DD2.5D

0

Generated by Doxygen 1.8.2

Sat Apr 20 2013 16:35:35

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

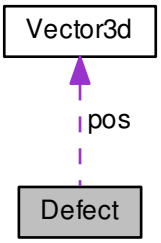
Data Structure Documentation

4.1 Defect Class Reference

Class Defect representing a generic defect in a material.

`#include <defect.h>`

Collaboration diagram for Defect:



Public Member Functions

- Defect ()
- Defect (double x, double y, double z)
- Defect (double *p)
- void setPosition (double *a)
- void setPosition (double x, double y, double z)
- void setX (double x)
- void setY (double y)
- void setZ (double z)
- double * getPosition ()
- void getPosition (double *a)
- double getX ()
- double getY ()
- double getZ ()
- virtual Matrix33 stressField (Vector3d p)

Protected Attributes

- Vector3d pos

4.1.1 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 24 of file defect.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Defect::Defect ( )

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

Definition at line 8 of file defect.cpp.

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

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

Constructor specifying the position.

Parameters

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

Definition at line 22 of file defect.cpp.

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

4.1.2.3 Defect::Defect (double * *p*)

Constructor specifying the position.

Parameters

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

Definition at line 33 of file defect.cpp.

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

4.1.3 Member Function Documentation

4.1.3.1 double * Defect::getPosition ( )

Returns in an array the position.

Definition at line 93 of file defect.cpp.

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

4.1.3.2 void Defect::getPosition (double * *a*)

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 102 of file defect.cpp.

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

4.1.3.3 double Defect::getX ( )

Returns the X-coordinate of the defect.

Returns

X-coordinate of the defect.

Returns the X-coordinate of the defect.

Definition at line 110 of file defect.cpp.

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

4.1.3.4 double Defect::getY ( )

Returns the Y-coordinate of the defect.

Returns

Y-coordinate of the defect.

Returns the Y-coordinate of the defect.

Definition at line 118 of file defect.cpp.

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

4.1.3.5 double Defect::getZ ( )

Returns the Z-coordinate of the defect.

Returns

Z-coordinate of the defect.

Returns the Z-coordinate of the defect.

Definition at line 126 of file defect.cpp.

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

4.1.3.6 void Defect::setPosition (double * *a*)

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 44 of file defect.cpp.

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

4.1.3.7 void Defect::setPosition (double *x*, double *y*, double *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 55 of file defect.cpp.

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

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

Sets the X-coordinate of the defect.

Parameters

x	X-coordinate of the defect.

Definition at line 66 of file defect.cpp.

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

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

Sets the Y-coordinate of the defect.

Parameters

y	Y-coordinate of the defect.

Definition at line 75 of file defect.cpp.

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

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

Sets the Z-coordinate of the defect.

Parameters

z	Z-coordinate of the defect.

Definition at line 84 of file defect.cpp.

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

4.1.3.11 virtual Matrix33 Defect::stressField (Vector3d *p*) [inline], [virtual]

Returns the value of the stress field of the given defect at the position given by the argument.

Parameters

p	Position vector of the the point where the stress field is to be calculated.

Returns

Stress field value at the position p.

Definition at line 113 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.
  Matrix33 r;
  return (r);
}
```

4.1.4 Field Documentation

4.1.4.1 Vector3d Defect::pos [protected]

Position of the defect in 2D space.

Definition at line 30 of file defect.h.

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

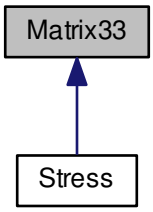
- defect.h
- defect.cpp

4.2 Matrix33 Class Reference

Matrix33 class representing a 3x3 square matrix.

`#include <matrix33.h>`

Inheritance diagram for Matrix33:



Public Member Functions

- Matrix33 ()
- Matrix33 (double **a)
- Matrix33 (Vector3d a)
- Matrix33 (Vector3d a, Vector3d b)
- void setValue (int row, int column, double value)
- double getValue (int row, int column)

- Matrix33 operator+ (const Matrix33 &) const
- void operator+= (const Matrix33 &)
- Matrix33 operator- (const Matrix33 &) const
- void operator-= (const Matrix33 &)
- Matrix33 operator∗ (const double &) const
- void operator∗= (const double &)
- Matrix33 operator∗ (const Matrix33 &) const
- void operator∗= (const Matrix33 &)
- Vector3d operator∗ (const Vector3d &) const
- Matrix33 operator^ () const
- double operator∼ () const
- Matrix33 operator! () const

Protected Attributes

- double x [3][3]

4.2.1 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.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Matrix33::Matrix33 ( )

Default constructor.

Definition at line 7 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;
        }
    }
}
```

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

Constructor with the values provided in a 3x3 matrix.

Parameters

a	Pointer to the two-dimensional 3x3 array.

Definition at line 24 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];
        }
    }
}
```

4.2.2.3 Matrix33::Matrix33 (Vector3d *a*)

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

Parameters

a	The vector whose dyadic product results in the matrix.

Definition at line 41 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];
        }
    }
}
```

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

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

Parameters

a	First vector.
b	Second vector.

Definition at line 59 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];
        }
    }
}
```

4.2.3 Member Function Documentation

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

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

Parameters

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

Definition at line 95 of file matrix33.cpp.

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

  return (0.0);
}
```

4.2.3.2 Matrix33 Matrix33::operator! ( ) const

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

Definition at line 319 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;

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

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

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

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

  return (r);
}
```

4.2.3.3 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.

Definition at line 190 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);
}
```

4.2.3.4 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.

Definition at line 227 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);
}
```

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

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

Definition at line 265 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);
}
```

4.2.3.6 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 210 of file matrix33.cpp.

```
{
  int i, j;

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

4.2.3.7 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 251 of file matrix33.cpp.

```
{
  Matrix33* r = new Matrix33;

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

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

4.2.3.8 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.

Definition at line 114 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);
}
```

4.2.3.9 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 134 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];
        }
    }
}
```

4.2.3.10 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.

Definition at line 152 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);
}
```

4.2.3.11 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 172 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];
        }
    }
}
```

4.2.3.12 Matrix33 Matrix33::operator^ ( ) const

Returns in a new matrix the transpose of the current matrix.

Definition at line 285 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);
}
```

4.2.3.13 double Matrix33::operator~ ( ) const

Returns the determinant of the current matrix.

Definition at line 304 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);
}
```

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

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

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 79 of file matrix33.cpp.

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

4.2.4 Field Documentation

4.2.4.1 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:

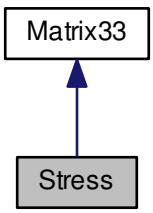
- matrix33.h
- matrix33.cpp

4.3 Stress Class Reference

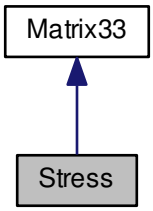
Stress class to represent the stress tensor.

`#include <stress.h>`

Inheritance diagram for Stress:



Collaboration diagram for Stress:



Public Member Functions

- Stress ()
- Stress (double *principal, double *shear)
- void populateMatrix ()
- double * getPrincipalStresses ()
- double * getShearStresses ()
- Stress rotate (Matrix33 alpha)
- void setValue (int row, int column, double value)
- double getValue (int row, int column)
- Matrix33 operator+ (const Matrix33 &) const
- void operator+= (const Matrix33 &)
- Matrix33 operator- (const Matrix33 &) const
- void operator-= (const Matrix33 &)
- Matrix33 operator* (const double &) const
- Matrix33 operator* (const Matrix33 &) const
- Vector3d operator* (const Vector3d &) const
- void operator*= (const double &)
- void operator*= (const Matrix33 &)
- Matrix33 operator^ () const
- double operator~ () const
- Matrix33 operator! () const

Protected Attributes

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

4.3.1 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 20 of file stress.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Stress::Stress ( )

Default constructor.

Fills the stress tensor with zeros.

Definition at line 8 of file stress.cpp.

```
{
  int i, j;

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

  this->populateMatrix ();
}
```

4.3.2.2 Stress::Stress (double * *principal*, double * *shear*)

Constructor specifying the principal and shear stresses.

Parameters

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

Definition at line 26 of file stress.cpp.

```
{
  int i;

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

  this->populateMatrix ();
}
```

4.3.3 Member Function Documentation

4.3.3.1 double * Stress::getPrincipalStresses ( )

Returns a 3-member array with the principal stresses.

Returns

3-member array with the principal stresses.

Definition at line 57 of file stress.cpp.

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

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

  return (p);
}
```

4.3.3.2 double * Stress::getShearStresses ( )

Returns a 3-member array with the shear stresses.

Returns

3-member array with the shear stresses.

Definition at line 74 of file stress.cpp.

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

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

  return (s);
}
```

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

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

Parameters

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

Definition at line 95 of file matrix33.cpp.

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

  return (0.0);
}
```

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

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

Definition at line 319 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;

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

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

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

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

  return (r);
}
```

4.3.3.5 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.

Definition at line 190 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);
}
```

4.3.3.6 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.

Definition at line 227 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);
}
```

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

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

Definition at line 265 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);
}
```

4.3.3.8 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 210 of file matrix33.cpp.

```
{
  int i, j;

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

4.3.3.9 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 251 of file matrix33.cpp.

```
{
  Matrix33* r = new Matrix33;

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

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

4.3.3.10 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.

Definition at line 114 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);
}
```

4.3.3.11 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 134 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];
        }
    }
}
```

4.3.3.12 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.

Definition at line 152 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);
}
```

4.3.3.13 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 172 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];
        }
    }
}
```

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

Returns in a new matrix the transpose of the current matrix.

Definition at line 285 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);
}
```

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

Returns the determinant of the current matrix.

Definition at line 304 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);
}
```

4.3.3.16 void Stress::populateMatrix ( )

Takes the values in principalStresses and shearStresses and constructs the stress matrix.

Definition at line 42 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];
}
```

4.3.3.17 Stress Stress::rotate (Matrix33 *alpha*)

Rotates the present stress matrix using the rotation matrix supplied and returns the result.

Parameters

alpha	Rotation matrix.

Returns

Rotated stress tensor.

Definition at line 92 of file stress.cpp.

```
{
  Matrix33 alphaT = ^alpha;
  Stress sNew;

  sNew = alpha * (*this) * alphaT;

  return (sNew);
}
```

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

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

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 79 of file matrix33.cpp.

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

4.3.4 Field Documentation

4.3.4.1 double Stress::principalStresses[3] [protected]

The three principal stresses: s11, s22, s33.

Definition at line 26 of file stress.h.

4.3.4.2 double Stress::shearStresses[3] [protected]

The three shear stresses: s12, s13, s23,

Definition at line 30 of file stress.h.

4.3.4.3 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

4.4 Vector3d Class Reference

Vector3d class representing a single 3-dimensional vector in the simulation.

`#include <vector3d.h>`

Public Member Functions

- Vector3d ()
- Vector3d (double *a)
- Vector3d (double a1, double a2, double a3)
- void setValue (int index, double value)
- void setVector (double *a)
- double getValue (int index)
- double * getVector ()
- double sum ()
- Vector3d operator+ (const Vector3d &) const
- void operator+= (const Vector3d &)
- Vector3d operator- (const Vector3d &) const
- void operator-= (const Vector3d &)
- Vector3d operator* (const double &) const
- void operator*= (const double &)
- double operator* (const Vector3d &) const
- Vector3d operator^ (const Vector3d &) const
- void operator^= (const Vector3d &)

Protected Attributes

- double x [3]

4.4.1 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 18 of file vector3d.h.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 Vector3d::Vector3d ( )

Default constructor.

Definition at line 15 of file vector3d.cpp.

```
{
  this->x[0] = 0.0;
  this->x[1] = 0.0;
  this->x[2] = 0.0;
}
```

4.4.2.2 Vector3d::Vector3d (double * *a*)

Constructor with values provided in an array.

Parameters

a	Pointer to the array containing the elements of the vector

Definition at line 25 of file vector3d.cpp.

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

4.4.2.3 Vector3d::Vector3d (double *a1,* double *a2,* double *a3*)

Constructor with values provided explicitly.

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 38 of file vector3d.cpp.

```
{
  this->x[0] = a1;
  this->x[1] = a2;
  this->x[2] = a3;
}
```

4.4.3 Member Function Documentation

4.4.3.1 double Vector3d::getValue (int *index*)

Function to get the value of an element of the vector.

Parameters

index	Index of the element whose value is to be got.

Definition at line 75 of file vector3d.cpp.

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

4.4.3.2 double * Vector3d::getVector ( )

Function to get the values of the elements of the vector in an array.

Definition at line 90 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);
}
```

4.4.3.3 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.

Definition at line 187 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);
}
```

4.4.3.4 double Vector3d::operator* (const Vector3d & *p*) const

Operator for the scalar product of two vectors.

Definition at line 217 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);
}
```

4.4.3.5 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 204 of file vector3d.cpp.

```
{
  int i;

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

4.4.3.6 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.

Definition at line 123 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);
}
```

4.4.3.7 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 140 of file vector3d.cpp.

```
{
  int i;

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

4.4.3.8 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.

Definition at line 155 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);
}
```

4.4.3.9 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 172 of file vector3d.cpp.

```
{
  int i;

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

4.4.3.10 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.

Definition at line 234 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);
}
```

4.4.3.11 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 249 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;
}
```

4.4.3.12 void Vector3d::setValue (int *index*, double *value*)

Function to set the value of an element of the vector.

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 51 of file vector3d.cpp.

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

4.4.3.13 void Vector3d::setVector (double * *a*)

Function to set the value of the entire vector using an array.

Parameters

a	Pointer ot the array containing the values of the elements of the vector.

Definition at line 63 of file vector3d.cpp.

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

4.4.3.14 double Vector3d::sum ( )

Computes the sum of the elements of the vector.

Definition at line 104 of file vector3d.cpp.

```
{
  double s = 0.0;
  int i;

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

  return (s);
}
```

4.4.4 Field Documentation

4.4.4.1 double Vector3d::x[3] [protected]

The elements if the vector.

Definition at line 24 of file vector3d.h.

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

- vector3d.h
- vector3d.cpp

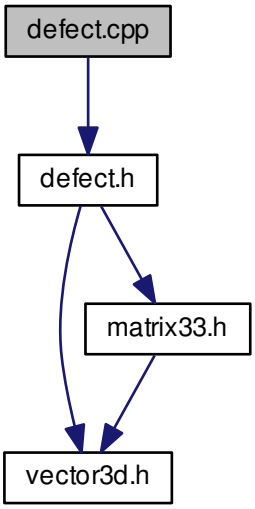
Chapter 5

File Documentation

5.1 defect.cpp File Reference

```
#include "defect.h"
```

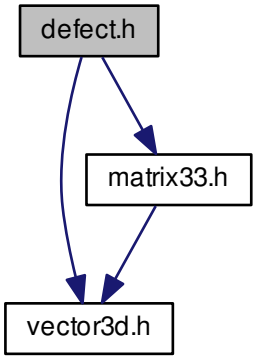
Include dependency graph for defect.cpp:



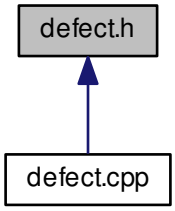
5.2 defect.h File Reference

```
#include "vector3d.h"
#include "matrix33.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.

5.2.1 Detailed Description

Author

Adhish Majumdar

Version

0.0

Date

16/04/2013

5.2.2 BRIEF

Definition of the Defect class.

5.2.3 DETAILS

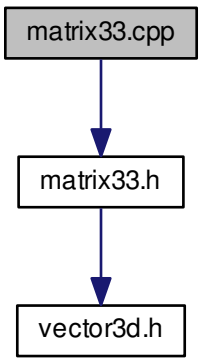
This file defines the Defect class representing an defect in the simulation. This is simply a generic description class with virtual functions. Later classes like dislocations, precipitates, boundaries etc will inherit from this class.

Definition in file defect.h.

5.3 matrix33.cpp File Reference

```
#include "matrix33.h"
```

Include dependency graph for matrix33.cpp:

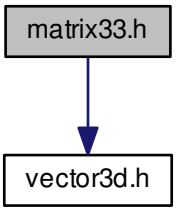


5.4 matrix33.h File Reference

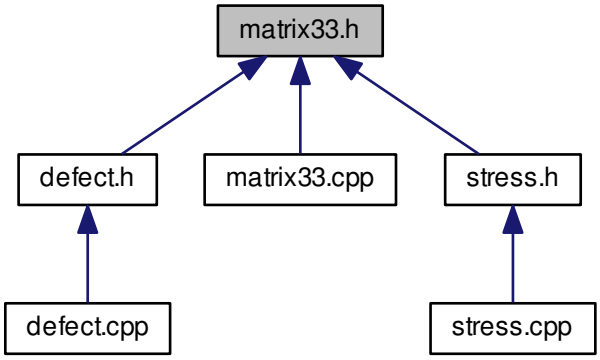
Definition of the Matrix33 class.

```
#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.

5.4.1 Detailed Description

Definition of the Matrix33 class.

Author

Adhish Majumdar

Version

0.0

Date

15/04/2013

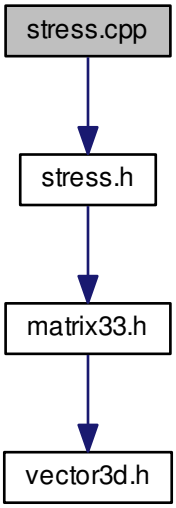
This file defines the Matrix33 class representing a 3x3 matrix in the simulation.

Definition in file matrix33.h.

5.5 stress.cpp File Reference

```
#include "stress.h"
```

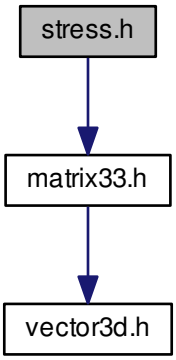
Include dependency graph for stress.cpp:



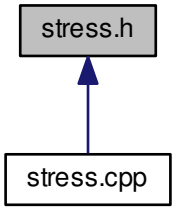
5.6 stress.h File Reference

Definition of the Stress class.

```
#include "matrix33.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.

5.6.1 Detailed Description

Definition of the Stress class.

Author

Adhish Majumdar

Version

0.0

Date

19/04/2013

This file defines the Stress class for the stress tensor.

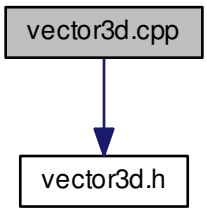
Definition in file stress.h.

5.7 vector3d.cpp File Reference

Definition of the Vector3d class and its functions.

```
#include "vector3d.h"
```

Include dependency graph for vector3d.cpp:



5.7.1 Detailed Description

Definition of the Vector3d class and its functions.

Author

Adhish Majumdar

Version

0.0

Date

15/04/2013

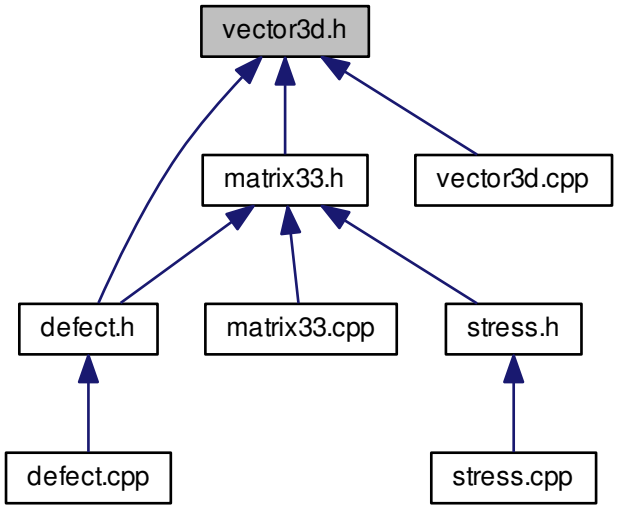
This file defines the Vector3d class representing a single 3-dimensional vector in the simulation and its member functions and operators.

Definition in file vector3d.cpp.

5.8 vector3d.h File Reference

Definition of the Vector3d class.

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.

5.8.1 Detailed Description

Definition of the Vector3d class.

Author

Adhish Majumdar

Version

0.0

Date

15/04/2013

This file defines the Vector3d class representing a single 3-dimensional vector in the simulation.

Definition in file vector3d.h.