator for reflexive addition of two matrices.
- Matrix33 operator- (const Matrix33 &) const

  Operator for the subtraction of two matrices.
- void operator-= (const Matrix33 &)

  Operator for reflexive subtraction of two matrices.
- Matrix33 operator* (const double &) const

  Operator for scaling the matrix by a scalar.
- void operator*= (const double &)

  Operator for reflexive scaling of the matrix by a scalar.
- Matrix33 operator* (const Matrix33 &) const

  Operator for the multiplication of two matrices.
- void operator*= (const Matrix33 &)

  Operator for reflexive multiplication of two matrices.
- Vector3d operator* (const Vector3d &) const

  Operator for the multiplication of a matrix with a vector.
- Matrix33 operator^ () const

Transpose.

- double operator∼ () const

  Determinant.

- Matrix33 operator! () const

  Inverse.

Protected Attributes

- double x [3][3]

  Array containing the elements of the matrix.

5.4.1 Detailed Description

Matrix33 class representing a 3x3 square matrix.

This class represents a 3x3 square matrix. The member functions and operators define various operations that may be carried out on the matrix.

Definition at line 20 of file matrix33.h.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 Matrix33::Matrix33 ( )

Default constructor.

Initializes the matrix with all elements equal to 0.0.

Definition at line 17 of file matrix33.cpp.

```
{
  int i, j;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          this->x[i][j] = 0.0;
        }
    }
}
```

5.4.2.2 Matrix33::Matrix33 (double ∗∗ *a*)

Constructor with the values provided in a 3x3 matrix.

Populated the mstrix with data present in corresponding elements of the provided 3x3 array.

Parameters

a	Pointer to the two-dimensional 3x3 array.

Definition at line 35 of file matrix33.cpp.

```
{
  int i, j;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          this->x[i][j] = a[i][j];
```

```
        }
    }
}
```

5.4.2.3 Matrix33::Matrix33 (Vector3d *a*)

Constructor to create the matrix from the dyadic product of a vector with itself.

The matrix is created by performing the dyadic product of the provided vector with itself.

Parameters

a	The vector whose dyadic product results in the matrix.

Definition at line 53 of file matrix33.cpp.

```
{
  int i, j;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          this->x[i][j] = a.x[i] * a.x[j];
        }
    }
}
```

5.4.2.4 Matrix33::Matrix33 (Vector3d *a*, Vector3d *b*)

Constructor with the vectors, the product of which will result in the matrix.

The matrix is created from the product the first vector with the second.

Parameters

a	First vector.
b	Second vector.

Definition at line 72 of file matrix33.cpp.

```
{
  int i, j;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          this->x[i][j] = a.x[i] * b.x[j];
        }
    }
}
```

5.4.3 Member Function Documentation

5.4.3.1 Matrix33 Matrix33::adjugate ( )

Returns the adjugate matrix of the present matrix.

The adjugate matrix of the present matrix is returned. The adjugate matrix is calculated by evaluating the determinant of the cofactor matrix of each element, and then replacing the corresponding element position by the value of the determinant. This operation is useful in calculating the inverse of a matrix.

Returns

The adjugate matrix of the present matrix.

Definition at line 129 of file matrix33.cpp.

```
{
  Matrix33 adj;

  adj.x[0][0] = (this->x[1][1]*this->x[2][2]) - (this->x[1][2]*this->x[2][1]);
  adj.x[0][1] = (this->x[1][2]*this->x[2][0]) - (this->x[1][0]*this->x[2][2]);
  adj.x[0][2] = (this->x[1][0]*this->x[2][1]) - (this->x[1][1]*this->x[2][0]);

  adj.x[1][0] = (this->x[2][1]*this->x[0][2]) - (this->x[0][1]*this->x[2][2]);
  adj.x[1][1] = (this->x[2][2]*this->x[0][0]) - (this->x[2][0]*this->x[0][2]);
  adj.x[1][2] = (this->x[2][0]*this->x[0][1]) - (this->x[2][1]*this->x[0][0]);

  adj.x[2][0] = (this->x[0][1]*this->x[1][2]) - (this->x[0][2]*this->x[1][1]);
  adj.x[2][1] = (this->x[0][2]*this->x[1][0]) - (this->x[0][0]*this->x[1][2]);
  adj.x[2][2] = (this->x[0][0]*this->x[1][1]) - (this->x[1][0]*this->x[0][1]);

  return (adj);
}
```

5.4.3.2 double Matrix33::getValue (int *row,* int *column*)

Returns the value of the element located by the row and column indices provided.

The value of the element indicated by the arguments row and column is returned. The values of row and column must correspond to array indices, and thus can be one of 0, 1 and 2. In any other case 0.0 is returned.

Parameters

row	Row index of the element.
column	Column index of the element.

Returns

Value of the element located at the given position.

Definition at line 111 of file matrix33.cpp.

```
{
  if (row>=0 && row<3)
    {
      if (column>=0 && column<3)
        {
          return (this->x[row][column]);
        }
    }

  return (0.0);
}
```

5.4.3.3 Matrix33 Matrix33::operator! ( ) const

Inverse.

Returns in a new matrix the inverse of the current matrix. If the current matrix is non-invertible, a zero matrix is returned.

Returns

Matrix with the inverse of the current matrix. If the current matrix is non-invertible, a zero matrix is returned.

Definition at line 370 of file matrix33.cpp.

```
{
  Matrix33 r;    // Result matrix

  double determinant = ~(*this);

  if (determinant == 0.0)
    {
      // The matrix is non-invertible
      return (r);        // Zero matrix
    }

  // If we are still here, the matrix is invertible

  //  Transpose
  Matrix33 tr = ^(*this);

  // Find Adjugate matrix
  Matrix33 adj = tr.adjugate();

  // Calculate the inverse by dividing the adjugate matrix by the determinant
  r = adj * (1.0/determinant);

  return (r);
}
```

5.4.3.4 Matrix33 Matrix33::operator* (const double & *p*) const

Operator for scaling the matrix by a scalar.

Scales the current matrix by the scalar provided and returns the result in a third matrix.

Returns

Matrix containing the result of scaling the current matrix by the scalar provided as argument.

Definition at line 233 of file matrix33.cpp.

```
{
  int i, j;
  Matrix33 r;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          r.x[i][j] = this->x[i][j] * p;
        }
    }

  return (r);
}
```

5.4.3.5 Matrix33 Matrix33::operator* (const Matrix33 & *p*) const

Operator for the multiplication of two matrices.

Multiplies the current matrix with another 3x3 matrix and returns the result in a new matrix.

Returns

The result of the multiplication of the current matrix with the one provided as argument.

Definition at line 271 of file matrix33.cpp.

```
{
  int i, j, k;
  Matrix33 r;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
```

```
        {
          r.x[i][j] = 0.0;
          for (k=0; k<3; k++)
            {
              r.x[i][j] += this->x[i][k] * p.x[k][j];
            }
        }
    }

  return (r);
}
```

5.4.3.6 Vector3d Matrix33::operator∗ (const Vector3d & *v*) const

Operator for the multiplication of a matrix with a vector.

Returns in a vector the result of the multiplication of the current matrix with the provided vector.

Returns

The vector resulting from the multiplication of the current matrix with a vector.

Definition at line 311 of file matrix33.cpp.

```
{
  Vector3d r(0.0, 0.0, 0.0);
  int i, j;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          r[i] += this->x[i][j] * v.x[j];
        }
    }

  return (r);
}
```

5.4.3.7 void Matrix33::operator∗= (const double & *p*)

Operator for reflexive scaling of the matrix by a scalar.

Scales the current matrix by the scalar provided and populates the current matrix elements with the result.

Definition at line 253 of file matrix33.cpp.

```
{
  int i, j;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          this->x[i][j] *= p;
        }
    }
}
```

5.4.3.8 void Matrix33::operator∗= (const Matrix33 & *p*)

Operator for reflexive multiplication of two matrices.

Multiplies the current matrix with another 3x3 matrix and populates the elements of the current matrix with the result.

Definition at line 295 of file matrix33.cpp.

```
{
  Matrix33* r = new Matrix33;

  *r = (*this) * p;
  *this = *r;

  delete(r);
  r = NULL;
}
```

5.4.3.9 Matrix33 Matrix33::operator+ (const Matrix33 & *p*) const

Operator for addition of two matrices.

Adds the current matrix to the provided matrix and returns a third matrix with the result.

Returns

Matrix containing the sum of the present matrix and the one provided.

Definition at line 155 of file matrix33.cpp.

```
{
  int i, j;
  Matrix33 r;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          r.x[i][j] = this->x[i][j] + p.x[i][j];
        }
    }

  return (r);
}
```

5.4.3.10 void Matrix33::operator+= (const Matrix33 & *p*)

Operator for reflexive addition of two matrices.

Adds the current matrix to the provided matrix and populates the current matrix elements with the result.

Definition at line 175 of file matrix33.cpp.

```
{
  int i, j;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          this->x[i][j] += p.x[i][j];
        }
    }
}
```

5.4.3.11 Matrix33 Matrix33::operator- (const Matrix33 & *p*) const

Operator for the subtraction of two matrices.

Subtracts the given matrix from the current matrix and returns the result in a new matrix.

Returns

Matrix containing the result of subtracting the provided matrix from the current matrix.

Definition at line 194 of file matrix33.cpp.

```
{
  int i, j;
  Matrix33 r;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          r.x[i][j] = this->x[i][j] - p.x[i][j];
        }
    }

  return (r);
}
```

5.4.3.12 void Matrix33::operator-= (const Matrix33 & *p*)

Operator for reflexive subtraction of two matrices.

Subtracts the given matrix from the current matrix and populates the current matrix with the result.

Definition at line 214 of file matrix33.cpp.

```
{
  int i, j;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          this->x[i][j] -= p.x[i][j];
        }
    }
}
```

5.4.3.13 Matrix33 Matrix33::operator^ ( ) const

Transpose.

Performs the transpose of the current matrix.

Returns

A new matrix with the transpose of the current matrix.

Definition at line 333 of file matrix33.cpp.

```
{
  Matrix33 r;
  int i, j;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          r.x[i][j] = this->x[j][i];
        }
    }

  return (r);
}
```

5.4.3.14 double Matrix33::operator∼ ( ) const

Determinant.

Calculates the determinant of the current matrix.

Returns

Returns the determinant of the current matrix.

Definition at line 354 of file matrix33.cpp.

```
{
  double d = 0.0;

  d += this.x[0][0] * ( (this->x[1][1]*this->x[2][2]) - (this->x[2][1]*this->x[1][2]) );
  d += this.x[0][1] * ( (this->x[1][2]*this->x[2][0]) - (this->x[1][0]*this->x[2][2]) );
  d += this.x[0][2] * ( (this->x[1][0]*this->x[2][1]) - (this->x[2][0]*this->x[1][1]) );

  return (d);
}
```

5.4.3.15 void Matrix33::setValue (int *row*, int *column*, double *value*)

Function to set the value of an element indicated by its position.

The element indicated by the arguments row and column is set to the value provided. The values of row and column must correspond to array indices, and thus can be one of 0, 1 and 2. In any other case 0.0 is returned.

Parameters

row	Row index of the element.
column	Column index of the element.
value	Value that the element is to be set to.

Definition at line 93 of file matrix33.cpp.

```
{
  if (row>=0 && row<3)
    {
      if (column>=0 && column<3)
        {
          this->x[row][column] = value;
        }
    }
}
```

5.4.4 Field Documentation

5.4.4.1 double Matrix33::x[3][3] [protected]

Array containing the elements of the matrix.

Definition at line 26 of file matrix33.h.

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

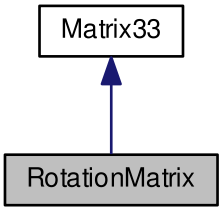
- matrix33.h
- matrix33.cpp

5.5 RotationMatrix Class Reference

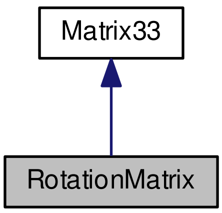
RotationMatrix class to represent a rotation matrix.

`#include <rotationMatrix.h>`

Inheritance diagram for RotationMatrix:



Collaboration diagram for RotationMatrix:



Public Member Functions

- RotationMatrix ()

  Default constructor.
- RotationMatrix (Vector3d ∗unPrimed, Vector3d ∗primed)

  Defines the rotation matrix based on two co-ordinate systems.
- void setValue (int row, int column, double value)

  Function to set the value of an element indicated by its position.
- double getValue (int row, int column)

  Returns the value of the element located by the row and column indices provided.
- Matrix33 adjugate ()

  Returns the adjugate matrix of the present matrix.
- Matrix33 operator+ (const Matrix33 &) const

  Operator for addition of two matrices.
- void operator+= (const Matrix33 &)

  Operator for reflexive addition of two matrices.

- Matrix33 operator- (const Matrix33 &) const

  Operator for the subtraction of two matrices.
- void operator-= (const Matrix33 &)

  Operator for reflexive subtraction of two matrices.
- Matrix33 operator∗ (const double &) const

  Operator for scaling the matrix by a scalar.
- Matrix33 operator∗ (const Matrix33 &) const

  Operator for the multiplication of two matrices.
- Vector3d operator∗ (const Vector3d &) const

  Operator for the multiplication of a matrix with a vector.
- void operator∗= (const double &)

  Operator for reflexive scaling of the matrix by a scalar.
- void operator∗= (const Matrix33 &)

  Operator for reflexive multiplication of two matrices.
- Matrix33 operator^ () const

  Transpose.
- double operator∼ () const

  Determinant.
- Matrix33 operator! () const

  Inverse.

Protected Attributes

- double x [3][3]

  Array containing the elements of the matrix.

5.5.1 Detailed Description

RotationMatrix class to represent a rotation matrix.

The member functions of this class create a rotation matrix for carrying out rotations in 3D and transformation of axes.

Definition at line 19 of file rotationMatrix.h.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 RotationMatrix::RotationMatrix ( )

Default constructor.

Initializes the rotation matrix with a unit matrix.

Definition at line 14 of file rotationMatrix.cpp.

```
{
    int i,  j;

    for ( i=0; i<3; i++ ) {
        for ( j=0; i<3; j++ ) {
            if ( i==j ) {
                this->setValue ( i, j, 1.0 );
            }
            else {
                this->setValue ( i, j, 0.0 );
            }
        }
    }
}
```

5.5.2.2 RotationMatrix::RotationMatrix (Vector3d * *unPrimed,* Vector3d * *primed*)

Defines the rotation matrix based on two co-ordinate systems.

The rotation matrix is created using the axes of the two co-ordinate systems provided as arguments. The vectors must be normalized to be unit vectors.

Parameters

unPrimed	Pointer to the array containing the three axes vectors of the unprimed (old) system.
primed	Pointer to the array containing the three axes vectors of the primed (new) system.

Definition at line 36 of file rotationMatrix.cpp.

```
{
    int i, j;

    for ( i=0; i<3; i++ ) {
        for ( j=0; j<3; j++ ) {
            this->setValue ( i, j, primed[i]*unPrimed[j] );
        }
    }
}
```

5.5.3 Member Function Documentation

5.5.3.1 Matrix33 Matrix33::adjugate ( ) [inherited]

Returns the adjugate matrix of the present matrix.

The adjugate matrix of the present matrix is returned. The adjugate matrix is calculated by evaluating the determinant of the cofactor matrix of each element, and then replacing the corresponding element position by the value of the determinant. This operation is useful in calculating the inverse of a matrix.

Returns

The adjugate matrix of the present matrix.

Definition at line 129 of file matrix33.cpp.

```
{
  Matrix33 adj;

  adj.x[0][0] = (this->x[1][1]*this->x[2][2]) - (this->x[1][2]*this->x[2][1]);
  adj.x[0][1] = (this->x[1][2]*this->x[2][0]) - (this->x[1][0]*this->x[2][2]);
  adj.x[0][2] = (this->x[1][0]*this->x[2][1]) - (this->x[1][1]*this->x[2][0]);

  adj.x[1][0] = (this->x[2][1]*this->x[0][2]) - (this->x[0][1]*this->x[2][2]);
  adj.x[1][1] = (this->x[2][2]*this->x[0][0]) - (this->x[2][0]*this->x[0][2]);
  adj.x[1][2] = (this->x[2][0]*this->x[0][1]) - (this->x[2][1]*this->x[0][0]);

  adj.x[2][0] = (this->x[0][1]*this->x[1][2]) - (this->x[0][2]*this->x[1][1]);
  adj.x[2][1] = (this->x[0][2]*this->x[1][0]) - (this->x[0][0]*this->x[1][2]);
  adj.x[2][2] = (this->x[0][0]*this->x[1][1]) - (this->x[1][0]*this->x[0][1]);

  return (adj);
}
```

5.5.3.2 double Matrix33::getValue (int *row,* int *column*) [inherited]

Returns the value of the element located by the row and column indices provided.

The value of the element indicated by the arguments row and column is returned. The values of row and column must correspond to array indices, and thus can be one of 0, 1 and 2. In any other case 0.0 is returned.

Parameters

row	Row index of the element.
column	Column index of the element.

Returns

Value of the element located at the given position.

Definition at line 111 of file matrix33.cpp.

```
{
  if (row>=0 && row<3)
    {
      if (column>=0 && column<3)
        {
          return (this->x[row][column]);
        }
    }

  return (0.0);
}
```

5.5.3.3 Matrix33 Matrix33::operator! ( ) const [inherited]

Inverse.

Returns in a new matrix the inverse of the current matrix. If the current matrix is non-invertible, a zero matrix is returned.

Returns

Matrix with the inverse of the current matrix. If the current matrix is non-invertible, a zero matrix is returned.

Definition at line 370 of file matrix33.cpp.

```
{
  Matrix33 r;    // Result matrix

  double determinant = ~(*this);

  if (determinant == 0.0)
    {
      // The matrix is non-invertible
      return (r);        // Zero matrix
    }

  // If we are still here, the matrix is invertible

  //  Transpose
  Matrix33 tr = ^(*this);

  // Find Adjugate matrix
  Matrix33 adj = tr.adjugate();

  // Calculate the inverse by dividing the adjugate matrix by the determinant
  r = adj * (1.0/determinant);

  return (r);
}
```

5.5.3.4 Matrix33 Matrix33::operator* (const double & *p*) const [inherited]

Operator for scaling the matrix by a scalar.

Scales the current matrix by the scalar provided and returns the result in a third matrix.

Returns

Matrix containing the result of scaling the current matrix by the scalar provided as argument.

Definition at line 233 of file matrix33.cpp.

```
{
  int i, j;
  Matrix33 r;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          r.x[i][j] = this->x[i][j] * p;
        }
    }

  return (r);
}
```

5.5.3.5 Matrix33 Matrix33::operator* (const Matrix33 & *p*) const [inherited]

Operator for the multiplication of two matrices.

Multiplies the current matrix with another 3x3 matrix and returns the result in a new matrix.

Returns

The result of the multiplication of the current matrix with the one provided as argument.

Definition at line 271 of file matrix33.cpp.

```
{
  int i, j, k;
  Matrix33 r;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          r.x[i][j] = 0.0;
          for (k=0; k<3; k++)
            {
              r.x[i][j] += this->x[i][k] * p.x[k][j];
            }
        }
    }

  return (r);
}
```

5.5.3.6 Vector3d Matrix33::operator* (const Vector3d & *v*) const [inherited]

Operator for the multiplication of a matrix with a vector.

Returns in a vector the result of the multiplication of the current matrix with the provided vector.

Returns

The vector resulting from the multiplication of the current matrix with a vector.

Definition at line 311 of file matrix33.cpp.

```
{
  Vector3d r(0.0, 0.0, 0.0);
  int i, j;

  for (i=0; i<3; i++)
```

```
    {
      for (j=0; j<3; j++)
        {
          r[i] += this->x[i][j] * v.x[j];
        }
    }

  return (r);
}
```

5.5.3.7 void Matrix33::operator*= (const double & *p*) [inherited]

Operator for reflexive scaling of the matrix by a scalar.

Scales the current matrix by the scalar provided and populates the current matrix elements with the result.

Definition at line 253 of file matrix33.cpp.

```
{
  int i, j;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          this->x[i][j] *= p;
        }
    }
}
```

5.5.3.8 void Matrix33::operator*= (const Matrix33 & *p*) [inherited]

Operator for reflexive multiplication of two matrices.

Multiplies the current matrix with another 3x3 matrix and populates the elements of the current matrix with the result.

Definition at line 295 of file matrix33.cpp.

```
{
  Matrix33* r = new Matrix33;

  *r = (*this) * p;
  *this = *r;

  delete(r);
  r = NULL;
}
```

5.5.3.9 Matrix33 Matrix33::operator+ (const Matrix33 & *p*) const [inherited]

Operator for addition of two matrices.

Adds the current matrix to the provided matrix and returns a third matrix with the result.

Returns

Matrix containing the sum of the present matrix and the one provided.

Definition at line 155 of file matrix33.cpp.

```
{
  int i, j;
  Matrix33 r;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
```

```
          r.x[i][j] = this->x[i][j] + p.x[i][j];
        }
    }

  return (r);
}
```

5.5.3.10 void Matrix33::operator+= (const Matrix33 & *p*) [inherited]

Operator for reflexive addition of two matrices.

Adds the current matrix to the provided matrix and populates the current matrix elements with the result.

Definition at line 175 of file matrix33.cpp.

```
{
  int i, j;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          this->x[i][j] += p.x[i][j];
        }
    }
}
```

5.5.3.11 Matrix33 Matrix33::operator- (const Matrix33 & *p*) const [inherited]

Operator for the subtraction of two matrices.

Subtracts the given matrix from the current matrix and returns the result in a new matrix.

Returns

Matrix containing the result of subtracting the provided matrix from the current matrix.

Definition at line 194 of file matrix33.cpp.

```
{
  int i, j;
  Matrix33 r;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          r.x[i][j] = this->x[i][j] - p.x[i][j];
        }
    }

  return (r);
}
```

5.5.3.12 void Matrix33::operator-= (const Matrix33 & *p*) [inherited]

Operator for reflexive subtraction of two matrices.

Subtracts the given matrix from the current matrix and populates the current matrix with the result.

Definition at line 214 of file matrix33.cpp.

```
{
  int i, j;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
```

```
        {
          this->x[i][j] -= p.x[i][j];
        }
    }
}
```

5.5.3.13 Matrix33 Matrix33::operator^ ( ) const [inherited]

Transpose.

Performs the transpose of the current matrix.

Returns

A new matrix with the transpose of the current matrix.

Definition at line 333 of file matrix33.cpp.

```
{
  Matrix33 r;
  int i, j;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          r.x[i][j] = this->x[j][i];
        }
    }

  return (r);
}
```

5.5.3.14 double Matrix33::operator~ ( ) const [inherited]

Determinant.

Calculates the determinant of the current matrix.

Returns

Returns the determinant of the current matrix.

Definition at line 354 of file matrix33.cpp.

```
{
  double d = 0.0;

  d += this.x[0][0] * ( (this->x[1][1]*this->x[2][2]) - (this->x[2][1]*this->x[1][2]) );
  d += this.x[0][1] * ( (this->x[1][2]*this->x[2][0]) - (this->x[1][0]*this->x[2][2]) );
  d += this.x[0][2] * ( (this->x[1][0]*this->x[2][1]) - (this->x[2][0]*this->x[1][1]) );

  return (d);
}
```

5.5.3.15 void Matrix33::setValue (int *row*, int *column*, double *value*) [inherited]

Function to set the value of an element indicated by its position.

The element indicated by the arguments row and column is set to the value provided. The values of row and column must correspond to array indices, and thus can be one of 0, 1 and 2. In any other case 0.0 is returned.

Parameters

row	Row index of the element.
column	Column index of the element.
value	Value that the element is to be set to.

Definition at line 93 of file matrix33.cpp.

```
{
  if (row>=0 && row<3)
    {
      if (column>=0 && column<3)
        {
          this->x[row][column] = value;
        }
    }
}
```

5.5.4 Field Documentation

5.5.4.1 double Matrix33::x[3][3] [protected], [inherited]

Array containing the elements of the matrix.

Definition at line 26 of file matrix33.h.

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

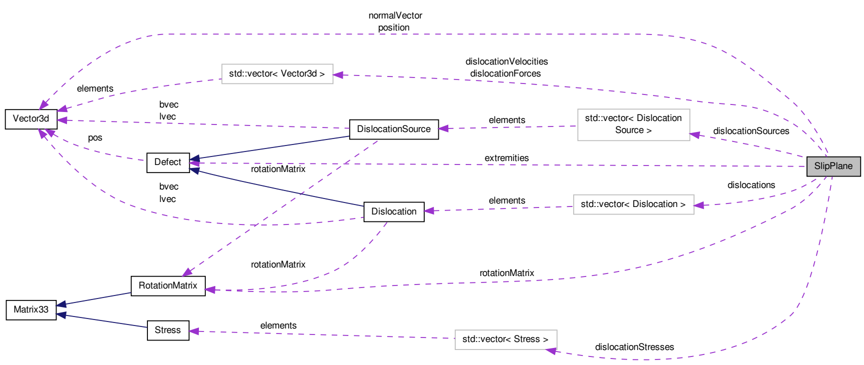
- rotationMatrix.h
- rotationMatrix.cpp

5.6 SlipPlane Class Reference

SlipPlane class representing a slip plane in the simulation.

`#include <slipPlane.h>`

Collaboration diagram for SlipPlane:



Public Member Functions

- SlipPlane ()

  Default constructor.
- SlipPlane (Vector3d *ends, Vector3d normal, Vector3d pos, std::vector< Dislocation > dislocationList, std::vector< DislocationSource > dislocationSourceList)

Constructor that specifies all members explicitly.
- void setExtremities (Vector3d ∗ends)

  Set the extremities of the slip plane.
- void setNormal (Vector3d normal)

  Set the normal vector of the slip plane.
- void setPosition (Vector3d pos)

  Set the position of the slip plane.
- void setDislocationList (std::vector< Dislocation > dislocationList)

  Set the list of dislocations of the slip plane.
- void setDislocationSourceList (std::vector< DislocationSource > dislocationSourceList)

  Set the list of dislocation sources on the slip plane.
- Vector3d getExtremity (int i)

  Get the position vector of the extremity whose index is provided as argument.
- Defect ∗ getExtremities ()

  Get the position vectors of the extremities of the slip plane.
- Vector3d getNormal ()

  Get the normal vector of the slip plane.
- Vector3d getPosition ()

  Get the position vector of the slip plane.
- bool getDislocation (int i, Dislocation ∗d)

  Get the dislocation on the slip plane indicated by the index provided as argument.
- std::vector< Dislocation > getDislocationList ()

  Get the entire vector container which holds the dislocations lying on this slip plane.
- int getNumDislocations ()

  Get the number of dislocations.
- bool getDislocationSource (int i, DislocationSource ∗dSource)

  Get the dislocation source on the slip plane indicated by the index provided as argument.
- int getNumDislocationSources ()

  Get the number of dislocation sources.
- std::vector< DislocationSource > getDislocationSourceList ()

  Get the entire vector container which holds the dislocation sources lying on this slip plane.
- RotationMatrix getRotationMatrix ()

  Get the rotation matrix for this slip plane.
- Vector3d getAxis (int i)

  Get the axis (expressed in the global co-ordinate system) of the slip plane's local co-ordinate system, as indicated by the argument. (0, 1, 2)=(x, y, z).
- void calculateRotationMatrix ()

  Calculates the rotation matrix for this slip plane.
- void calculateDislocationStresses (Stress appliedStress, double mu, double nu)

  Calculates the total stress field experienced by each dislocation and stored it in the STL vector container dislocation-Stresses.
- void calculateDislocationForces (double tau_crss)

  This function populates the STL vector container dislocationForces with the Peach-Koehler force experienced by each dislocation.
- void calculateVelocities (double B)

  Calculates the velocities of dislocations and stores them in the std::vector container velocities.
- void calculateTimeIncrement (double minDistance, double minDt)

  Calculate the time increment based on the velocities of the dislocations.
- void moveDislocations ()

  Displaces the dislocations according to their velocities and the time increment.
- double distanceFromExtremity (Vector3d pos, int n)

  The distance of the point pos from the n^th extremity is returned.
- void sortDislocations ()

  Sorts the dislocations present on the slip plane in the ascending order of distance from the first extremity.

Protected Attributes

- Defect extremities [2]

  The extremities of the slip plane.
- Vector3d normalVector

  The normal vector to the slip plane.
- Vector3d position

  The position vector of the slip plane.
- std::vector< Dislocation > dislocations

  STL vector container with dislocations.
- std::vector< Stress > dislocationStresses

  STL vector container with the stress fields of dislocations.
- std::vector< Vector3d > dislocationForces

  The Peach-Koehler force experienced by each dislocation.
- std::vector< Vector3d > dislocationVelocities

  STL vector container with dislocation velocities.
- std::vector< DislocationSource > dislocationSources

  STL vector container with dislocation sources.
- double dt

  Time increment for the slip plane.
- RotationMatrix rotationMatrix

  Rotation matrix for co-ordinate system transformations.

5.6.1 Detailed Description

SlipPlane class representing a slip plane in the simulation.

This is the definition of the class SlipPlane. It represents a slip plane in the simulation. A slip plane is considered to be a collection of defects, such as dislocations and dislocation sources. In these simulations in two dimensions, the slip plane becomes a straight line. Its attributes are: position vectors of the extremities, normal vector (since we are concerned with the cubic system here, the normal vector's indices are the same as those of the plane), and the collection of defects.

Definition at line 29 of file slipPlane.h.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 SlipPlane::SlipPlane ( )

Default constructor.

The slip plane is initialized with default parameters specified in the file slipPlaneDefaults.h.

Definition at line 17 of file slipPlane.cpp.

```
{
  // Initialize the default variables.
  Vector3d pos(DEFAULT_SLIPPLANE_POSITION_0,
               DEFAULT_SLIPPLANE_POSITION_1,
               DEFAULT_SLIPPLANE_POSITION_2);
  Vector3d normal(DEFAULT_SLIPPLANE_NORMALVECTOR_0,
                  DEFAULT_SLIPPLANE_NORMALVECTOR_1,
                  DEFAULT_SLIPPLANE_NORMALVECTOR_2);
  Vector3d ends[2];
  ends[0] = Vector3d(DEFAULT_SLIPPLANE_EXTREMITY1_0,
                     DEFAULT_SLIPPLANE_EXTREMITY1_1,
                     DEFAULT_SLIPPLANE_EXTREMITY1_2);
  ends[1] = Vector3d(DEFAULT_SLIPPLANE_EXTREMITY2_0,
                     DEFAULT_SLIPPLANE_EXTREMITY2_1,
                     DEFAULT_SLIPPLANE_EXTREMITY2_2);
  std::vector<Dislocation> dislocationList(1, Dislocation());
```

```
  std::vector<DislocationSource> dislocationSourceList(1, DislocationSource());

  *this = SlipPlane(ends, normal, pos, dislocationList, dislocationSourceList);
}
```

5.6.2.2 SlipPlane::SlipPlane (Vector3d * *ends*, Vector3d *normal*, Vector3d *pos*, std::vector< Dislocation > *dislocationList*, std::vector< DislocationSource > *dislocationSourceList*)

Constructor that specifies all members explicitly.

The slip plane is initialized with parameters specified in the arguments.

Parameters

ends	Pointer to an array of type Vector3d, containing the position vectors of the extremities of the slip plane in consecutive locations.
normal	The normal vector of the slip plane.
pos	The position vector of the slip plane. (This parameter is useful for locating the slip plane within a slip system)
dislocationList	A vector container of type Dislocation containing the dislocations lying on this slip plane.
dislocation-SourceList	A vector container of type DislocationSource containing the dislocation sources lying on this slip plane.

Definition at line 48 of file slipPlane.cpp.

```
{
  this->setExtremities (ends);
  this->setNormal (normal);
  this->setPosition (pos);
  this->setDislocationList (dislocationList);
  this->setDislocationSourceList (dislocationSourceList);

  // Fill the vectors and stresses with zero vectors and stresses
  int nDisl = this->getNumDislocations ();
  this->dislocationStresses.resize(nDisl, Stress ());
  this->dislocationVelocities.resize(nDisl, Vector3d());
  this->dislocationForces.resize(nDisl, Vector3d());

  // Time increment
  this->dt = 0;

  this->calculateRotationMatrix ();
}
```

5.6.3 Member Function Documentation

5.6.3.1 void SlipPlane::calculateDislocationForces (double *tau_crss*)

This function populates the STL vector container dislocationForces with the Peach-Koehler force experienced by each dislocation.

This function calculates the Peach-Koehler force experienced by each dislocation using the function Dislocation::forcePeachKoehler and the STL vector SlipPlane::dislocationStresses. The argument tau_crss is the Critical Resolved Shear Stress in Pa.

Parameters

tau_crss	Critical Resolved Shear Stress in Pa.

Definition at line 354 of file slipPlane.cpp.

```
{
  std::vector<Dislocation>::iterator d;  // Iterator for dislocations
  std::vector<Vector3d>::iterator f;     // Iterator for forces
  std::vector<Stress>::iterator s;       // Iterator for stresses
```

```

    s = this->dislocationStresses.begin();
    f = this->dislocationForces.begin();

    for (d = this->dislocations.begin(); d!=this->dislcoations.end(); d++)
      {
        f = d->forcePeachKoehler (*s, tau_crss);
        s++;
        f++;
      }
  }
```

5.6.3.2 void SlipPlane::calculateDislocationStresses (Stress *appliedStress*, double *mu*, double *nu*)

Calculates the total stress field experienced by each dislocation and stored it in the STL vector container dislocation-Stresses.

The total stress field is calculated as a superposition of the applied stress field and the stress fields experienced by each dislocation due to every other dislocation in the simulation.

Parameters

appliedStress	The stress applied externally.
mu	Shear modulus of the material.
nu	Poisson's ratio.

Definition at line 322 of file slipPlane.cpp.

```
  {
    std::vector<Dislocation>::iterator d1;  // Iterator for each dislocation
    std::vector<Dislocation>::iterator d2;  // Nested iterator
    std::vector<Stress>::iterator s;        // Iterator for the Stress

    Vector3d p;                             // Position vector

    s = this->dislocationStresses.begin();
    for (d1=this->dislocations.begin(); d1!=this->dislocations.end(); d1++)
      {
        *s = appliedStress;
        p = d1->getPosition();
        for (d2 = this->dislocations.begin(); d2!=this->dislocations.end(); d2++)
          {
            if (d1==d2)
              {
                continue;
              }
            else
              {
                *s = *s + d2->stressField(p, mu, nu);
              }
          }
      }
  }
```

5.6.3.3 void SlipPlane::calculateRotationMatrix ( )

Calculates the rotation matrix for this slip plane.

The slip plane has a local co-ordinate system whose axes are the following: z-axis||normal vector and x-axis||slip plane vector (vector joining the extremities). The rotation matrix is calculated in order to carry out transformations between the global and local co-ordinate systems.

Definition at line 289 of file slipPlane.cpp.

```
{
  Vector3d *unPrimed = new Vector3d[3]; // Old system (global)
  Vector3d *primed   = new Vector3d[3]; // New system (local)

  int i, j;

```

```
  // Prepare the global and local systems
  for (i=0; i<3; i++)
  {
    for (j=0; j<3; j++)
    {
      unPrimed[i].setValue(j, (double)(i==j));
    }

    primed[i] = this->getAxis(i);
  }

  // Calculate the rotationMatrix
  this->rotationMatrix = RotationMatrix(unPrimed, primed);

  // Free memory
  delete(unPrimed);     unPrimed = NULL;
  delete(primed);       primed = NULL;
}
```

5.6.3.4 void SlipPlane::calculateTimeIncrement (double *minDistance,* double *minDt*)

Calculate the time increment based on the velocities of the dislocations.

In order to avoid the collision of dislocations with similar sign of Burgers vector, it is important to specify a minimum distance of approach between dislocations. When a dislocation reaches this limit, it is pinned. The velocities of the dislocations all being different, a time increment needs to be evaluated, which will limit the distance traveled by the dislocations in a given iteration.

Parameters

minDistance	Minimum distance of approach between dislocations having Burgers vectors of the same sign.
minDt	The smallest time step permissible. Dislocations having time steps smaller than this are made immobile for the present iteration.

Definition at line 408 of file slipPlane.cpp.

```
{
  // Get the number of dislocations
  int nDisl = this->dislocations.size();

  // Vector of time increments
  std::vector<double> timeIncrement(nDisl, 1000.0);

  // Position vectors
  Vector3d p0, p1;
  double norm_p01;

  // Velocity vectors
  Vector3d v0, v1;
  double norm_v01;

  int i;          // Counter for the loop
  double t1, t2;
  double dtMin;   // Minimum time increment

  // For the first dislocation, the time increment has to be calculated
  // for approach to both a dislocation and the slip plane extremity.
  // Time for slip plane extremity
  t1 = this->dislocations[0].idealTimeIncrement(this->dislocationVelocities[0],
                                               minDistance,
                                               this->extremity[0],
                                               Vector3d(0.0, 0.0, 0.0));
  t2 = this->dislocations[0].idealTimeIncrement(this->dislocationVelocities[0],
                                               minDistance,
                                               this->dislocations[1],
                                               this->dislocationVelocities[1]);
  // Choose the smaller of the two
  timeIncrement[0] = t1 < t2 ? t1:t2;
  if (timeIncrement[0] < minDt)
    {
      // This dislocation should not move in this iteration because it might collide with the next defect
      timeIncrement[0] = minDt;
      this->dislocationVelocities[0] = Vector3d(0.0, 0.0, 0.0);

      // The other defect is a slip plane extremity
```

```
      // This dislocation will not move any more
      this->dislocations[0].setPinned();
    }

  for (i=1; i<(nDisl-1); i++)
    {
      t1 = this->dislocations[i].idealTimeIncrement(this->dislocationVelocities[i],
                                                     minDistance,
                                                     this->dislocations[i-1],
                                                     this->dislcoationVelocities[i-1]);
      t2 = this->dislocations[i].idealTimeIncrement(this->dislocationVelocities[i],
                                                     minDistance,
                                                     this->dislocations[i+1],
                                                     this->dislcoationVelocities[i+1]);
      timeIncrement[i] = t1 < t2 ? t1:t2;

      if (timeIncrement[i] < minDt)
        {
          // This dislocation should not move in this iteration because it might collide with the next defect
          timeIncrement[i] = minDt;
          this->dislocationVelocities[i] = Vector3d(0.0, 0.0, 0.0);
        }
    }

  // For the last dislocation, the time increment has to be calculated
  // for approach to both a dislocation and the slip plane extremity.
  // Time for slip plane extremity
  i=nDisl-1;
  t1 = this->dislocations[i].idealTimeIncrement(this->dislocationVelocities[i],
                                                 minDistance,
                                                 this->extremity[1],
                                                 Vector3d(0.0, 0.0, 0.0));
  t2 = this->dislocations[i].idealTimeIncrement(this->dislocationVelocities[i],
                                                 minDistance,
                                                 this->dislocations[i-1],
                                                 this->dislocationVelocities[i-1]);
  // Choose the smaller of the two
  timeIncrement[i] = t1 < t2 ? t1:t2;

  if (timeIncrement[i] < minDt)
    {
      // This dislocation should not move in this iteration because it might collide with the next defect
      timeIncrement[i] = minDt;
      this->dislocationVelocities[i] = Vector3d(0.0, 0.0, 0.0);

      // The other defect is a slip plane extremity
      // This dislocation will not move any more
      this->dislocations[i].setPinned();
    }

  dtMin = 1000;
  for (i=0; i<nDisl; i++)
    {
      if (timeIncrement[i] < dtMin)
        {
          dtMin = timeIncrement[i];
        }
    }

  this->dt = dtMin;
}
```

5.6.3.5 void SlipPlane::calculateVelocities (double *B*)

Calculates the velocities of dislocations and stores them in the std::vector container velocities.

The velocities of the dislocations are calculated and stored in the std::vector container called velocities. The velocities are calculated using the proportionality law between them and the Peach-Koehler force, using the drag coefficient B as the constant of proportionality. param B The drag coefficient.

Definition at line 376 of file slipPlane.cpp.

```
{
  std::vector<Dislocation>::iterator d;  // Iterator for dislocations
  std::vector<Vector3d>::iterator f;     // Iterator for forces
  std::vector<Vector3d>::iterator v;     // Iterator for velocities
```

```
  d = this->dislocations.begin();
  f = this->dislocationForces.begin();
  v = this->dislocationVelocities.begin();

  while (v != this->dislocationVelocities.end())
    {
      if (d->isMobile())
        {
          *v = (*f)/B;
        }
      else
        {
          *v = Vector3d(0.0, 0.0, 0.0);
        }
      d++;
      f++;
      v++;
    }
}
```

5.6.3.6 double SlipPlane::distanceFromExtremity (Vector3d *pos*, int *n*)

The distance of the point pos from the n^th extremity is returned.

Parameters

pos	Position vector of the point whose distance is to be calculated.
n	Index of the extremity. Can be only 0 or 1. In all other cases 0.0 is returned.

Returns

Distance of the point pos from the n^th extremity of the slip plane.

Definition at line 539 of file slipPlane.cpp.

```
{
  if (n!=0 && n!=1)
    {
      return (0.0);
    }

  Vector3d r = this->extremities[n].getPosition();
  return ( (r-pos).magnitude() );
}
```

5.6.3.7 Vector3d SlipPlane::getAxis (int *i*)

Get the axis (expressed in the global co-ordinate system) of the slip plane's local co-ordinate system, as indicated by the argument. (0, 1, 2)=(x, y, z).

Parameters

i	Index of the axis that is to be returned. (0, 1, 2)=(x, y, z).

Returns

The desired axis of the slip plane's local co-ordinate system, expressed in the global co-ordinate system. In case of invalid argument, a zero vector is returned.

Definition at line 251 of file slipPlane.cpp.

```
{
  Vector3d = axis;
```

```
  if (i==2)
  {
    // Z-axis
    axis = this->normalVector;
  }

  if (i==0)
  {
    // X-axis
    Vector3d *e1 = new Vector3d;
    Vector3d *e2 = new Vector3d;

    *e1 = this->extremities[0].getPosition();
    *e2 = this->extremities[1].getPosition();
    axis = ((*e2) - (*e1));

    delete(e1);  e1 = NULL;
    delete(e2);  e2 = NULL;
  }

  if (i==1)
  {
    // Y-axis = Z x X
    axis = this->getAxis(2) ^ this->getAxis(0);
  }

  return ( axis.normalize() );
}
```

5.6.3.8 bool SlipPlane::getDislocation (int *i*, **Dislocation** $*$ *d*)

Get the dislocation on the slip plane indicated by the index provided as argument.

The slip plane contains several dislocations that are stored in a vector container. This function returns the dislocation in that vector that corresponds to the index provided as argument.

Parameters

i	Index of the required dislocation in the vector. This value should be greater than or equal to 0 and less than the number of dislocations on the slip plane.
d	Pointer to the memory location where the required dislocation is to be stored. Space in memory must be pre-allocated.

Returns

True if the provided index is greater than or equal to 0 and less than the number of dislocations on the slip plane (the memory location pointed to by d is populated with the Dislocation data). Otherwise, the return value is false.

Definition at line 168 of file slipPlane.cpp.

```
{
  if (i>=0 && i<this->dislocations.size ())
  {
    *d = this->dislocations[i];
    return (true);
  }
  else
  {
    return (false);
  }
}
```

5.6.3.9 std::vector$<$ **Dislocation** $>$ SlipPlane::getDislocationList ( )

Get the entire vector container which holds the dislocations lying on this slip plane.

Returns

The vector of dislocations lying on this slip plane.

Definition at line 185 of file slipPlane.cpp.

```
{
  return (this->dislocations);
}
```

5.6.3.10 bool SlipPlane::getDislocationSource (int *i*, DislocationSource * *dSource*)

Get the dislocation source on the slip plane indicated by the index provided as argument.

The slip plane contains several dislocation sources that are stored in a vector container. This function returns the dislocation source in that vector that corresponds to the index provided as argument.

Parameters

i	Index of the required dislocation source in the vector. This value should be greater than or equal to 0 and less than the number of dislocation sources on the slip plane.
dSource	Pointer to the memory location where the required dislocation source is to be stored. Space in memory must be pre-allocated.

Returns

True if the provided index is greater than or equal to 0 and less than the number of dislocation sources on the slip plane (the memory location pointed to by d is populated with the DislocationSource data). Otherwise, the return value is false.

Definition at line 206 of file slipPlane.cpp.

```
{
  if (i>=0 && i<this->dislocationSources.size ())
  {
    *dSource = this->dislocationSources[i];
    return (true);
  }
  else
  {
    return (false);
  }
}
```

5.6.3.11 std::vector< DislocationSource > SlipPlane::getDislocationSourceList ( )

Get the entire vector container which holds the dislocation sources lying on this slip plane.

Returns

The vector of dislocation sources lying on this slip plane.

Definition at line 223 of file slipPlane.cpp.

```
{
  return (this->dislocationSources);
}
```

5.6.3.12 Defect ∗ SlipPlane::getExtremities ( )

Get the position vectors of the extremities of the slip plane.

Returns

Pointer to an array containing the two extremities of the slip plane, variables of type Defect.
Pointer to an array containing the position vectors of the two extremities of the slip plane, variables of type Vector3d.

Definition at line 137 of file slipPlane.cpp.

```
{
  return (this->extremities);
}
```

5.6.3.13 Vector3d SlipPlane::getExtremity (int *i*)

Get the position vector of the extremity whose index is provided as argument.

Parameters

i	Index of the extremity. Possible values: 0, 1

Returns

Position vector of the extremity indicated by the argument, returned as a variable of type Vector3d.

Definition at line 121 of file slipPlane.cpp.

```
{
  if (i==0 || i==1)
  {
    return (this->extremities[i].getPosition());
  }
  else
  {
    return (Vector3d());
  }
}
```

5.6.3.14 Vector3d SlipPlane::getNormal ( )

Get the normal vector of the slip plane.

Returns

The normal vector of the slip plane, in a variable of type Vector3d.

Definition at line 146 of file slipPlane.cpp.

```
{
  return (this->normalVector);
}
```

5.6.3.15 int SlipPlane::getNumDislocations ( )

Get the number of dislocations.

Returns

The number of dislocations on the slip plane.

Definition at line 194 of file slipPlane.cpp.

```
{
  return (this->dislocations.size ());
}
```

5.6.3.16 int SlipPlane::getNumDislocationSources ( )

Get the number of dislocation sources.

Returns

The number of dislocation sources on the slip plane.

Definition at line 232 of file slipPlane.cpp.

```
{
  return (this->dislocationSources.size ());
}
```

5.6.3.17 Vector3d SlipPlane::getPosition ( )

Get the position vector of the slip plane.

This function returns the position vector of the slip plane. The position vector is redundant because the slip plane is completely defined by its extremities and the normal vector. Nevertheless, this value can be useful to locate the slip plane within a slip system.

Returns

Position vector of the slip plane, in a variable of type Vector3d.

Definition at line 156 of file slipPlane.cpp.

```
{
  return (this->position);
}
```

5.6.3.18 RotationMatrix SlipPlane::getRotationMatrix ( )

Get the rotation matrix for this slip plane.

Returns

The rotation matrix of this slip plane, in a variable of type RotationMatrix.

Definition at line 241 of file slipPlane.cpp.

```
{
  return (this->rotationMatrix);
}
```

5.6.3.19 void SlipPlane::moveDislocations ( )

Displaces the dislocations according to their velocities and the time increment.

Definition at line 513 of file slipPlane.cpp.

```
{
  std::vector<Dislocation>::iterator d;
  std::vector<Vector3d>::iterator v;
  Vector3d p;

  d = this->dislocations.begin();
  v = this->dislocationVelocities.begin();

  while (d != this->dislocations.end())
    {
      p = d->getPosition();
      p += (*v) * (this->dt);
      d->setPosition(p);

      d++;
      v++;
    }
}
```

5.6.3.20 void SlipPlane::setDislocationList (std::vector< Dislocation > *dislocationList*)

Set the list of dislocations of the slip plane.

Parameters

dislocationList	A vector container of type Dislocation containing the dislocations lying on this slip plane.

Definition at line 101 of file slipPlane.cpp.

```
{
  this->dislocations = dislocationList;
}
```

5.6.3.21 void SlipPlane::setDislocationSourceList (std::vector< DislocationSource > *dislocationSourceList*)

Set the list of dislocation sources on the slip plane.

Parameters

dislocation-SourceList	A vector container of type DislocationSource containing the dislocation sources lying on this slip plane.

Definition at line 110 of file slipPlane.cpp.

```
{
  this->dislocationSources = dislocationSourceList;
}
```

5.6.3.22 void SlipPlane::setExtremities (Vector3d * *ends*)

Set the extremities of the slip plane.

Parameters

ends	Pointer to an array of type Vector3d, containing the position vectors of the extremities of the slip plane in consecutive locations.

Definition at line 73 of file slipPlane.cpp.

```
{
  this->extremities[0] = Defect(ends);
  this->extremities[1] = Defect(ends+1);
}
```

5.6.3.23 void SlipPlane::setNormal (Vector3d *normal*)

Set the normal vector of the slip plane.

Parameters

normal	The normal vector of the slip plane.

Definition at line 83 of file slipPlane.cpp.

```
{
  this->normalVector = normal;
}
```

5.6.3.24 void SlipPlane::setPosition (Vector3d *pos*)

Set the position of the slip plane.

Parameters

pos	The position vector of the slip plane. (This parameter is useful for locating the slip plane within a slip system)

Definition at line 92 of file slipPlane.cpp.

```
{
  this->position = pos;
}
```

5.6.3.25 void SlipPlane::sortDislocations ( )

Sorts the dislocations present on the slip plane in the ascending order of distance from the first extremity.

The dislocations present on the slip plane are sorted in ascending order of distance from the first extremity of the slip plane.

Definition at line 554 of file slipPlane.cpp.

```
{
  int nDisl = this->dislocations.size();
  int i, j;
  double di, dj;
  Vector pi, pj;
  Dislocation temp;

  for (i=0; i<nDisl-1; i++)
    {
      for (j=i+1; j<nDisl; j++)
        {
          pi = this->dislocations[i].getPosition();
          di = this->distanceFromExtremity(pi, 0);

          pj = this->dislocations[j].getPosition();
          dj = this->distanceFromExtremity(pj, 0);

          if (dj < di)
```

```
            {
              // Swap the two
              temp = this->dislocations[i];
              this->dislocations[i] = this->dislocations[j];
              this->dislocations[j] = temp;
            }
        }
    }
}
```

5.6.4 Field Documentation

5.6.4.1 std::vector<Vector3d> SlipPlane::dislocationForces [protected]

The Peach-Koehler force experienced by each dislocation.

This vector container stores the Peah-Koehler force experienced by each dislocation. They are calculated in each iteration by thefunction calculateDislocationForces(tau_crss).

Definition at line 66 of file slipPlane.h.

5.6.4.2 std::vector<Dislocation> SlipPlane::dislocations [protected]

STL vector container with dislocations.

A slip plane may contain several dislocations. These are stored in this vector container dislocations.

Definition at line 54 of file slipPlane.h.

5.6.4.3 std::vector<DislocationSource> SlipPlane::dislocationSources [protected]

STL vector container with dislocation sources.

A slip plane may contain several dislocation sources. These are stored in this vector container dislocationSources.

Definition at line 78 of file slipPlane.h.

5.6.4.4 std::vector<Stress> SlipPlane::dislocationStresses [protected]

STL vector container with the stress fields of dislocations.

The stress fields experienced by the dislocations, expressed in the global co-ordinate system, are stored in this vector with positions corresponding to the positions of dislocations in the vector dislocations.

Definition at line 60 of file slipPlane.h.

5.6.4.5 std::vector<Vector3d> SlipPlane::dislocationVelocities [protected]

STL vector container with dislocation velocities.

The dislocations on this slip plane will have a velocity associated with them. These velocity vectors are stored in this container. The order is the same as the order of the dislocations.

Definition at line 72 of file slipPlane.h.

5.6.4.6 double SlipPlane::dt [protected]

Time increment for the slip plane.

A time increment is calculated for each slip plane based on the distances traveled by the dislocations.

Definition at line 84 of file slipPlane.h.

5.6.4.7 Defect SlipPlane::extremities[2] [protected]

The extremities of the slip plane.

The slip plane is represented as a straight line in these two dimensional simulations. The position vectors of the two ends are given here.

Definition at line 36 of file slipPlane.h.

5.6.4.8 Vector3d SlipPlane::normalVector [protected]

The normal vector to the slip plane.

This is the vector normal to the slip plane. Since we are concerned with the cubic system here, the indices of the normal vector are the same as those of the slip plane.

Definition at line 42 of file slipPlane.h.

5.6.4.9 Vector3d SlipPlane::position [protected]

The position vector of the slip plane.

This position vector is redundant because the combination of the position vectors of the extremities and the normal vector define the slip plane completely. However, this vector, position, is useful to locate the slip plane in a given slip system.

Definition at line 48 of file slipPlane.h.

5.6.4.10 RotationMatrix SlipPlane::rotationMatrix [protected]

Rotation matrix for co-ordinate system transformations.

The slip plane's local co-ordinate system is defined as follows: z-axis||NormalVector; x-axis||slipPlane line. The rotation matrix is created using this convention.

Definition at line 90 of file slipPlane.h.

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

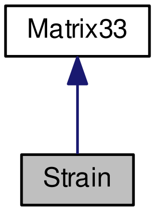
- slipPlane.h
- slipPlane.cpp

5.7 Strain Class Reference

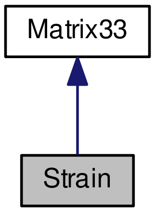
Strain class to represent the strain tensor.

`#include <strain.h>`

Inheritance diagram for Strain:



Collaboration diagram for Strain:



Public Member Functions

- Strain ()

    Default constructor.
- Strain (double ∗principal, double ∗shear)

    Constructor specifying the principal and shear strains.
- void populateMatrix ()

    Construct the strain tensor from the principal and shear strains.
- double ∗ getPrincipalStrains ()

    Get the principal strains.
- double ∗ getShearStrains ()

    Get the shear strains.
- Strain rotate (RotationMatrix alpha)

    Rotate the strain tensor from one coordinate system to another.
- void setValue (int row, int column, double value)

    Function to set the value of an element indicated by its position.
- double getValue (int row, int column)

    Returns the value of the element located by the row and column indices provided.
- Matrix33 adjugate ()

    Returns the adjugate matrix of the present matrix.

- Matrix33 operator+ (const Matrix33 &) const

  Operator for addition of two matrices.
- void operator+= (const Matrix33 &)

  Operator for reflexive addition of two matrices.
- Matrix33 operator- (const Matrix33 &) const

  Operator for the subtraction of two matrices.
- void operator-= (const Matrix33 &)

  Operator for reflexive subtraction of two matrices.
- Matrix33 operator∗ (const double &) const

  Operator for scaling the matrix by a scalar.
- Matrix33 operator∗ (const Matrix33 &) const

  Operator for the multiplication of two matrices.
- Vector3d operator∗ (const Vector3d &) const

  Operator for the multiplication of a matrix with a vector.
- void operator∗= (const double &)

  Operator for reflexive scaling of the matrix by a scalar.
- void operator∗= (const Matrix33 &)

  Operator for reflexive multiplication of two matrices.
- Matrix33 operator^ () const

  Transpose.
- double operator∼ () const

  Determinant.
- Matrix33 operator! () const

  Inverse.

Protected Attributes

- double principalStrains [3]
- double shearStrains [3]
- double x [3][3]

  Array containing the elements of the matrix.

5.7.1 Detailed Description

Strain class to represent the strain tensor.

The member functions of this class construct the symmetric strain tensor and operate on it.

Definition at line 21 of file strain.h.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 Strain::Strain ( )

Default constructor.

Initializes the strain tensor with zeros.

Definition at line 16 of file strain.cpp.

```
{
  int i, j;

  for (i=0; i<3; i++)
    {
      principalStrains [i] = 0.0;
      shearStrains [i] = 0.0;
    }

  this->populateMatrix ();
}
```

5.7.2.2 Strain::Strain (double ∗ *principal,* double ∗ *shear*)

Constructor specifying the principal and shear strains.

The principal and shear strains are provided in the arguments and the symmetrical strain tensor is contstructed using them.

Parameters

principal	Pointer to the array containing principal strains.
shear	Pointer to the array containing shear strains.

Definition at line 35 of file strain.cpp.

```
{
  int i;

  for (i=0; i<3; i++)
    {
      this->principalStrains [i] = principal [i];
      this->shearStrains [i] = shear [i];
    }

  this->populateMatrix ();
}
```

5.7.3 Member Function Documentation

5.7.3.1 Matrix33 Matrix33::adjugate ( ) [inherited]

Returns the adjugate matrix of the present matrix.

The adjugate matrix of the present matrix is returned. The adjugate matrix is calculated by evaluating the determinant of the cofactor matrix of each element, and then replacing the corresponding element position by the value of the determinant. This operation is useful in calculating the inverse of a matrix.

Returns

The adjugate matrix of the present matrix.

Definition at line 129 of file matrix33.cpp.

```
{
  Matrix33 adj;

  adj.x[0][0] = (this->x[1][1]*this->x[2][2]) - (this->x[1][2]*this->x[2][1]);
  adj.x[0][1] = (this->x[1][2]*this->x[2][0]) - (this->x[1][0]*this->x[2][2]);
  adj.x[0][2] = (this->x[1][0]*this->x[2][1]) - (this->x[1][1]*this->x[2][0]);

  adj.x[1][0] = (this->x[2][1]*this->x[0][2]) - (this->x[0][1]*this->x[2][2]);
  adj.x[1][1] = (this->x[2][2]*this->x[0][0]) - (this->x[2][0]*this->x[0][2]);
  adj.x[1][2] = (this->x[2][0]*this->x[0][1]) - (this->x[2][1]*this->x[0][0]);

  adj.x[2][0] = (this->x[0][1]*this->x[1][2]) - (this->x[0][2]*this->x[1][1]);
  adj.x[2][1] = (this->x[0][2]*this->x[1][0]) - (this->x[0][0]*this->x[1][2]);
  adj.x[2][2] = (this->x[0][0]*this->x[1][1]) - (this->x[1][0]*this->x[0][1]);

  return (adj);
}
```

5.7.3.2 double ∗ Strain::getPrincipalStrains ( )

Get the principal strains.

Returns a 3-member array with the principal strains: s11 s22 s33.

Returns

3-member array with the principal strains.

Definition at line 68 of file strain.cpp.

```
{
  double p[3];
  int i;

  for (i=0; i<3; i++)
    {
      p[i] = this->principalStrains[i];
    }

  return (p);
}
```

5.7.3.3 double ∗ Strain::getShearStrains ( )

Get the shear strains.

Returns a 3-member array with the shear strains: s12 s13 s23.

Returns

3-member array with the shear strains.

Definition at line 86 of file strain.cpp.

```
{
  double s[3];
  int i;

  for (i=0; i<3; i++)
    {
      s[i] = this->shearStrains[i];
    }

  return (s);
}
```

5.7.3.4 double Matrix33::getValue (int *row*, int *column*) [inherited]

Returns the value of the element located by the row and column indices provided.

The value of the element indicated by the arguments row and column is returned. The values of row and column must correspond to array indices, and thus can be one of 0, 1 and 2. In any other case 0.0 is returned.

Parameters

row	Row index of the element.
column	Column index of the element.

Returns

Value of the element located at the given position.

Definition at line 111 of file matrix33.cpp.

```
{
  if (row>=0 && row<3)
    {
      if (column>=0 && column<3)
        {
          return (this->x[row][column]);
        }
    }

  return (0.0);
}
```

5.7.3.5 Matrix33 Matrix33::operator! ( ) const [inherited]

Inverse.

Returns in a new matrix the inverse of the current matrix. If the current matrix is non-invertible, a zero matrix is returned.

Returns

Matrix with the inverse of the current matrix. If the current matrix is non-invertible, a zero matrix is returned.

Definition at line 370 of file matrix33.cpp.

```
{
  Matrix33 r;    // Result matrix

  double determinant = ~(*this);

  if (determinant == 0.0)
    {
      // The matrix is non-invertible
      return (r);        // Zero matrix
    }

  // If we are still here, the matrix is invertible

  //  Transpose
  Matrix33 tr = ^(*this);

  // Find Adjugate matrix
  Matrix33 adj = tr.adjugate();

  // Calculate the inverse by dividing the adjugate matrix by the determinant
  r = adj * (1.0/determinant);

  return (r);
}
```

5.7.3.6 Matrix33 Matrix33::operator* (const double & *p*) const [inherited]

Operator for scaling the matrix by a scalar.

Scales the current matrix by the scalar provided and returns the result in a third matrix.

Returns

Matrix containing the result of scaling the current matrix by the scalar provided as argument.

Definition at line 233 of file matrix33.cpp.

```
{
  int i, j;
  Matrix33 r;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          r.x[i][j] = this->x[i][j] * p;
        }
    }

  return (r);
}
```

5.7.3.7 Matrix33 Matrix33::operator* (const Matrix33 & *p*) const [inherited]

Operator for the multiplication of two matrices.

Multiplies the current matrix with another 3x3 matrix and returns the result in a new matrix.

Returns

The result of the multiplication of the current matrix with the one provided as argument.

Definition at line 271 of file matrix33.cpp.

```
{
  int i, j, k;
  Matrix33 r;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          r.x[i][j] = 0.0;
          for (k=0; k<3; k++)
            {
              r.x[i][j] += this->x[i][k] * p.x[k][j];
            }
        }
    }

  return (r);
}
```

5.7.3.8 Vector3d Matrix33::operator* (const Vector3d & *v*) const [inherited]

Operator for the multiplication of a matrix with a vector.

Returns in a vector the result of the multiplication of the current matrix with the provided vector.

Returns

The vector resulting from the multiplication of the current matrix with a vector.

Definition at line 311 of file matrix33.cpp.

```
{
  Vector3d r(0.0, 0.0, 0.0);
  int i, j;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          r[i] += this->x[i][j] * v.x[j];
        }
    }

  return (r);
}
```

5.7.3.9 void Matrix33::operator*= (const double & *p*) [inherited]

Operator for reflexive scaling of the matrix by a scalar.

Scales the current matrix by the scalar provided and populates the current matrix elements with the result.

Definition at line 253 of file matrix33.cpp.

```
{
  int i, j;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          this->x[i][j] *= p;
        }
    }
}
```

5.7.3.10 void Matrix33::operator*= (const Matrix33 & *p*) [inherited]

Operator for reflexive multiplication of two matrices.

Multiplies the current matrix with another 3x3 matrix and populates the elements of the current matrix with the result.

Definition at line 295 of file matrix33.cpp.

```
{
  Matrix33* r = new Matrix33;

  *r = (*this) * p;
  *this = *r;

  delete(r);
  r = NULL;
}
```

5.7.3.11 Matrix33 Matrix33::operator+ (const Matrix33 & *p*) const [inherited]

Operator for addition of two matrices.

Adds the current matrix to the provided matrix and returns a third matrix with the result.

Returns

Matrix containing the sum of the present matrix and the one provided.

Definition at line 155 of file matrix33.cpp.

```
{
  int i, j;
  Matrix33 r;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          r.x[i][j] = this->x[i][j] + p.x[i][j];
        }
    }

  return (r);
}
```

5.7.3.12 void Matrix33::operator+= (const Matrix33 & *p*) [inherited]

Operator for reflexive addition of two matrices.

Adds the current matrix to the provided matrix and populates the current matrix elements with the result.

Definition at line 175 of file matrix33.cpp.

```
{
  int i, j;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          this->x[i][j] += p.x[i][j];
        }
    }
}
```

5.7.3.13 Matrix33 Matrix33::operator- (const Matrix33 & *p*) const [inherited]

Operator for the subtraction of two matrices.

Subtracts the given matrix from the current matrix and returns the result in a new matrix.

Returns

Matrix containing the result of subtracting the provided matrix from the current matrix.

Definition at line 194 of file matrix33.cpp.

```
{
  int i, j;
  Matrix33 r;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          r.x[i][j] = this->x[i][j] - p.x[i][j];
        }
    }

  return (r);
}
```

5.7.3.14 void Matrix33::operator-= (const Matrix33 & *p*) [inherited]

Operator for reflexive subtraction of two matrices.

Subtracts the given matrix from the current matrix and populates the current matrix with the result.

Definition at line 214 of file matrix33.cpp.

```
{
  int i, j;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          this->x[i][j] -= p.x[i][j];
        }
    }
}
```

5.7.3.15 Matrix33 Matrix33::operator^ ( ) const [inherited]

Transpose.

Performs the transpose of the current matrix.

Returns

A new matrix with the transpose of the current matrix.

Definition at line 333 of file matrix33.cpp.

```
{
  Matrix33 r;
  int i, j;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          r.x[i][j] = this->x[j][i];
        }
    }

  return (r);
}
```

5.7.3.16 double Matrix33::operator~ ( ) const [inherited]

Determinant.

Calculates the determinant of the current matrix.

Returns

Returns the determinant of the current matrix.

Definition at line 354 of file matrix33.cpp.

```
{
  double d = 0.0;

  d += this.x[0][0] * ( (this->x[1][1]*this->x[2][2]) - (this->x[2][1]*this->x[1][2]) );
  d += this.x[0][1] * ( (this->x[1][2]*this->x[2][0]) - (this->x[1][0]*this->x[2][2]) );
  d += this.x[0][2] * ( (this->x[1][0]*this->x[2][1]) - (this->x[2][0]*this->x[1][1]) );

  return (d);
}
```

5.7.3.17 void Strain::populateMatrix ( )

Construct the strain tensor from the principal and shear strains.

Takes the values in principalStrains and shearStrains and constructs the symmetrical strain matrix.

Definition at line 52 of file strain.cpp.

```
{
  this->x[0][0] = this->principalStrains [0];
  this->x[1][1] = this->principalStrains [1];
  this->x[2][2] = this->principalStrains [2];

  this->x[0][1] = this->x[1][0] = this->shearStrains [0];
  this->x[0][2] = this->x[2][0] = this->shearStrains [1];
  this->x[1][2] = this->x[2][1] = this->shearStrains [2];
}
```

5.7.3.18 Strain Strain::rotate (RotationMatrix *alpha*)

Rotate the strain tensor from one coordinate system to another.

Rotates the present strain matrix from one coordinate system to another using the rotation matrix supplied. The result is returned in a new Strain matrix.

Parameters

alpha	Rotation matrix.

Returns

Rotated strain tensor.

Definition at line 105 of file strain.cpp.

```
{
  Matrix33 alphaT = ^alpha;  // Transpose
  Strain sNew;

  sNew = alpha * (*this) * alphaT;  // Rotate the strain matrix

  return (sNew);
}
```

5.7.3.19 void Matrix33::setValue (int *row*, int *column*, double *value*) [inherited]

Function to set the value of an element indicated by its position.

The element indicated by the arguments row and column is set to the value provided. The values of row and column must correspond to array indices, and thus can be one of 0, 1 and 2. In any other case 0.0 is returned.

Parameters

row	Row index of the element.
column	Column index of the element.
value	Value that the element is to be set to.

Definition at line 93 of file matrix33.cpp.

```
{
  if (row>=0 && row<3)
    {
      if (column>=0 && column<3)
        {
          this->x[row][column] = value;
         }
     }
}
```

5.7.4 Field Documentation

5.7.4.1 double Strain::principalStrains[3] [protected]

The three principal strains: s11, s22, s33.

Definition at line 27 of file strain.h.

5.7.4.2 double Strain::shearStrains[3] [protected]

The three shear strains: s12, s13, s23,

Definition at line 31 of file strain.h.

5.7.4.3 double Matrix33::x[3][3] [protected], [inherited]

Array containing the elements of the matrix.

Definition at line 26 of file matrix33.h.

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

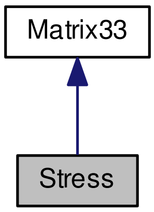
- strain.h
- strain.cpp

5.8 Stress Class Reference

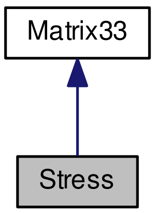
Stress class to represent the stress tensor.

`#include <stress.h>`

Inheritance diagram for Stress:



Collaboration diagram for Stress:



Public Member Functions

- Stress ()

Default constructor.
- Stress (double ∗principal, double ∗shear)

  Constructor specifying the principal and shear stresses.
- void populateMatrix ()

  Construct the stress tensor from the principal and shear stresses.
- double ∗ getPrincipalStresses ()

  Get the principal stresses.
- double ∗ getShearStresses ()

  Get the shear stresses.
- Stress rotate (RotationMatrix alpha)

  Rotate the stress tensor from one coordinate system to another.
- void setValue (int row, int column, double value)

  Function to set the value of an element indicated by its position.
- double getValue (int row, int column)

  Returns the value of the element located by the row and column indices provided.
- Matrix33 adjugate ()

  Returns the adjugate matrix of the present matrix.
- Matrix33 operator+ (const Matrix33 &) const

  Operator for addition of two matrices.
- void operator+= (const Matrix33 &)

  Operator for reflexive addition of two matrices.
- Matrix33 operator- (const Matrix33 &) const

  Operator for the subtraction of two matrices.
- void operator-= (const Matrix33 &)

  Operator for reflexive subtraction of two matrices.
- Matrix33 operator∗ (const double &) const

  Operator for scaling the matrix by a scalar.
- Matrix33 operator∗ (const Matrix33 &) const

  Operator for the multiplication of two matrices.
- Vector3d operator∗ (const Vector3d &) const

  Operator for the multiplication of a matrix with a vector.
- void operator∗= (const double &)

  Operator for reflexive scaling of the matrix by a scalar.
- void operator∗= (const Matrix33 &)

  Operator for reflexive multiplication of two matrices.
- Matrix33 operator^ () const

  Transpose.
- double operator∼ () const

  Determinant.
- Matrix33 operator! () const

  Inverse.

Protected Attributes

- double principalStresses [3]
- double shearStresses [3]
- double x [3][3]

  Array containing the elements of the matrix.

5.8.1 Detailed Description

Stress class to represent the stress tensor.

The member functions of this class construct the symmetric stress tensor and operate on it.

Definition at line 21 of file stress.h.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 Stress::Stress ( )

Default constructor.

Initializes the stress tensor with zeros.

Definition at line 16 of file stress.cpp.

```
{
  int i, j;

  for (i=0; i<3; i++)
    {
      principalStresses [i] = 0.0;
      shearStresses [i] = 0.0;
    }

  this->populateMatrix ();
}
```

5.8.2.2 Stress::Stress (double ∗ *principal*, double ∗ *shear*)

Constructor specifying the principal and shear stresses.

The principal and shear stresses are provided in the arguments and the symmetrical stress tensor is contstructed using them.

Parameters

principal	Pointer to the array containing principal stresses.
shear	Pointer to the array containing shear stresses.

Definition at line 35 of file stress.cpp.

```
{
  int i;

  for (i=0; i<3; i++)
    {
      this->principalStresses [i] = principal [i];
      this->shearStresses [i] = shear [i];
    }

  this->populateMatrix ();
}
```

5.8.3 Member Function Documentation

5.8.3.1 Matrix33 Matrix33::adjugate ( ) [inherited]

Returns the adjugate matrix of the present matrix.

The adjugate matrix of the present matrix is returned. The adjugate matrix is calculated by evaluating the determinant of the cofactor matrix of each element, and then replacing the corresponding element position by the value of the determinant. This operation is useful in calculating the inverse of a matrix.

Returns

The adjugate matrix of the present matrix.

Definition at line 129 of file matrix33.cpp.

```
{
  Matrix33 adj;

  adj.x[0][0] = (this->x[1][1]*this->x[2][2]) - (this->x[1][2]*this->x[2][1]);
  adj.x[0][1] = (this->x[1][2]*this->x[2][0]) - (this->x[1][0]*this->x[2][2]);
  adj.x[0][2] = (this->x[1][0]*this->x[2][1]) - (this->x[1][1]*this->x[2][0]);

  adj.x[1][0] = (this->x[2][1]*this->x[0][2]) - (this->x[0][1]*this->x[2][2]);
  adj.x[1][1] = (this->x[2][2]*this->x[0][0]) - (this->x[2][0]*this->x[0][2]);
  adj.x[1][2] = (this->x[2][0]*this->x[0][1]) - (this->x[2][1]*this->x[0][0]);

  adj.x[2][0] = (this->x[0][1]*this->x[1][2]) - (this->x[0][2]*this->x[1][1]);
  adj.x[2][1] = (this->x[0][2]*this->x[1][0]) - (this->x[0][0]*this->x[1][2]);
  adj.x[2][2] = (this->x[0][0]*this->x[1][1]) - (this->x[1][0]*this->x[0][1]);

  return (adj);
}
```

5.8.3.2 double * Stress::getPrincipalStresses ( )

Get the principal stresses.

Returns a 3-member array with the principal stresses: s11 s22 s33.

Returns

3-member array with the principal stresses.

Definition at line 68 of file stress.cpp.

```
{
  double p[3];
  int i;

  for (i=0; i<3; i++)
    {
      p[i] = this->principalStresses[i];
    }

  return (p);
}
```

5.8.3.3 double * Stress::getShearStresses ( )

Get the shear stresses.

Returns a 3-member array with the shear stresses: s12 s13 s23.

Returns

3-member array with the shear stresses.

Definition at line 86 of file stress.cpp.

```
{
  double s[3];
  int i;

  for (i=0; i<3; i++)
    {
      s[i] = this->shearStresses[i];
    }

  return (s);
}
```

5.8.3.4 double Matrix33::getValue (int *row*, int *column*) [inherited]

Returns the value of the element located by the row and column indices provided.

The value of the element indicated by the arguments row and column is returned. The values of row and column must correspond to array indices, and thus can be one of 0, 1 and 2. In any other case 0.0 is returned.

Parameters

row	Row index of the element.
column	Column index of the element.

Returns

Value of the element located at the given position.

Definition at line 111 of file matrix33.cpp.

```
{
  if (row>=0 && row<3)
    {
      if (column>=0 && column<3)
        {
          return (this->x[row][column]);
        }
    }

  return (0.0);
}
```

5.8.3.5 Matrix33 Matrix33::operator! ( ) const [inherited]

Inverse.

Returns in a new matrix the inverse of the current matrix. If the current matrix is non-invertible, a zero matrix is returned.

Returns

Matrix with the inverse of the current matrix. If the current matrix is non-invertible, a zero matrix is returned.

Definition at line 370 of file matrix33.cpp.

```
{
  Matrix33 r;    // Result matrix

  double determinant = ~(*this);

  if (determinant == 0.0)
    {
      // The matrix is non-invertible
      return (r);        // Zero matrix
    }

  // If we are still here, the matrix is invertible

  //  Transpose
  Matrix33 tr = ^(*this);

  // Find Adjugate matrix
  Matrix33 adj = tr.adjugate();

  // Calculate the inverse by dividing the adjugate matrix by the determinant
  r = adj * (1.0/determinant);

  return (r);
}
```

5.8.3.6 Matrix33 Matrix33::operator* (const double & *p*) const [inherited]

Operator for scaling the matrix by a scalar.

Scales the current matrix by the scalar provided and returns the result in a third matrix.

Returns

Matrix containing the result of scaling the current matrix by the scalar provided as argument.

Definition at line 233 of file matrix33.cpp.

```
{
  int i, j;
  Matrix33 r;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          r.x[i][j] = this->x[i][j] * p;
        }
    }

  return (r);
}
```

5.8.3.7 Matrix33 Matrix33::operator* (const Matrix33 & *p*) const [inherited]

Operator for the multiplication of two matrices.

Multiplies the current matrix with another 3x3 matrix and returns the result in a new matrix.

Returns

The result of the multiplication of the current matrix with the one provided as argument.

Definition at line 271 of file matrix33.cpp.

```
{
  int i, j, k;
  Matrix33 r;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          r.x[i][j] = 0.0;
          for (k=0; k<3; k++)
            {
              r.x[i][j] += this->x[i][k] * p.x[k][j];
            }
        }
    }

  return (r);
}
```

5.8.3.8 Vector3d Matrix33::operator* (const Vector3d & *v*) const [inherited]

Operator for the multiplication of a matrix with a vector.

Returns in a vector the result of the multiplication of the current matrix with the provided vector.

Returns

The vector resulting from the multiplication of the current matrix with a vector.

Definition at line 311 of file matrix33.cpp.

```
{
  Vector3d r(0.0, 0.0, 0.0);
  int i, j;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          r[i] += this->x[i][j] * v.x[j];
        }
    }

  return (r);
}
```

5.8.3.9 void Matrix33::operator*= (const double & *p*) [inherited]

Operator for reflexive scaling of the matrix by a scalar.

Scales the current matrix by the scalar provided and populates the current matrix elements with the result.

Definition at line 253 of file matrix33.cpp.

```
{
  int i, j;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          this->x[i][j] *= p;
        }
    }
}
```

5.8.3.10 void Matrix33::operator*= (const Matrix33 & *p*) [inherited]

Operator for reflexive multiplication of two matrices.

Multiplies the current matrix with another 3x3 matrix and populates the elements of the current matrix with the result.

Definition at line 295 of file matrix33.cpp.

```
{
  Matrix33* r = new Matrix33;

  *r = (*this) * p;
  *this = *r;

  delete(r);
  r = NULL;
}
```

5.8.3.11 Matrix33 Matrix33::operator+ (const Matrix33 & *p*) const [inherited]

Operator for addition of two matrices.

Adds the current matrix to the provided matrix and returns a third matrix with the result.

Returns

Matrix containing the sum of the present matrix and the one provided.

Definition at line 155 of file matrix33.cpp.

```
{
  int i, j;
  Matrix33 r;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          r.x[i][j] = this->x[i][j] + p.x[i][j];
        }
    }

  return (r);
}
```

5.8.3.12 void Matrix33::operator+= (const Matrix33 & *p*) [inherited]

Operator for reflexive addition of two matrices.

Adds the current matrix to the provided matrix and populates the current matrix elements with the result.

Definition at line 175 of file matrix33.cpp.

```
{
  int i, j;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          this->x[i][j] += p.x[i][j];
        }
    }
}
```

5.8.3.13 Matrix33 Matrix33::operator- (const Matrix33 & *p*) const [inherited]

Operator for the subtraction of two matrices.

Subtracts the given matrix from the current matrix and returns the result in a new matrix.

Returns

Matrix containing the result of subtracting the provided matrix from the current matrix.

Definition at line 194 of file matrix33.cpp.

```
{
  int i, j;
  Matrix33 r;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          r.x[i][j] = this->x[i][j] - p.x[i][j];
        }
    }

  return (r);
}
```

5.8.3.14 void Matrix33::operator-= (const Matrix33 & *p*) [inherited]

Operator for reflexive subtraction of two matrices.

Subtracts the given matrix from the current matrix and populates the current matrix with the result.

Definition at line 214 of file matrix33.cpp.

```
{
  int i, j;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          this->x[i][j] -= p.x[i][j];
        }
    }
}
```

5.8.3.15 Matrix33 Matrix33::operator^ ( ) const [inherited]

Transpose.

Performs the transpose of the current matrix.

Returns

A new matrix with the transpose of the current matrix.

Definition at line 333 of file matrix33.cpp.

```
{
  Matrix33 r;
  int i, j;

  for (i=0; i<3; i++)
    {
      for (j=0; j<3; j++)
        {
          r.x[i][j] = this->x[j][i];
        }
    }

  return (r);
}
```

5.8.3.16 double Matrix33::operator~ ( ) const [inherited]

Determinant.

Calculates the determinant of the current matrix.

Returns

Returns the determinant of the current matrix.

Definition at line 354 of file matrix33.cpp.

```
{
  double d = 0.0;

  d += this.x[0][0] * ( (this->x[1][1]*this->x[2][2]) - (this->x[2][1]*this->
  x[1][2]) );
  d += this.x[0][1] * ( (this->x[1][2]*this->x[2][0]) - (this->x[1][0]*this->
  x[2][2]) );
  d += this.x[0][2] * ( (this->x[1][0]*this->x[2][1]) - (this->x[2][0]*this->
  x[1][1]) );

  return (d);
}
```

5.8.3.17 void Stress::populateMatrix ( )

Construct the stress tensor from the principal and shear stresses.

Takes the values in principalStresses and shearStresses and constructs the symmetrical stress matrix.

Definition at line 52 of file stress.cpp.

```
{
  this->x[0][0] = this->principalStresses [0];
  this->x[1][1] = this->principalStresses [1];
  this->x[2][2] = this->principalStresses [2];

  this->x[0][1] = this->x[1][0] = this->shearStresses [0];
  this->x[0][2] = this->x[2][0] = this->shearStresses [1];
  this->x[1][2] = this->x[2][1] = this->shearStresses [2];
}
```

5.8.3.18 Stress Stress::rotate (RotationMatrix *alpha*)

Rotate the stress tensor from one coordinate system to another.

Rotates the present stress matrix from one coordinate system to another using the rotation matrix supplied. The result is returned in a new Stress matrix.

Parameters

alpha	Rotation matrix.

Returns

Rotated stress tensor.

Definition at line 105 of file stress.cpp.

```
{
  Matrix33 alphaT = ^alpha;  // Transpose
  Stress sNew;

  sNew = alpha * (*this) * alphaT;  // Rotate the stress matrix

  return (sNew);
}
```

5.8.3.19 void Matrix33::setValue (int *row*, int *column*, double *value*) [inherited]

Function to set the value of an element indicated by its position.

The element indicated by the arguments row and column is set to the value provided. The values of row and column must correspond to array indices, and thus can be one of 0, 1 and 2. In any other case 0.0 is returned.

Parameters

row	Row index of the element.
column	Column index of the element.
value	Value that the element is to be set to.

Definition at line 93 of file matrix33.cpp.

```
{
  if (row>=0 && row<3)
    {
      if (column>=0 && column<3)
        {
```

```
        this->x[row][column] = value;
      }
  }
}
```

5.8.4 Field Documentation

5.8.4.1 double Stress::principalStresses[3] [protected]

The three principal stresses: s11, s22, s33.

Definition at line 27 of file stress.h.

5.8.4.2 double Stress::shearStresses[3] [protected]

The three shear stresses: s12, s13, s23,

Definition at line 31 of file stress.h.

5.8.4.3 double Matrix33::x[3][3] [protected], [inherited]

Array containing the elements of the matrix.

Definition at line 26 of file matrix33.h.

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

- stress.h
- stress.cpp

5.9 Vector3d Class Reference

Vector3d class representing a single 3-dimensional vector in the simulation.

`#include <vector3d.h>`

Public Member Functions

- Vector3d ()

  Default constructor.
- Vector3d (double ∗a)

  Constructor with values provided in an array.
- Vector3d (double a1, double a2, double a3)

  Constructor with values provided explicitly.
- void setValue (int index, double value)

  Function to set the value of an element of the vector.
- void setVector (double ∗a)

  Function to set the value of the entire vector using an array.
- double getValue (int index)

  Function to get the value of an element of the vector.
- double ∗ getVector ()

  Function to get the values of the elements of the vector in an array.
- double sum ()

  Computes the sum of the elements of the vector.

- double magnitude ()

  Computes the magnitude of the vector.
- Vector3d normalize ()

  Returns the vector normalized to be a unit vector.
- Vector3d operator+ (const Vector3d &) const

  Operator for addition of two vectors.
- void operator+= (const Vector3d &)

  Operator for reflexive addition of two vectors.
- Vector3d operator- (const Vector3d &) const

  Operator for the subtraction of two vectors.
- void operator-= (const Vector3d &)

  Operator for reflexive subtraction of two vectors.
- Vector3d operator* (const double &) const

  Operator for scaling the vector by a scalar.
- void operator*= (const double &)

  Operator for reflexive scaling of the vector by a scalar.
- double operator* (const Vector3d &) const

  Operator for the scalar product of two vectors.
- Vector3d operator^ (const Vector3d &) const

  Operator for the vector product of two vectors.
- void operator^= (const Vector3d &)

  Operator for reflexive vector product of two vectors.

Protected Attributes

- double x [3]

  The elements of the vector.

5.9.1 Detailed Description

Vector3d class representing a single 3-dimensional vector in the simulation.

This class represents a vector in 3D space. The member functions and operators define various operations on the vector and its interactions with other data types.

Definition at line 20 of file vector3d.h.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 Vector3d::Vector3d ( )

Default constructor.

Initializes the vector with all elements equal to 0.0.

Definition at line 16 of file vector3d.cpp.

```
{
  this->x[0] = 0.0;
  this->x[1] = 0.0;
  this->x[2] = 0.0;
}
```

5.9.2.2 Vector3d::Vector3d (double ∗ *a*)

Constructor with values provided in an array.

Initializes the vector with the values provided in the array.

Parameters

a	Pointer to the array containing the elements of the vector

Definition at line 28 of file vector3d.cpp.

```
{
  this->x[0] = a[0];
  this->x[1] = a[1];
  this->x[2] = a[2];
}
```

5.9.2.3 Vector3d::Vector3d (double *a1*, double *a2*, double *a3*)

Constructor with values provided explicitly.

Initializes the vector with the three values provided as arguments.

Parameters

a1	Value of the first element of the vector.
a2	Value of the second element of the vector.
a3	Value of the third element of the vector.

Definition at line 42 of file vector3d.cpp.

```
{
  this->x[0] = a1;
  this->x[1] = a2;
  this->x[2] = a3;
}
```

5.9.3 Member Function Documentation

5.9.3.1 double Vector3d::getValue (int *index*)

Function to get the value of an element of the vector.

Returns the value of the element at the position indicated by the argument index.

Parameters

index	Index of the element whose value is to be got.

Returns

The value of the element of the vector at the position

Definition at line 83 of file vector3d.cpp.

```
{
  if (index>=0 && index<3)
    {
      return (this->x[index]);
    }
  else
```

```
    {
      return (0);
    }
}
```

5.9.3.2 double * Vector3d::getVector ( )

Function to get the values of the elements of the vector in an array.

The vector is returned in an array.

Returns

Pointer to the first term of an array containing the elements of the vector.

Definition at line 100 of file vector3d.cpp.

```
{
  double* a = new double[3];

  a[0] = this->x[0];
  a[1] = this->x[1];
  a[2] = this->x[2];

  return (a);
}
```

5.9.3.3 double Vector3d::magnitude ( )

Computes the magnitude of the vector.

Computes the magnitude of the vector. Basically the square root of the sum of the squares of the vector elements.

Returns

The magnitude of the vector.

Definition at line 134 of file vector3d.cpp.

```
{
  double s = 0.0;
  int i;

  for (i=0; i<3; i++)
  {
    s += this->x[i] * this->x[i];
  }

  return ( sqrt (s) );
}
```

5.9.3.4 Vector3d Vector3d::normalize ( )

Returns the vector normalized to be a unit vector.

This function normalizes a vector by dividing its elements by the magnitude. In case the magnitude is zero, a zero vector is returned.

Returns

Normalized vector.

Definition at line 152 of file vector3d.cpp.

```
{
  double m = this->magnitude ();

  if (m==0.0)
  {
    return (Vector3d ());
  }
  else
  {
    return ((*this) * (1.0/m));
  }
}
```

5.9.3.5 Vector3d Vector3d::operator* (const double & *p*) const

Operator for scaling the vector by a scalar.

Scales the current vector by the scalar provided and returns the result in a third vector.

Returns

Vector containing the result of scaling the current vector by the scala provided as argument.

Definition at line 239 of file vector3d.cpp.

```
{
  Vector3d r(0.0, 0.0, 0.0);
  int i;

  for (i=0; i<3; i++)
    {
      r.x[i] = this->x[i] * p;
    }

  return (r);
}
```

5.9.3.6 double Vector3d::operator* (const Vector3d & *p*) const

Operator for the scalar product of two vectors.

Performs the scalar product or dot product of the current vector with the one provided as argument and returns the result.

Returns

Scalar value of the scalar product of dot product of the current vector with the one provided as argument.

Definition at line 271 of file vector3d.cpp.

```
{
  double s = 0.0;
  int i;

  for (i=0; i<3; i++)
    {
      s += this->x[i] * p.x[i];
    }

  return (s);
}
```

5.9.3.7 void Vector3d::operator∗= (const double & *p*)

Operator for reflexive scaling of the vector by a scalar.

Scales the current vector by the scalar provided and populates the current vector elements with the result.

Definition at line 256 of file vector3d.cpp.

```
{
  int i;

  for (i=0; i<3; i++)
    {
      this->x[i] *= p;
    }
}
```

5.9.3.8 Vector3d Vector3d::operator+ (const Vector3d & *p*) const

Operator for addition of two vectors.

Adds the current vector to the provided vector and returns a third vector with the result.

Returns

Vector containing the sum of the current vector with the one provided as argument.

Definition at line 173 of file vector3d.cpp.

```
{
  Vector3d r (0.0, 0.0, 0.0);
  int i;

  for (i=0; i<3; i++)
    {
      r.x[i] = this->x[i] + p.x[i];
    }

  return (r);
}
```

5.9.3.9 void Vector3d::operator+= (const Vector3d & *p*)

Operator for reflexive addition of two vectors.

Adds the current vector to the provided vector and populates the current vector elements with the result.

Definition at line 190 of file vector3d.cpp.

```
{
  int i;

  for (i=0; i<3; i++)
    {
      this->x[i] += p.x[i];
    }
}
```

5.9.3.10 Vector3d Vector3d::operator- (const Vector3d & *p*) const

Operator for the subtraction of two vectors.

Subtracts the given vector from the current vector and returns the result in a new vector.

Returns

Vector containing the result of subtracting the vector provided as argument from the current vector.

Definition at line 206 of file vector3d.cpp.

```
{
  Vector3d r(0.0, 0.0, 0.0);
  int i;

  for (i=0; i<3; i++)
    {
      r.x[i] = this->x[i] - p.x[i];
    }

  return (r);
}
```

5.9.3.11 void Vector3d::operator-= (const Vector3d & *p*)

Operator for reflexive subtraction of two vectors.

Subtracts the given vector from the current vector and populates the current vector with the result.

Definition at line 223 of file vector3d.cpp.

```
{
  int i;

  for (i=0; i<3; i++)
    {
      this->x[i] -= p.x[i];
    }
}
```

5.9.3.12 Vector3d Vector3d::operator^ (const Vector3d & *p*) const

Operator for the vector product of two vectors.

Evaluates the vector product of the current vector with the provided vector and returns the result in a third vector.

Returns

Vector containing the result of the vector product of the current vector with the one provided as argument.

Definition at line 289 of file vector3d.cpp.

```
{
  Vector3d r(0.0, 0.0, 0.0);

  r.x[0] = (this->x[1] * p.x[2]) - (this->x[2] * p.x[1]);
  r.x[1] = (this->x[2] * p.x[0]) - (this->x[0] * p.x[2]);
  r.x[2] = (this->x[0] * p.x[1]) - (this->x[1] * p.x[0]);

  return (r);
}
```

5.9.3.13 void Vector3d::operator^= (const Vector3d & *p*)

Operator for reflexive vector product of two vectors.

Evaluates the vector product of the current vector and the one provided, and populates the result in the current vector.

Definition at line 304 of file vector3d.cpp.

```
{
  Vector3d* r = Vector3d(0.0, 0.0, 0.0);

  r->x[0] = (this->x[1] * p.x[2]) - (this->x[2] * p.x[1]);
  r->x[1] = (this->x[2] * p.x[0]) - (this->x[0] * p.x[2]);
  r->x[2] = (this->x[0] * p.x[1]) - (this->x[1] * p.x[0]);

  *this = *r;

  delete (r);
  r = NULL;
}
```

5.9.3.14 void Vector3d::setValue (int *index*, double *value*)

Function to set the value of an element of the vector.

Sets the value of the element indicated by the index argument.

Parameters

index	Index of the element whose value is to be set.
value	Value that is to be given to the element.

Definition at line 56 of file vector3d.cpp.

```
{
  if (index>=0 && index <3)
    {
      this->x[index] = value;
    }
}
```

5.9.3.15 void Vector3d::setVector (double ∗ *a*)

Function to set the value of the entire vector using an array.

Sets the values of the elements if the vector to values in the array pointed to by the argument a.

Parameters

a	Pointer ot the array containing the values of the elements of the vector.

Definition at line 69 of file vector3d.cpp.

```
{
  this->x[0] = a[0];
  this->x[1] = a[1];
  this->x[2] = a[2];
}
```

5.9.3.16 double Vector3d::sum ( )

Computes the sum of the elements of the vector.

Sums the elements of the vector and returns the result.

Returns

The sum of the elements of the vector.

Definition at line 116 of file vector3d.cpp.

```
{
  double s = 0.0;
  int i;

  for (i=0; i<3; i++)
    {
      s += this->x[i];
    }

  return (s);
}
```

5.9.4 Field Documentation

5.9.4.1 double Vector3d::x[3] [protected]

The elements of the vector.

Definition at line 26 of file vector3d.h.

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

- vector3d.h
- vector3d.cpp

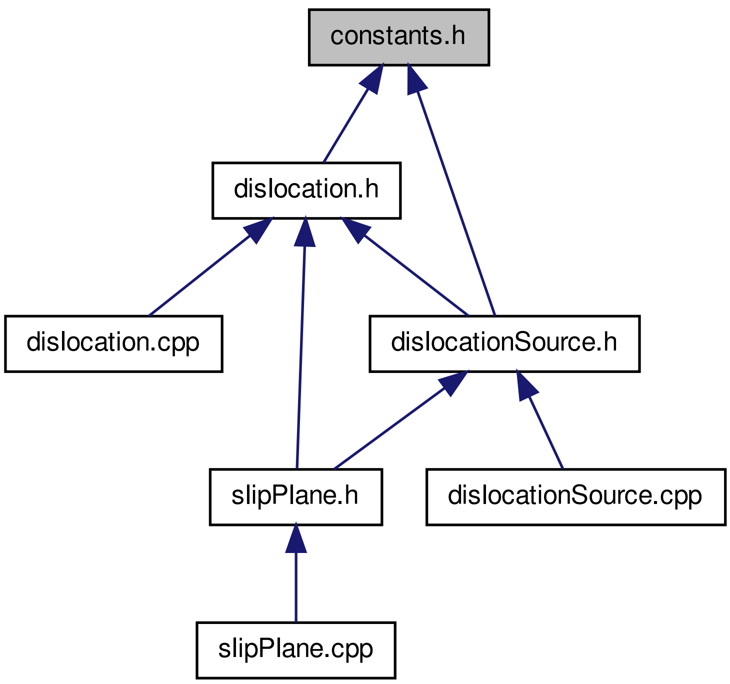
Chapter 6

File Documentation

6.1 constants.h File Reference

Definition of constants used in the program.

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



Macros

- #define PI 3.141592654

    The irrational number pi.
- #define SQRT2 1.414213562

The square root of 2.
- #define SQRT3 1.732050808

  The square root of 3.
- #define SQRT5 2.236067978

  The square root of 5.

6.1.1 Detailed Description

Definition of constants used in the program.

Author

Adhish Majumdar

Version

0.0

Date

26/04/2013

This file defines the values of various constants used in the program.

Definition in file constants.h.

6.1.2 Macro Definition Documentation

6.1.2.1 #define PI 3.141592654

The irrational number pi.

Definition at line 16 of file constants.h.

6.1.2.2 #define SQRT2 1.414213562

The square root of 2.

Definition at line 21 of file constants.h.

6.1.2.3 #define SQRT3 1.732050808

The square root of 3.

Definition at line 26 of file constants.h.

6.1.2.4 #define SQRT5 2.236067978

The square root of 5.

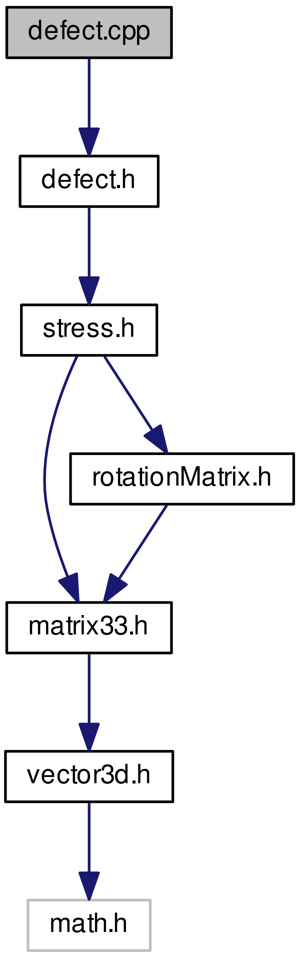
Definition at line 31 of file constants.h.

6.2 defect.cpp File Reference

Definition of member functions of the Defect class.

```
#include "defect.h"
```

Include dependency graph for defect.cpp:



6.2.1 Detailed Description

Definition of member functions of the Defect class.

Author

Adhish Majumdar

Version

0.0

Date

03/06/2013

This file defines the member functions of the Defect class representing a single defect in the simulation.

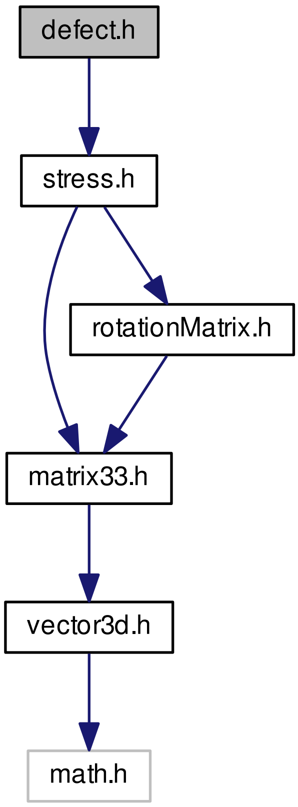
Definition in file defect.cpp.

6.3 defect.h File Reference

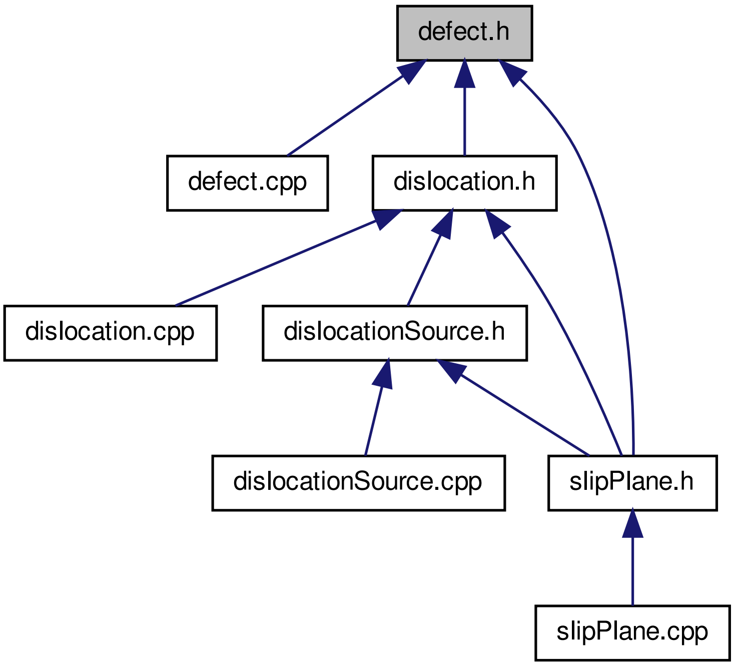
Definition of the Defect class.

`#include "stress.h"`

Include dependency graph for defect.h:



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



Data Structures

- class Defect

    Class Defect representing a generic defect in a material.

6.3.1 Detailed Description

Definition of the Defect class.

Author

Adhish Majumdar

Version

0.0

Date

03/06/2013

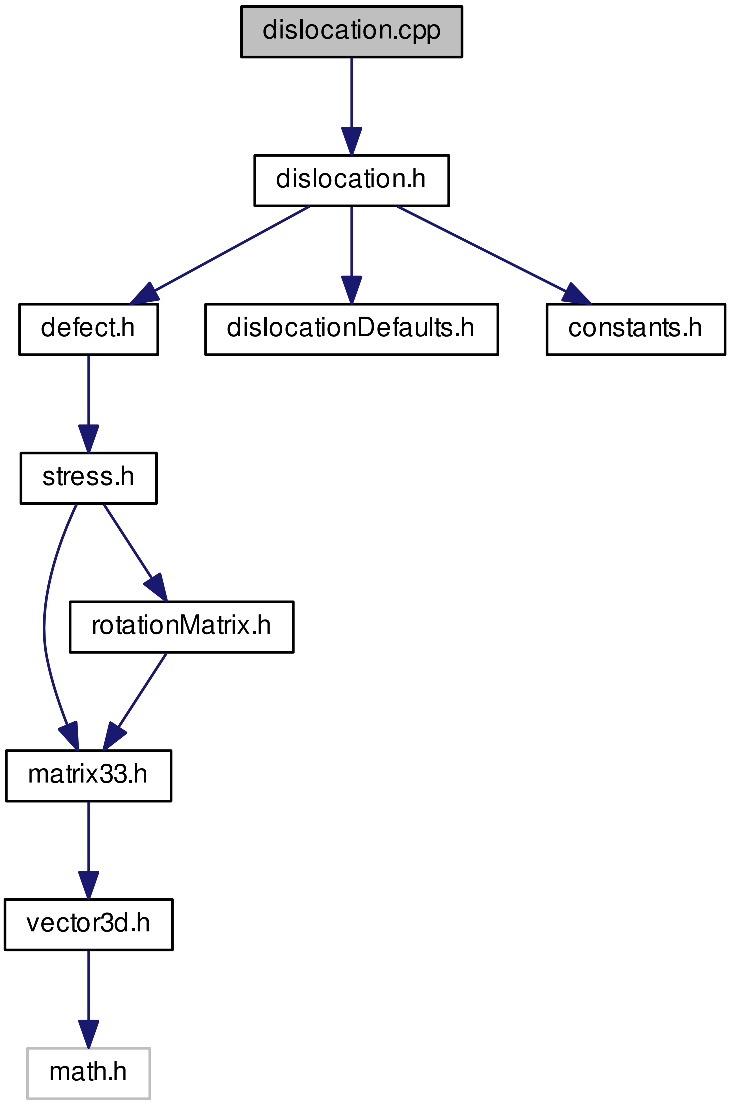
This file defines the Defect class representing an defect in the simulation. This is simply a generic description class with virtual functions. Later classes like dislocations, precipitates, boundaries etc will inherit from this class.

Definition in file defect.h.

6.4 dislocation.cpp File Reference

Definition of constructors and member functions of the Dislocation class.

`#include "dislocation.h"`
Include dependency graph for dislocation.cpp:



6.4.1 Detailed Description

Definition of constructors and member functions of the Dislocation class.

Author

Adhish Majumdar

Version

0.0

Date

03/06/2013

This file defines the constructors and member functions of the Dislocation class. This class inherits from the Defect class.

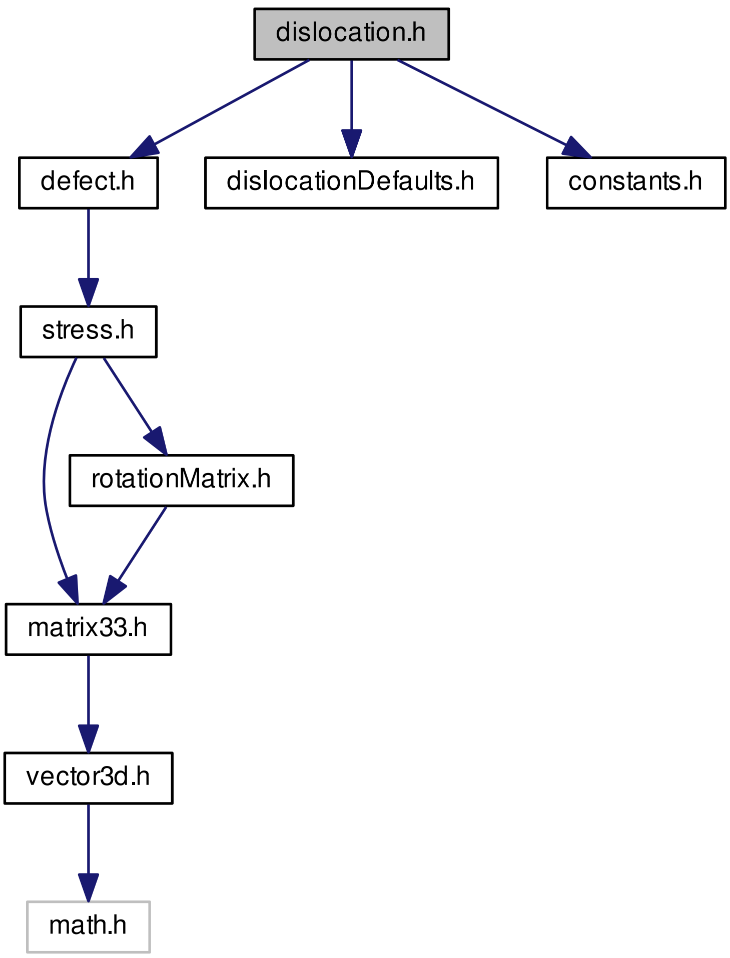
Definition in file dislocation.cpp.

6.5 dislocation.h File Reference

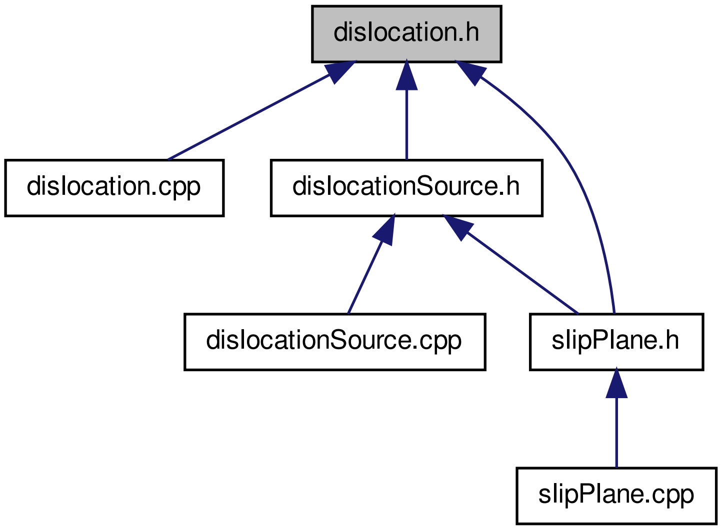
Definition of the Dislocation class.

```
#include "defect.h"
#include "dislocationDefaults.h"
#include "constants.h"
```

Include dependency graph for dislocation.h:



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



Data Structures

- class Dislocation

  Dislocation class representing a dislocation in the simulation.

6.5.1 Detailed Description

Definition of the Dislocation class.

Author

Adhish Majumdar

Version

0.0

Date

03/06/2013

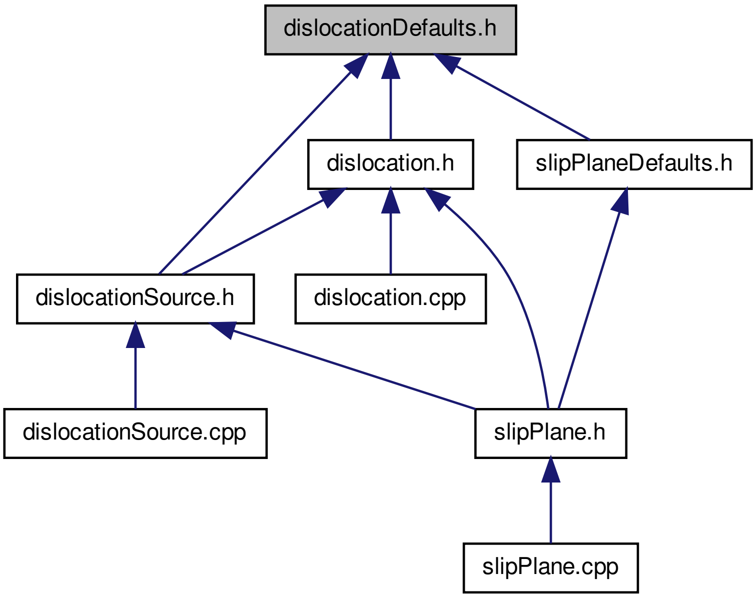
This file defines the Dislocation class representing a dislocation in the simulation. This class inherits from the Defect class.

Definition in file dislocation.h.

6.6 dislocationDefaults.h File Reference

Definition of certain default values for members of the Dislocation class.

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



Macros

- #define DEFAULT_POSITION_0 0.0

  Default value of the position vector x-coordinate.
- #define DEFAULT_POSITION_1 0.0

  Default value of the position vector y-coordinate.
- #define DEFAULT_POSITION_2 0.0

  Default value of the position vector z-coordinate.
- #define DEFAULT_BURGERS_MAGNITUDE 5.0e-09

  Default value of the magnitude of the Burgers vector.
- #define DEFAULT_BURGERS_0 1.0

  Default value of the Burgers vector x-coordinate.
- #define DEFAULT_BURGERS_1 1.0

  Default value of the Burgers vector y-coordinate.
- #define DEFAULT_BURGERS_2 0.0

  Default value of the Burgers vector z-coordinate.
- #define DEFAULT_LINEVECTOR_0 1.0

  Default value of the line vector x-coordinate.
- #define DEFAULT_LINEVECTOR_1 -1.0

  Default value of the line vector y-coordinate.
- #define DEFAULT_LINEVECTOR_2 -2.0

  Default value of the line vector z-coordinate.

6.6.1 Detailed Description

Definition of certain default values for members of the Dislocation class.

Author

Adhish Majumdar

Version

0.0

Date

26/04/2013

This file defines some default values for members of the Dislocation class representing a dislocation in the simulation.

Definition in file dislocationDefaults.h.

6.6.2 Macro Definition Documentation

6.6.2.1 #define DEFAULT_BURGERS_0 1.0

Default value of the Burgers vector x-coordinate.

Definition at line 34 of file dislocationDefaults.h.

6.6.2.2 #define DEFAULT_BURGERS_1 1.0

Default value of the Burgers vector y-coordinate.

Definition at line 38 of file dislocationDefaults.h.

6.6.2.3 #define DEFAULT_BURGERS_2 0.0

Default value of the Burgers vector z-coordinate.

Definition at line 42 of file dislocationDefaults.h.

6.6.2.4 #define DEFAULT_BURGERS_MAGNITUDE 5.0e-09

Default value of the magnitude of the Burgers vector.

Definition at line 29 of file dislocationDefaults.h.

6.6.2.5 #define DEFAULT_LINEVECTOR_0 1.0

Default value of the line vector x-coordinate.

Definition at line 47 of file dislocationDefaults.h.

6.6.2.6 #define DEFAULT_LINEVECTOR_1 -1.0

Default value of the line vector y-coordinate.

Definition at line 51 of file dislocationDefaults.h.

6.6.2.7 #define DEFAULT_LINEVECTOR_2 -2.0

Default value of the line vector z-coordinate.

Definition at line 55 of file dislocationDefaults.h.

6.6.2.8 #define DEFAULT_POSITION_0 0.0

Default value of the position vector x-coordinate.

Definition at line 16 of file dislocationDefaults.h.

6.6.2.9 #define DEFAULT_POSITION_1 0.0

Default value of the position vector y-coordinate.

Definition at line 20 of file dislocationDefaults.h.

6.6.2.10 #define DEFAULT_POSITION_2 0.0

Default value of the position vector z-coordinate.

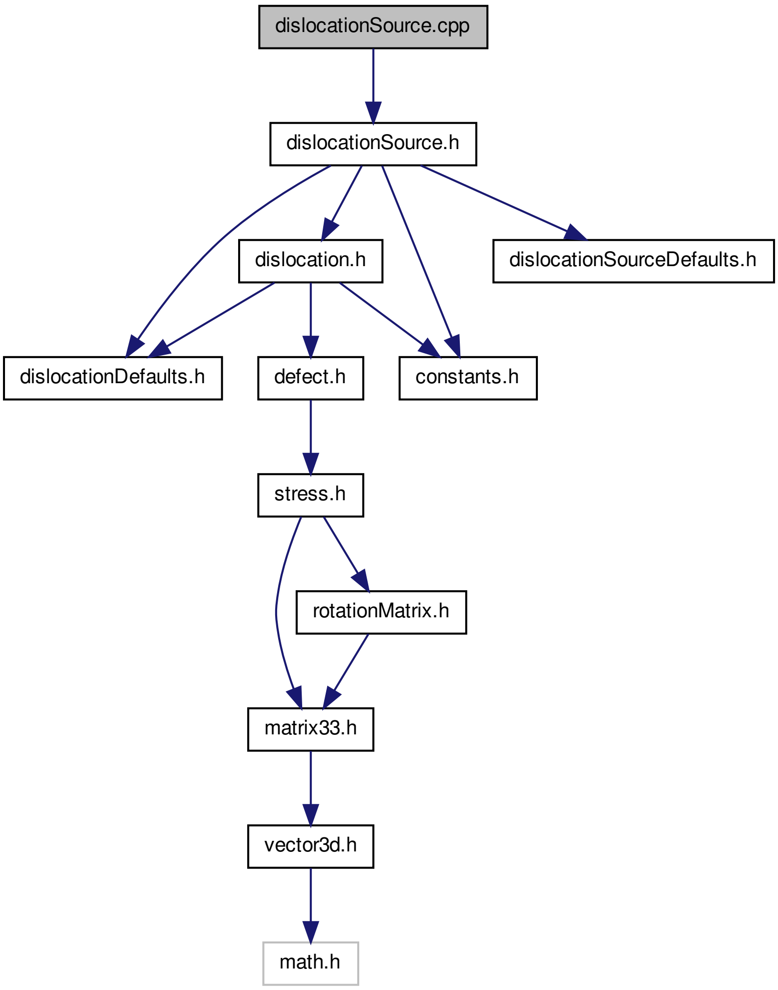
Definition at line 24 of file dislocationDefaults.h.

6.7 dislocationSource.cpp File Reference

Definition of the member functions of the DislocationSource class.

`#include "dislocationSource.h"`

Include dependency graph for dislocationSource.cpp:



6.7.1 Detailed Description

Definition of the member functions of the DislocationSource class.

Author

Adhish Majumdar

Version

0.0

Date

27/05/2013

This file defines the member functions of the DislocationSource class representing a source of dislocations in the simulation. This class inherits from the Defect class. This object is basically the representation of a Frank-Read source emitting dislocation dipoles. When the dislocation source experiences a shear stress greater than a critical value for a certain amount of time (or number of iterations), it emits a dislocation dipole with a length that is a function of the applied stress.

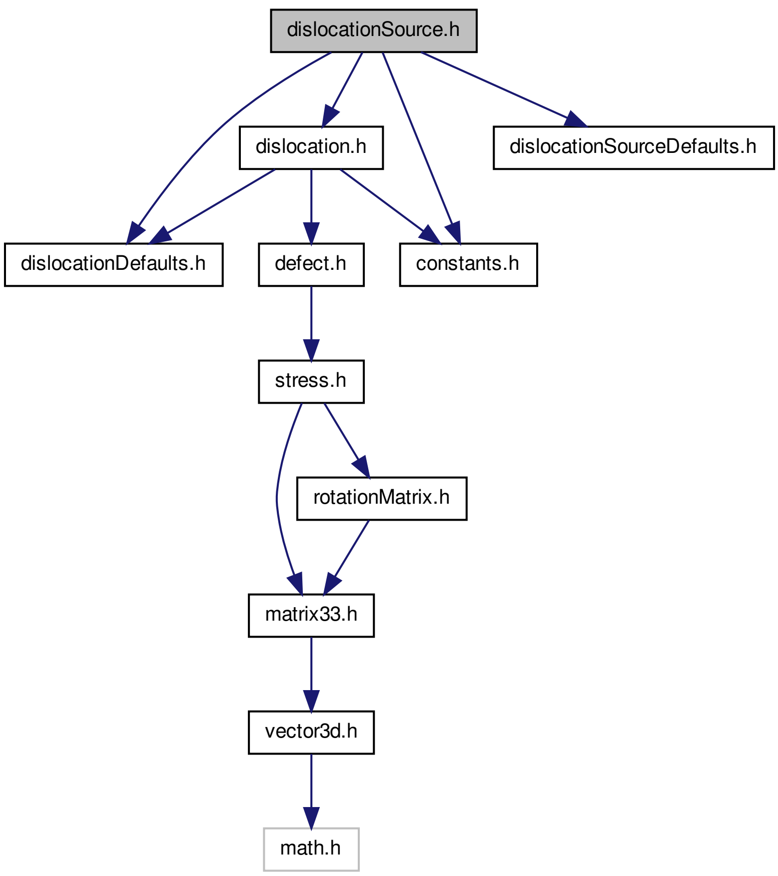
Definition in file dislocationSource.cpp.

6.8 dislocationSource.h File Reference

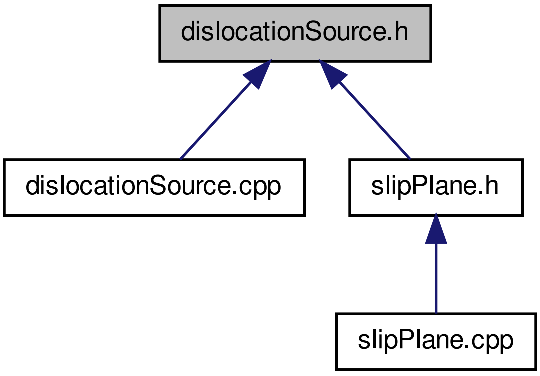
Definition of the DislocationSource class.

```
#include "dislocation.h"
#include "constants.h"
#include "dislocationDefaults.h"
#include "dislocationSourceDefaults.h"
```

Include dependency graph for dislocationSource.h:



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



Data Structures

- class DislocationSource

  DislocationSource class representing a source of dislocations in the simulation.

6.8.1 Detailed Description

Definition of the DislocationSource class.

Author

Adhish Majumdar

Version

0.0

Date

27/05/2013

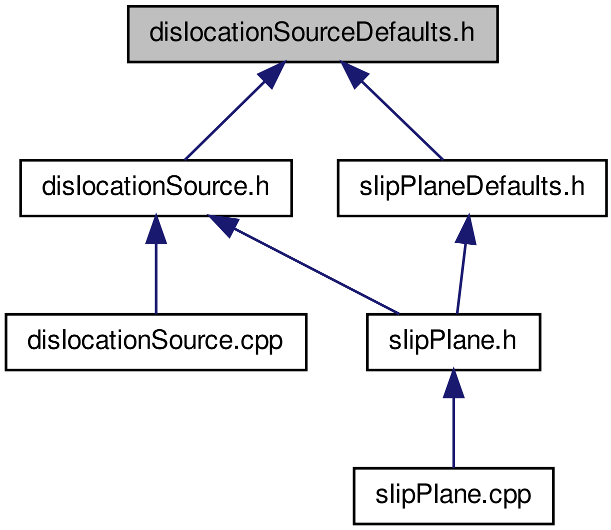
This file defines the DislocationSource class representing a source of dislocations in the simulation. This class inherits from the Defect class. This object is basically the representation of a Frank-Read source emitting dislocation dipoles. When the dislocation source experiences a shear stress greater than a critical value for a certain amount of time (or number of iterations), it emits a dislocation dipole with a length that is a function of the applied stress.

Definition in file dislocationSource.h.

6.9 dislocationSourceDefaults.h File Reference

Definition of certain default values for members of the DislocationSource class.

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



Macros

- #define DEFAULT_TAU_CRITICAL 1.0e09

    Default value of the critical shear stress for a dislocation source to emit a dipole.
- #define DEFAULT_NITERATIONS 10

    Default value of the number of iterations required for a dislocation source to emit a dipole.

6.9.1 Detailed Description

Definition of certain default values for members of the DislocationSource class.

Author

Adhish Majumdar

Version

0.0

Date

02/05/2013

This file defines some default values for members of the DislocationSource class representing a dislocation dipole source in the simulation.

Definition in file dislocationSourceDefaults.h.

6.9.2 Macro Definition Documentation

6.9.2.1 #define DEFAULT_NITERATIONS 10

Default value of the number of iterations required for a dislocation source to emit a dipole.

The dislocation source must experience a shear stress greater than the critical value in order to emit a dipole. This time is expressed in terms of the number of iterations here.

Definition at line 23 of file dislocationSourceDefaults.h.

6.9.2.2 #define DEFAULT_TAU_CRITICAL 1.0e09

Default value of the critical shear stress for a dislocation source to emit a dipole.

Default value of the critical shear stress for a dislocation source to emit a dipole. The number is expressed in Pa.

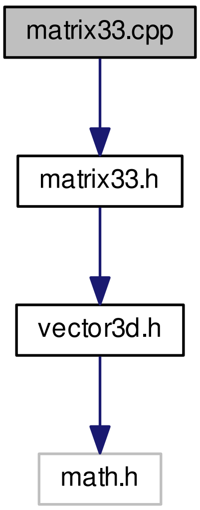
Definition at line 17 of file dislocationSourceDefaults.h.

6.10 mainpage.dox File Reference

6.11 matrix33.cpp File Reference

Definition of the member functions and operators of the Matrix33 class.

`#include "matrix33.h"`
Include dependency graph for matrix33.cpp:



6.11.1 Detailed Description

Definition of the member functions and operators of the Matrix33 class.

Author

Adhish Majumdar

Version

0.0

Date

22/04/2013

This file defines the member functions and operators of the Matrix33 class representing a 3x3 matrix in the simulation.

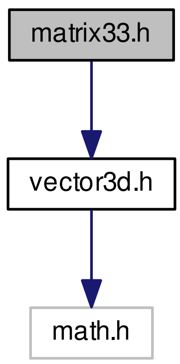
Definition in file matrix33.cpp.

6.12 matrix33.h File Reference

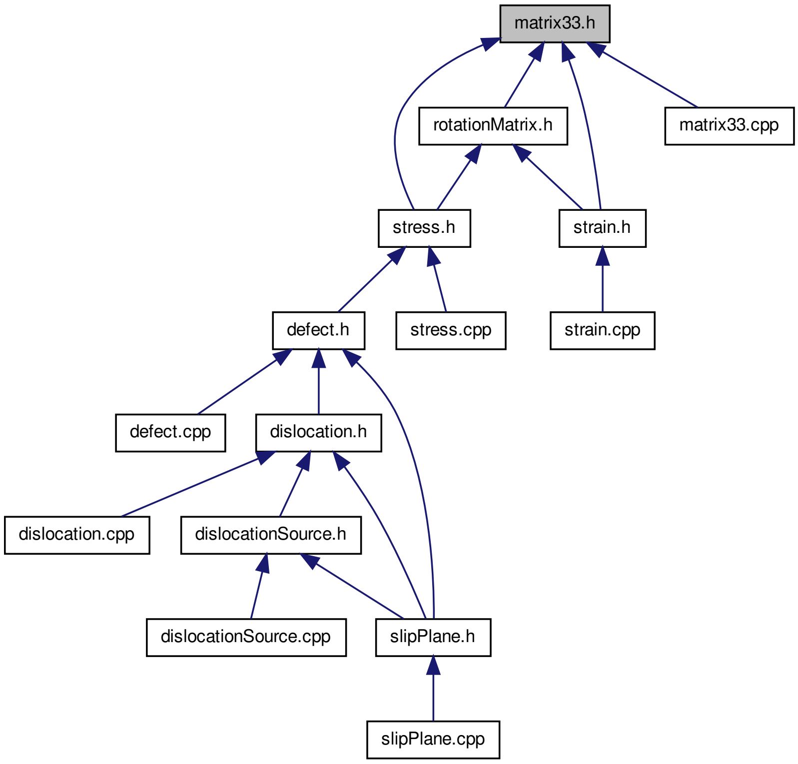
Definition of the Matrix33 class.

```
#include "vector3d.h"
```

Include dependency graph for matrix33.h:



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



Data Structures

- class Matrix33

  Matrix33 class representing a 3x3 square matrix.

6.12.1 Detailed Description

Definition of the Matrix33 class.

Author

Adhish Majumdar

Version

0.0

Date

22/04/2013

This file defines the Matrix33 class representing a 3x3 matrix in the simulation.

Definition in file matrix33.h.

6.13 rotationMatrix.cpp File Reference

Definition of the RotationMatrix class member functions.

6.13.1 Detailed Description

Definition of the RotationMatrix class member functions.

Author

Adhish Majumdar

Version

0.0

Date

25/04/2013

This file defines member functions of the RotationMatrix class for carrying out 3D rotations and axes transformations.

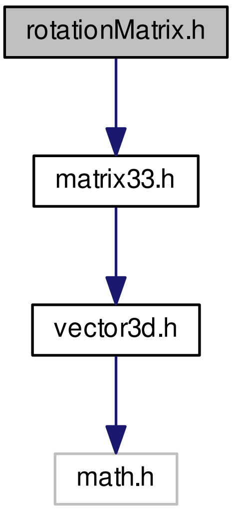
Definition in file rotationMatrix.cpp.

6.14 rotationMatrix.h File Reference

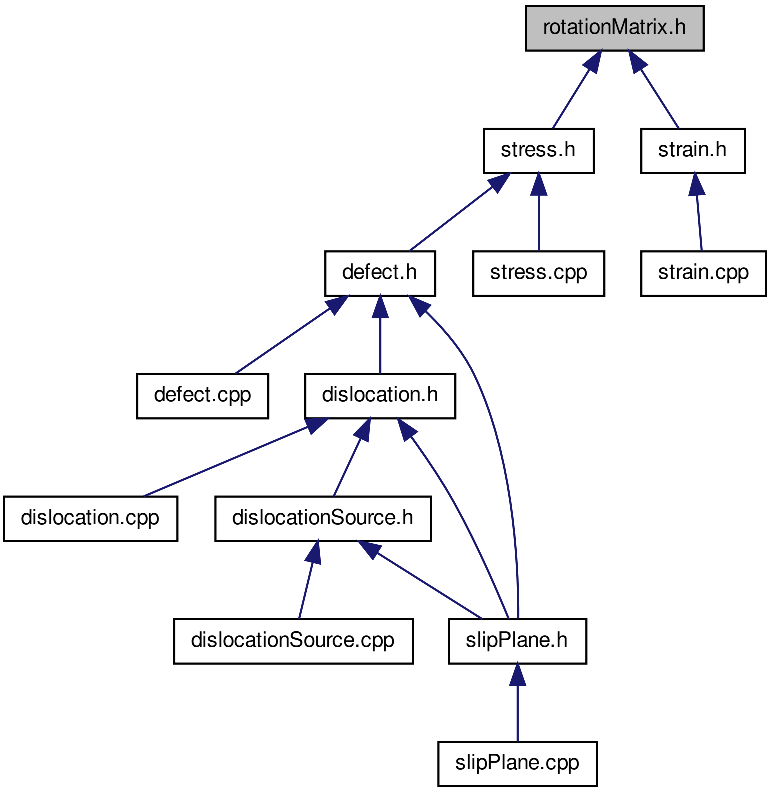
Definition of the RotationMatrix class.

```
#include "matrix33.h"
```

Include dependency graph for rotationMatrix.h:



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



Data Structures

- class RotationMatrix

  RotationMatrix class to represent a rotation matrix.

6.14.1 Detailed Description

Definition of the RotationMatrix class.

Author

Adhish Majumdar

Version

0.0

Date

25/04/2013

This file defines the RotationMatrix class for carrying out 3D rotations and axes transformations.

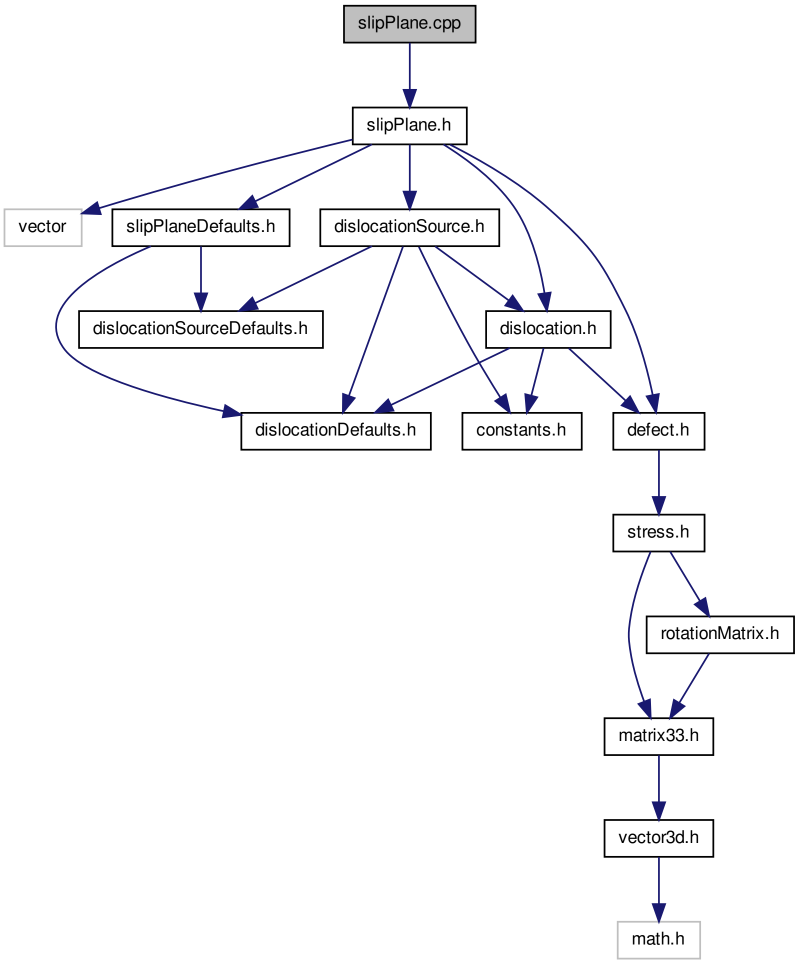
Definition in file rotationMatrix.h.

6.15 slipPlane.cpp File Reference

Definition of the member functions of the SlipPlane class.

`#include "slipPlane.h"`

Include dependency graph for slipPlane.cpp:



6.15.1 Detailed Description

Definition of the member functions of the SlipPlane class.

Author

Adhish Majumdar

Version

0.0

Date

03/06/2013

This file defines the member functions of the SlipPlane class.

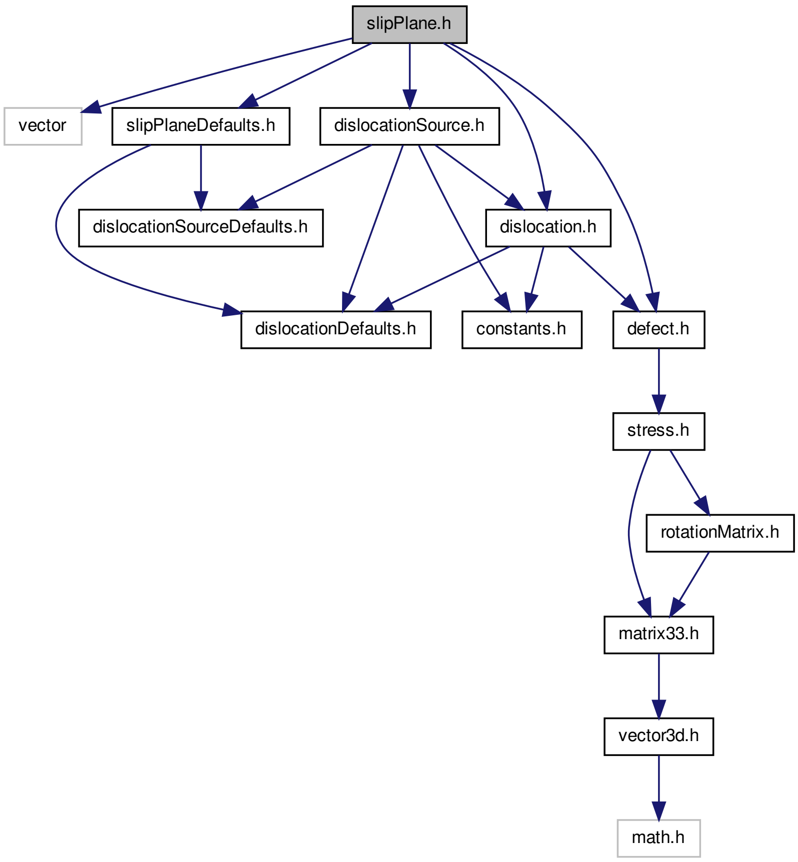
Definition in file slipPlane.cpp.

6.16 slipPlane.h File Reference

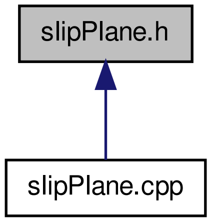
Definition of the SlipPlane class.

```
#include <vector>
#include "slipPlaneDefaults.h"
#include "defect.h"
#include "dislocation.h"
#include "dislocationSource.h"
```

Include dependency graph for slipPlane.h:



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



Data Structures

- class SlipPlane

  SlipPlane class representing a slip plane in the simulation.

6.16.1 Detailed Description

Definition of the SlipPlane class.

Author

Adhish Majumdar

Version

0.0

Date

03/06/2013

This file defines the SlipPlane class representing a slip plane in the simulation.

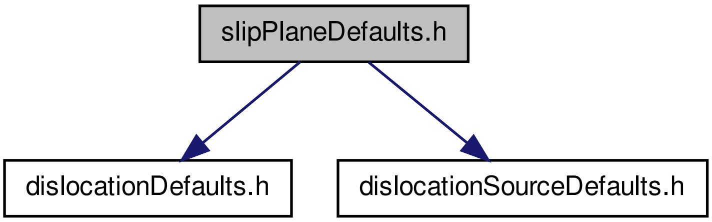
Definition in file slipPlane.h.

6.17 slipPlaneDefaults.h File Reference

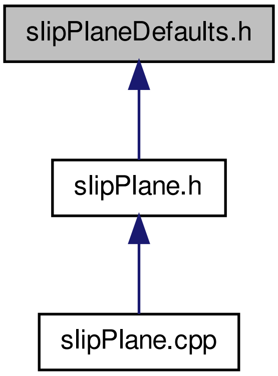
Definition of certain default values for members of the SlipPlane class.

```
#include "dislocationDefaults.h"
#include "dislocationSourceDefaults.h"
```

Include dependency graph for slipPlaneDefaults.h:



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



Macros

- #define DEFAULT_SLIPPLANE_POSITION_0 0.0

  Default value of the position vector x-coordinate.
- #define DEFAULT_SLIPPLANE_POSITION_1 0.0

  Default value of the position vector y-coordinate.
- #define DEFAULT_SLIPPLANE_POSITION_2 0.0

  Default value of the position vector z-coordinate.
- #define DEFAULT_SLIPPLANE_NORMALVECTOR_0 1.0

  Default value of the normal vector x-coordinate.
- #define DEFAULT_SLIPPLANE_NORMALVECTOR_1 1.0

  Default value of the normal vector y-coordinate.
- #define DEFAULT_SLIPPLANE_NORMALVECTOR_2 1.0

  Default value of the normal vector z-coordinate.
- #define DEFAULT_SLIPPLANE_EXTREMITY1_0 5.0e-06

Default value of the position vector of extremity 1 x-coordinate.
- #define DEFAULT_SLIPPLANE_EXTREMITY1_1 0.0

  Default value of the position vector of extremity 1 y-coordinate.
- #define DEFAULT_SLIPPLANE_EXTREMITY1_2 0.0

  Default value of the position vector of extremity 1 z-coordinate.
- #define DEFAULT_SLIPPLANE_EXTREMITY2_0 0.0

  Default value of the position vector of extremity 2 x-coordinate.
- #define DEFAULT_SLIPPLANE_EXTREMITY2_1 5.0e-6

  Default value of the position vector of extremity 2 y-coordinate.
- #define DEFAULT_SLIPPLANE_EXTREMITY2_2 0.0

  Default value of the position vector of extremity 2 z-coordinate.

6.17.1 Detailed Description

Definition of certain default values for members of the SlipPlane class.

Author

Adhish Majumdar

Version

0.0

Date

31/05/2013

This file defines some default values for members of the SlipPlane class representing a slip plane in the simulation.

Definition in file slipPlaneDefaults.h.

6.17.2 Macro Definition Documentation

6.17.2.1 #define DEFAULT_SLIPPLANE_EXTREMITY1_0 5.0e-06

Default value of the position vector of extremity 1 x-coordinate.

Definition at line 50 of file slipPlaneDefaults.h.

6.17.2.2 #define DEFAULT_SLIPPLANE_EXTREMITY1_1 0.0

Default value of the position vector of extremity 1 y-coordinate.

Definition at line 55 of file slipPlaneDefaults.h.

6.17.2.3 #define DEFAULT_SLIPPLANE_EXTREMITY1_2 0.0

Default value of the position vector of extremity 1 z-coordinate.

Definition at line 60 of file slipPlaneDefaults.h.

6.17.2.4 #define DEFAULT_SLIPPLANE_EXTREMITY2_0 0.0

Default value of the position vector of extremity 2 x-coordinate.

Definition at line 65 of file slipPlaneDefaults.h.

6.17.2.5 #define DEFAULT_SLIPPLANE_EXTREMITY2_1 5.0e-6

Default value of the position vector of extremity 2 y-coordinate.

Definition at line 70 of file slipPlaneDefaults.h.

6.17.2.6 #define DEFAULT_SLIPPLANE_EXTREMITY2_2 0.0

Default value of the position vector of extremity 2 z-coordinate.

Definition at line 75 of file slipPlaneDefaults.h.

6.17.2.7 #define DEFAULT_SLIPPLANE_NORMALVECTOR_0 1.0

Default value of the normal vector x-coordinate.

Definition at line 35 of file slipPlaneDefaults.h.

6.17.2.8 #define DEFAULT_SLIPPLANE_NORMALVECTOR_1 1.0

Default value of the normal vector y-coordinate.

Definition at line 40 of file slipPlaneDefaults.h.

6.17.2.9 #define DEFAULT_SLIPPLANE_NORMALVECTOR_2 1.0

Default value of the normal vector z-coordinate.

Definition at line 45 of file slipPlaneDefaults.h.

6.17.2.10 #define DEFAULT_SLIPPLANE_POSITION_0 0.0

Default value of the position vector x-coordinate.

Definition at line 20 of file slipPlaneDefaults.h.

6.17.2.11 #define DEFAULT_SLIPPLANE_POSITION_1 0.0

Default value of the position vector y-coordinate.

Definition at line 25 of file slipPlaneDefaults.h.

6.17.2.12 #define DEFAULT_SLIPPLANE_POSITION_2 0.0

Default value of the position vector z-coordinate.

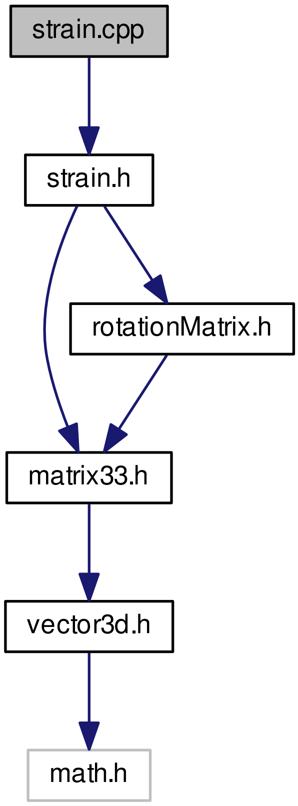
Definition at line 30 of file slipPlaneDefaults.h.

6.18 strain.cpp File Reference

Definition of the member functions if the Strain class.

```
#include "strain.h"
```

Include dependency graph for strain.cpp:



6.18.1 Detailed Description

Definition of the member functions if the Strain class.

Author

Adhish Majumdar

Version

0.0

Date

22/04/2013

This file defines the member functions of the Strain class for the strain tensor.

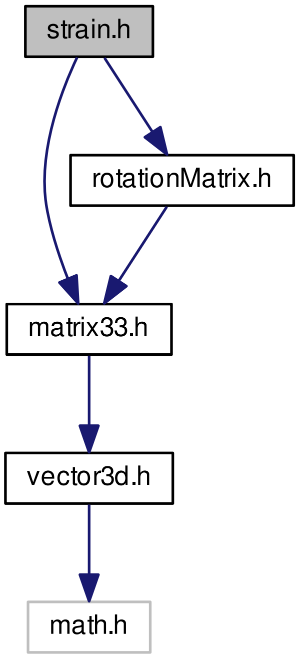
Definition in file strain.cpp.

6.19 strain.h File Reference

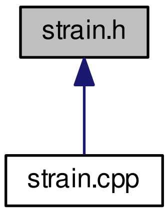
Definition of the Strain class.

```
#include "matrix33.h"
#include "rotationMatrix.h"
```

Include dependency graph for strain.h:



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



Data Structures

- class Strain

Strain class to represent the strain tensor.

6.19.1 Detailed Description

Definition of the Strain class.

Author

Adhish Majumdar

Version

0.0

Date

25/04/2013

This file defines the Strain class for the strain tensor.

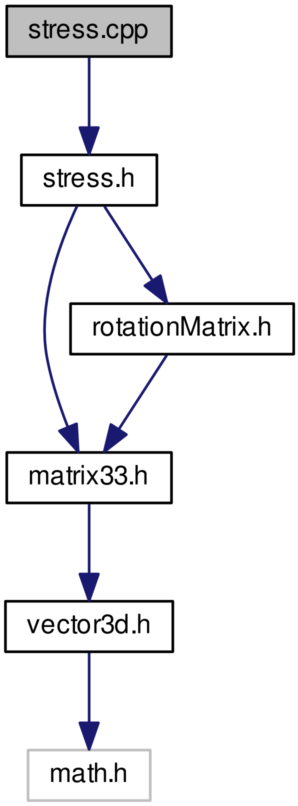
Definition in file strain.h.

6.20 stress.cpp File Reference

Definition of the member functions if the Stress class.

```
#include "stress.h"
```

Include dependency graph for stress.cpp:



6.20.1 Detailed Description

Definition of the member functions if the Stress class.

Author

Adhish Majumdar

Version

0.0

Date

22/04/2013

This file defines the member functions of the Stress class for the stress tensor.

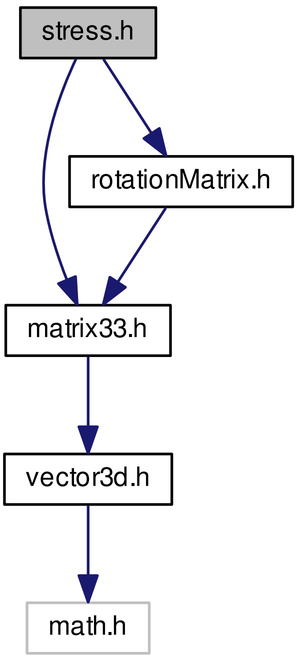
Definition in file stress.cpp.

6.21 stress.h File Reference

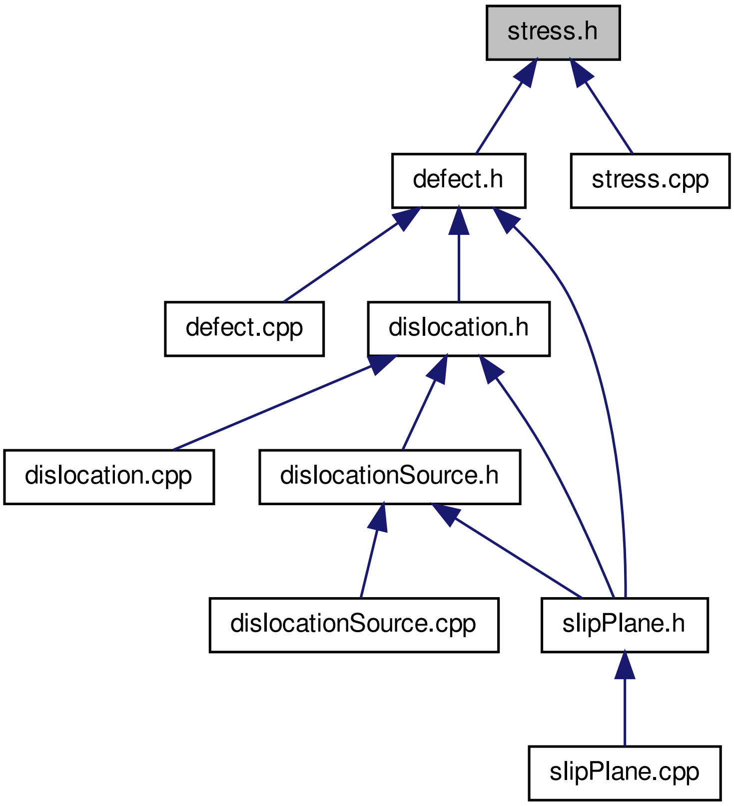
Definition of the Stress class.

```
#include "matrix33.h"
#include "rotationMatrix.h"
```

Include dependency graph for stress.h:



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



Data Structures

- class Stress

  Stress class to represent the stress tensor.

6.21.1 Detailed Description

Definition of the Stress class.

Author

Adhish Majumdar

Version

0.0

Date

22/04/2013

This file defines the Stress class for the stress tensor.

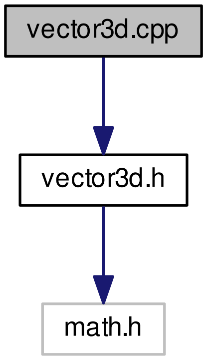
Definition in file stress.h.

6.22 vector3d.cpp File Reference

Definition of member functions and operators of the Vector3d class.

```
#include "vector3d.h"
```

Include dependency graph for vector3d.cpp:



6.22.1 Detailed Description

Definition of member functions and operators of the Vector3d class.

Author

Adhish Majumdar

Version

0.0

Date

29/04/2013

This file defines the member functions and operators of the Vector3d class representing a single 3-dimensional vector in the simulation.

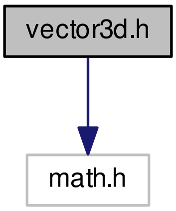
Definition in file vector3d.cpp.

6.23 vector3d.h File Reference

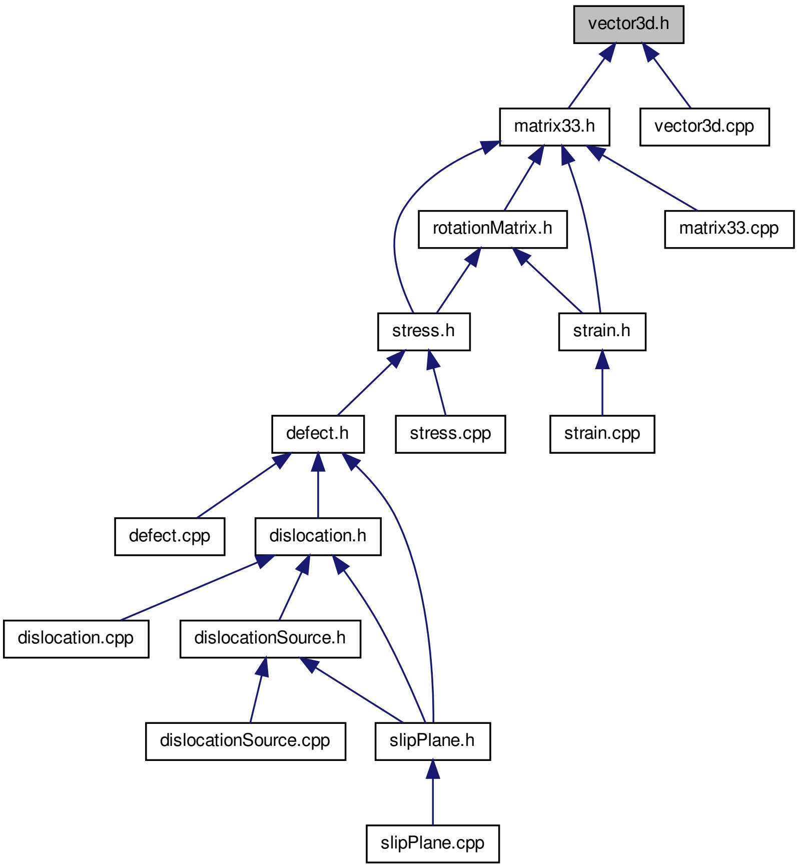
Definition of the Vector3d class.

`#include <math.h>`

Include dependency graph for vector3d.h:



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



Data Structures

- class Vector3d

  Vector3d class representing a single 3-dimensional vector in the simulation.

6.23.1 Detailed Description

Definition of the Vector3d class.

Author

Adhish Majumdar

Version

0.0

Date

29/04/2013

This file defines the Vector3d class representing a single 3-dimensional vector in the simulation.

Definition in file vector3d.h.

Index