Computed Tomography

Generated by Doxygen 1.8.8

Fri Jan 16 2015 03:34:55

Contents

Chapter 1

Developing a Platform for Computed Tomography

##Final Project for APC524 Fall 2014, Princeton University

Suerfu, Qi Li, Yao Zhou, Xiang Gao

X-ray computed tomography, or simply CT, is a technique for non-invasive imaging through objects. With appropriate reconstruction algorithm, two/three dimensional cross-sectional images can be obtained from multiple projections along different directions. This software is capable of simulating such process: generating test objects, forming projections, reconstruction and visualization.

Description of the Software

- **TEST OBJECT** Test objects are used to study and check the performance of reconstruction algorithms. The overall base class is called *Image*, from which derives *Curve*(1D), *Surface*(2D), and *Volume*(3D). Each of the derived class must implement *operator(double,double,double,intpl)*, which is used to access values at given points. Currently test objects are implemented either analytically or numerically. Analytical objects are created by specifying a function rule that determines field values. Numerical objects are implemented by storing coordinates and corresponding values, and uses Interpolation method to access field values. If HDF5 is enabled, numerical images can also be read from .h5 files that has the correct format. If one wants to interface his own test objects, at least *operator()* must be implemented such that line integration is possible.
- **PROJECTION** Projection is the key process in reconstruction. In CT projection refers to line integrals (of X-ray attenuation coefficient) along a set of parallel lines. Therefore the result of a projection is another function with lower dimension. *Surface* object has a method *GetProjection(intg,double,...)* that returns a *NumCurve* object as a result of projection along specified direction. The curve object is characterized by the parallel distance to the center. In calling projection method, an *Integrator* must be specified. This integrator is an abstract class that performs line integrals. Currently implemented integration schemes are: *Trapezoid*, *Parabola*, *Romberg* and *Monte-Carlo*. Similarly *Volume* has a *GetProjection* method that returns a *NumCurve* at the specified angle and height. Result of projections are stored in an object called *ImageArray*.
- **RECONSTRUCTION** Reconstruction refers to combining information from multiple projections to reproduce the scalar field.
  - *ImageArray* object, which stores result of projections has the following features:
  - Store and access *Image* and the angle/height at which they were taken,
  - Call convolution on all the stored *Image* objects with a specified kernal. This kernal is by default Hamming function. The core of reconstruction is *FilteredBackProjection* class, which takes in an *ImageArray* object, performs convolution and back-project images. The results are superposed and returns a *Num↩Surface* object. For more information about reconstruction algorithm, please refer to *Principles of Computerized Tomographic Imaging* by Kak and Slaney. If one wants to interface with our software by implementing different reconstruction algorithm, it is better to work with *ImageArray* objects since they contain essentially all information that is needed for reconstruction. About usage of classes, please refer to user manual.

- **VISUALIZATION** All the derived classes of *Image* (*Curve* & *NumCurve*, *Surface* & *NumSurface*, *Volume* & *NumVolume*) are equipped with a method named ExportHDF. When the method is called, the data in the class is saved into a designated HDF5 file in directory 'output'. The file includes 1D arrays (/x, /y, & /z, depending on dimension) storing the coordinates of the rectilinear mesh, and array /data storing the value at each node. To enable this feature, one has to install HDF5, specify HDF5 header and lib locations, and run 'make USE_HDF=1'. A few sample python scripts can also be found in the output directory to visualize the output data (plotCurve.py, plotSurface.py, & movieVolume.py). plotCurve.py is used to take in an HDF5 file and produce a 1D function. plotSurface.py is used to produce png images from a 2D surface. movieVolume.py will generate a movie that shows cross-section in order from top to bottom. Note that movieVolume.py requires codec (such as FFMpeg) to save the movie. Example usage: "python plotCurve.py Curve.h5". Also note that all classes deriving from *Image* have a *Print()* method which prints data points to stdout. User can redirect stdout to a file and use tools of their choice to visualize. When exporting HDF5 files from 2D and 3D Images, a XDMF file is also generated in the output directory to enable reading and visualizing with VisIt (visit.llnl.gov). In VisIt, simply open the .xmf file with format Xdmf and draw contour, pseudocolor, etc. Besides, all the derived classes of Image also have a constructor from reading in data from a HDF5 file that has the same "flavor" as the output file.

System Requirement

- The software is written in C++. Since lambda function deature is used in performing line integral, to compile C++ compilor that supports C++11 is required.
- Currently this software has been tested on Linux(Ubuntu 12.4) and Mac OS with g++.
- For visualization, HDF5 is required.
- The result can be viewed with either `VisIt` or python package `Mayavi` which uses VTK.

Install and File Description

- To compile: make
- To enable HDF5: make USE_HDF=1
- By default:
  - ./include/ contains all header files.
  - ./src/ contains all source files.
  - ./test/ contains functions used to test during development.
  - ./demo/ contains codes for demonstrating usage of this software.
  - ./output/ default location for exporting HDF5 files.
  - ./bin/ binary executable files produced by make.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 movieVolume Namespace Reference

Functions

- def animate

Variables

- list fname = sys.argv[1]
- tuple infile = h5py.File(fname, 'r')
- tuple x = np.array((infile["x"]))
- tuple y = np.array((infile["y"]))
- tuple z = np.array((infile["z"]))
- tuple data = np.array((infile["data"]))
- tuple dmin = data.min()
- tuple dmax = data.max()
- tuple fig = plt.figure()
- tuple ani = manimation.FuncAnimation(fig, animate, frames = z.shape[0], interval = 50)

6.1.1 Function Documentation

6.1.1.1 def movieVolume.animate (*i*)

Definition at line 21 of file movieVolume.py.

```
21
22 def animate(i):
23   im = plt.pcolormesh(X, Y, data[i, :, :], vmin = dmin, vmax = dmax)
24   plt.title('z = %.2f' % z[i])
25   return im
```

6.1.2 Variable Documentation

6.1.2.1 tuple movieVolume.ani = manimation.FuncAnimation(fig, animate, frames = z.shape[0], interval = 50)

Definition at line 26 of file movieVolume.py.

6.1.2.2 tuple movieVolume.data = np.array((infile["data"]))

Definition at line 14 of file movieVolume.py.

6.1.2.3 tuple movieVolume.dmax = data.max()

Definition at line 17 of file movieVolume.py.

6.1.2.4 tuple movieVolume.dmin = data.min()

Definition at line 16 of file movieVolume.py.

6.1.2.5 tuple movieVolume.fig = plt.figure()

Definition at line 19 of file movieVolume.py.

6.1.2.6 list movieVolume.fname = sys.argv[1]

Definition at line 9 of file movieVolume.py.

6.1.2.7 tuple movieVolume.infile = h5py.File(fname, 'r')

Definition at line 10 of file movieVolume.py.

6.1.2.8 tuple movieVolume.x = np.array((infile["x"]))

Definition at line 11 of file movieVolume.py.

6.1.2.9 tuple movieVolume.y = np.array((infile["y"]))

Definition at line 12 of file movieVolume.py.

6.1.2.10 tuple movieVolume.z = np.array((infile["z"]))

Definition at line 13 of file movieVolume.py.

6.2 plotCurve Namespace Reference

Variables

- list fname = sys.argv[1]
- tuple infile = h5py.File(fname + ".h5", 'r')
- list x = infile["x"]
- list data = infile["data"]

6.2.1 Variable Documentation

6.2.1.1 list plotCurve.data = infile["data"]

Definition at line 12 of file plotCurve.py.

6.2.1.2 list plotCurve.fname = sys.argv[1]

Definition at line 8 of file plotCurve.py.

6.2.1.3 tuple plotCurve.infile = h5py.File(fname + ".h5", 'r')

Definition at line 10 of file plotCurve.py.

6.2.1.4 list plotCurve.x = infile["x"]

Definition at line 11 of file plotCurve.py.

6.3 plotSurface Namespace Reference

Variables

- list fname = sys.argv[1]
- tuple infile = h5py.File(fname, 'r')
- tuple x = np.array((infile["x"]))
- tuple y = np.array((infile["y"]))
- tuple data = np.array((infile["data"]))

6.3.1 Variable Documentation

6.3.1.1 tuple plotSurface.data = np.array((infile["data"]))

Definition at line 14 of file plotSurface.py.

6.3.1.2 list plotSurface.fname = sys.argv[1]

Definition at line 9 of file plotSurface.py.

6.3.1.3 tuple plotSurface.infile = h5py.File(fname, 'r')

Definition at line 11 of file plotSurface.py.

6.3.1.4 tuple plotSurface.x = np.array((infile["x"]))

Definition at line 12 of file plotSurface.py.

6.3.1.5 tuple plotSurface.y = np.array((infile["y"]))

Definition at line 13 of file plotSurface.py.

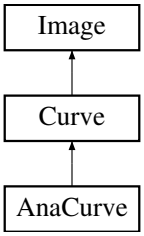
Chapter 7

Class Documentation

7.1 AnaCurve Class Reference

`#include <AnaImage.h>`

Inheritance diagram for AnaCurve:



Public Member Functions

- AnaCurve (f1D, double range)

  Constructs with R->R. X is the range of this function.
- ~AnaCurve ()

  Destructor, does nothing.
- double operator() (double, Interpolator *) const

  Evaluate function value and returns by reference.

Private Attributes

- f1D _f1d

  1D function

Additional Inherited Members

7.1.1 Detailed Description

Concrete 1D image class with analytical expressions. It is defined on a domain of radius given as the second argument of constructor.

Definition at line 14 of file AnaImage.h.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 AnaCurve::AnaCurve (f1D *f*, double *range*)

Constructs with R->R. X is the range of this function.

Definition at line 6 of file AnaImage.cpp.

```
6                                       : Curve(x)
7 { _f1d = f; }
```

7.1.2.2 AnaCurve::~AnaCurve ( )

Destructor, does nothing.

Definition at line 9 of file AnaImage.cpp.

```
10 {}
```

7.1.3 Member Function Documentation

7.1.3.1 double AnaCurve::operator() (double *x*, Interpolator * *intp*) const [virtual]

Evaluate function value and returns by reference.

Implements Curve.

Definition at line 24 of file AnaImage.cpp.

```
25 { return _f1d(x); }
```

7.1.4 Member Data Documentation

7.1.4.1 f1D AnaCurve::_f1d [private]

1D function

Definition at line 22 of file AnaImage.h.

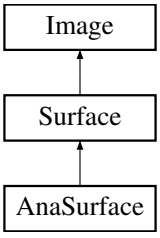
The documentation for this class was generated from the following files:

- include/AnaImage.h
- src/AnaImage.cpp

7.2 AnaSurface Class Reference

```
#include <AnaImage.h>
```

Inheritance diagram for AnaSurface:



Public Member Functions

- AnaSurface (f2D, double, double)
- ∼AnaSurface ()

  Constructs with R2->R.
- double operator() (double, double, Interpolator ∗) const

  Destructor, does nothing.

Private Attributes

- f2D _f2d

  Evaluate function value.

Additional Inherited Members

7.2.1 Detailed Description

Concrete 2D image class with analytical expressions.

Definition at line 28 of file AnaImage.h.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 AnaSurface::AnaSurface (f2D *f*, double *x*, double *y*)

Definition at line 12 of file AnaImage.cpp.

```
12                                                  :Surface(x,y)
13 { _f2d = f; }
```

7.2.2.2 AnaSurface::∼AnaSurface ( )

Constructs with R2->R.

Definition at line 15 of file AnaImage.cpp.

```
16 { }
```

7.2.3 Member Function Documentation

7.2.3.1 double AnaSurface::operator() (double *x*, double *y*, Interpolator ∗ *intp*) const [virtual]

Destructor, does nothing.

Implements Surface.

Definition at line 27 of file AnaImage.cpp.

```
28 { return _f2d(x,y); }
```

7.2.4 Member Data Documentation

7.2.4.1 f2D AnaSurface::_f2d [private]

Evaluate function value.

Definition at line 35 of file AnaImage.h.

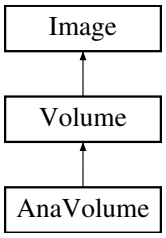
The documentation for this class was generated from the following files:

- include/AnaImage.h
- src/AnaImage.cpp

7.3 AnaVolume Class Reference

`#include <AnaImage.h>`

Inheritance diagram for AnaVolume:



Public Member Functions

- AnaVolume (f3D, double, double, double)
- ~AnaVolume ()

  constructs with R3->R
- double operator() (double, double, double, Interpolator *) const

  destructor, does nothing.

Private Attributes

- f3D _f3d

  evaluate function value.

Additional Inherited Members

7.3.1 Detailed Description

Concrete 3D image class with analytical expressions.

Definition at line 41 of file AnaImage.h.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 AnaVolume::AnaVolume (f3D *f*, double *x*, double *y*, double *z*)

Definition at line 18 of file AnaImage.cpp.

```
18                                                          :Volume(x,y,z)
19 { _f3d = f; }
```

7.3.2.2 AnaVolume::~AnaVolume ( )

constructs with R3->R

Definition at line 21 of file AnaImage.cpp.

```
22 { }
```

7.3.3 Member Function Documentation

7.3.3.1 double AnaVolume::operator() (double *x*, double *y*, double *z*, Interpolator * *intp*) const [virtual]

destructor, does nothing.

Implements Volume.

Definition at line 30 of file AnaImage.cpp.

```
31 { return _f3d(x,y,z); }
```

7.3.4 Member Data Documentation

7.3.4.1 f3D AnaVolume::_f3d [private]

evaluate function value.

Definition at line 48 of file AnaImage.h.

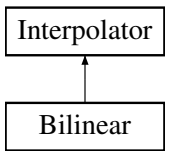
The documentation for this class was generated from the following files:

- include/AnaImage.h
- src/AnaImage.cpp

7.4 Bilinear Class Reference

`#include <Bilinear.h>`

Inheritance diagram for Bilinear:



Public Member Functions

- Bilinear ()

  Constructor for bilinear Interpolator.
- ~Bilinear ()

  Destructor for bilinear Interpolator. Free memory.

- double Interpolate (double x)

  Interpolate the one-dimensional data linearly at a given x.
- double Interpolate (double x, double y)

  Interpolate the two-dimensional data bilinearly at given (x,y).
- double Interpolate (double x, double y, double z)

  Implements the virtual method of class Interpolator.

Additional Inherited Members

7.4.1 Detailed Description

Interpolate a function using nearest neighbour method.

Definition at line 11 of file Bilinear.h.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 Bilinear::Bilinear ( )

Constructor for bilinear Interpolator.

Definition at line 4 of file Bilinear.cpp.

7.4.2.2 Bilinear::~Bilinear ( )

Destructor for bilinear Interpolator. Free memory.

Definition at line 5 of file Bilinear.cpp.

7.4.3 Member Function Documentation

7.4.3.1 double Bilinear::Interpolate (double *x*) [virtual]

Interpolate the one-dimensional data linearly at a given x.

Implements Interpolator.

Definition at line 8 of file Bilinear.cpp.

```
9 {
10     int i0 = ArryIndexFloor(x,_xptr,_sizex);
11     if(i0<0) return 0;
12     int i1 = i0 +1;
13     return _yptr[i0]+(_yptr[i1]-_yptr[i0])*(x-_xptr[i0])/(
       _xptr[_sizex-1]-_xptr[0]);
14 }
```

7.4.3.2 double Bilinear::Interpolate (double *x*, double *y*) [virtual]

Interpolate the two-dimensional data bilinearly at given (x,y).

Implements Interpolator.

Definition at line 16 of file Bilinear.cpp.

```
17 {
18     int i0x = ArryIndexFloor(x,_xptr,_sizex);
19     int i1x = i0x+1;
20     int i0y = ArryIndexFloor(y,_yptr,_sizey);
```

```
21     int i1y = i0y+1;
22
23     if ( i0x<0 || i0y < 0){return 0;}
24
25     double Q11 = _zzptr[i0x][i0y];
26     double Q21 = _zzptr[i1x][i0y];
27     double Q12 = _zzptr[i0x][i1y];
28     double Q22 = _zzptr[i1x][i1y];
29     double s1 = Q11*(_xptr[i1x]-x)*(_yptr[i1y]-y);
30     double s2 = Q21*(x-_xptr[i0x])*(_yptr[i1y]-y);
31     double s3 = Q12*(_xptr[i1x]-x)*(y-_yptr[i0y]);
32     double s4 = Q22*(x-_xptr[i0x])*(y-_yptr[i0y]);
33     return 1./((_xptr[i1x]-_xptr[i0x])*(_yptr[i1y]-_yptr[i0y]))*(s1+s2+s3+s4);
34 }
```

7.4.3.3 double Bilinear::Interpolate (double *x*, double *y*, double *z*) [virtual]

Implements the virtual method of class Interpolator.

Interpolate the three-dimensional data trilinearly at given (x,y,z).

Implements Interpolator.

Definition at line 36 of file Bilinear.cpp.

```
37 {
38     int i0x = ArryIndexFloor(x,_xptr,_sizex);
39     int i1x = i0x+1;
40     int i0y = ArryIndexFloor(y,_yptr,_sizey);
41     int i1y = i0y+1;
42     int i0z = ArryIndexFloor(z,_zptr,_sizez);
43     int i1z = i0z+1;
44     if(i0x<0 || i0y<0 || i0z<0) return 0;
45
46     double dx = (_xptr[_sizex-1]-_xptr[0])/(_sizex-1);
47     double dy = (_yptr[_sizey-1]-_yptr[0])/(_sizey-1);
48     double dz = (_zptr[_sizez-1]-_zptr[0])/(_sizez-1);
49
50     double xd = (x-_xptr[i0x])/dx;
51     double yd = (y-_yptr[i0y])/dy;
52     double zd = (z-_zptr[i0z])/dz;
53     double w1 = _wptr[i0x][i0y][i0z];
54     double w2 = _wptr[i1x][i0y][i0z];
55     double w3 = _wptr[i0x][i1y][i0z];
56     double w4 = _wptr[i1x][i1y][i0z];
57     double w5 = _wptr[i0x][i0y][i1z];
58     double w6 = _wptr[i1x][i0y][i1z];
59     double w7 = _wptr[i0x][i1y][i1z];
60     double w8 = _wptr[i1x][i1y][i1z];
61     double c00 = w1*(1-xd)+w2*xd;
62     double c10 = w3*(1-xd)+w4*xd;
63     double c01 = w5*(1-xd)+w6*xd;
64     double c11 = w7*(1-xd)+w8*xd;
65     double c0 = c00*(1-yd)+c10*yd;
66     double c1 = c01*(1-yd)+c11*yd;
67     return c0*(1-zd)+c1*zd;
68 }
```

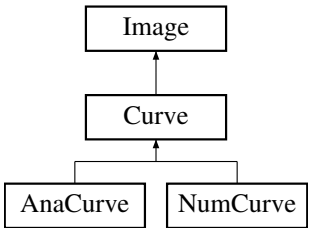
The documentation for this class was generated from the following files:

- include/Bilinear.h
- src/Bilinear.cpp

7.5 Curve Class Reference

`#include <Curve.h>`

Inheritance diagram for Curve:



Public Member Functions

- Curve (double rx)

  Constructor. Argument is the radial size of the function.
- virtual ∼Curve ()

  Virtual destructor, in case someone calls delete derived.
- virtual double operator() (double x, Interpolator ∗intpl=0) const =0

  Returns image value at the place speficied by the argument.
- virtual void Print ()

  Implements Image::Print. It should print the function as two columns.
- virtual void Print (double xmin, double xmax, int N=100, Interpolator ∗intpl=0)

  Output the function values in the range specified.
- void SetRange (double rx)

  Set symmetrized range of independent variable. S.
- double GetRange () const

  Returns the symmetrized range. S.

Protected Attributes

- double _r

  Range of the function, same as the radius. S.

7.5.1 Detailed Description

Definition at line 17 of file Curve.h.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 Curve::Curve (double *rx*)

Constructor. Argument is the radial size of the function.

< Curve must be initialized with the range over which it is defined.

Definition at line 8 of file Curve.cpp.

```
8                       :Image(Dim1)
9 {
10      _r = rx;
12 }
```

7.5.2.2 Curve::~Curve ( ) [virtual]

Virtual destructor, in case someone calls delete derived.

Definition at line 14 of file Curve.cpp.

```
14 {}
```

7.5.3 Member Function Documentation

7.5.3.1 double Curve::GetRange ( ) const

Returns the symmetrized range. S.

Definition at line 21 of file Curve.cpp.

```
22 {
23     return _r;
24 }
```

7.5.3.2 virtual double Curve::operator() (double *x*, Interpolator * *intpl* = 0) const [pure virtual]

Returns image value at the place speficied by the argument.

Implemented in NumCurve, and AnaCurve.

7.5.3.3 void Curve::Print ( ) [virtual]

Implements Image::Print. It should print the function as two columns.

Implements Image.

Reimplemented in NumCurve.

Definition at line 76 of file Curve.cpp.

```
77 {
78   this->Print(-_r,_r,100);
79 }
```

7.5.3.4 void Curve::Print (double *xmin*, double *xmax*, int *N* = 100, Interpolator * *intpl* = 0) [virtual]

Output the function values in the range specified.

Definition at line 67 of file Curve.cpp.

```
68 {
69   double step = (xmax-xmin)/N;
70   for( int i = 0; i < N; i++) {
71     double x =  xmin + step * i;
72     printf("%.8f %.8f\n",x,(*this)(x,intpl));
73   }
74 }
```

7.5.3.5 void Curve::SetRange (double *rx*)

Set symmetrized range of independent variable. S.

Definition at line 16 of file Curve.cpp.

```
17 {
18     _r = rx;
19 }
```

7.5.4 Member Data Documentation

7.5.4.1 double Curve::_r [protected]

Range of the function, same as the radius. S.

Definition at line 41 of file Curve.h.

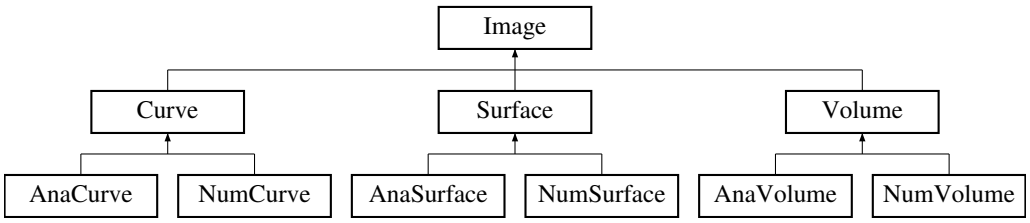
The documentation for this class was generated from the following files:

- include/Curve.h
- src/Curve.cpp

7.6 Image Class Reference

`#include <Image.h>`

Inheritance diagram for Image:



Public Member Functions

- Image (Dimension dim=Dim0)

  Constructor. Dimension by default is 0.
- virtual ~Image ()

  Virtual destructor, in case someone calls delete derived.
- virtual void Print ()=0

  Default method for printing.
- Dimension GetDimension ()

  Returns the dimension of the image.

Protected Attributes

- Dimension _dim

  Dimension of the problem, will be 1D, 2D, or 3D.

7.6.1 Detailed Description

An abstract Image class should contain the following abstract virtual methods: (1) A function to return dimensionality. (2) A Print method for default output. (3) An Export method to create HDF5 file. An Image is further classified into 1D (Curve), 2D(Surface) and 3D(Volume). Object in each dimension will have to implement operator(), which returns the image value at the argument point. S

Definition at line 25 of file Image.h.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 Image::Image (Dimension *dim* = Dim0)

Constructor. Dimension by default is 0.

Definition at line 3 of file Image.cpp.

```
3 { _dim = dim; }
```

7.6.2.2 Image::~Image ( ) [virtual]

Virtual destructor, in case someone calls delete derived.

Definition at line 5 of file Image.cpp.

```
5 {}
```

7.6.3 Member Function Documentation

7.6.3.1 Dimension Image::GetDimension ( )

Returns the dimension of the image.

Definition at line 7 of file Image.cpp.

```
7 { return _dim;}
```

7.6.3.2 virtual void Image::Print ( ) [pure virtual]

Default method for printing.

Implemented in NumSurface, NumVolume, NumCurve, Surface, Curve, and Volume.

7.6.4 Member Data Documentation

7.6.4.1 Dimension Image::_dim [protected]

Dimension of the problem, will be 1D, 2D, or 3D.

Definition at line 39 of file Image.h.

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

- include/Image.h
- src/Image.cpp

7.7 ImageArray Class Reference

```
#include <ImageArray.h>
```

Public Member Functions

- ImageArray ()
- ~ImageArray ()
- int GetSize ()

  Return the total number of elem in the vector. Number of view.
- void SetSlice (int)

  Set the total number of horizontal slice.
- int GetSlice ()

  Get the total number of horizontal slice.
- double GetAngle (int)

  Get the angle for ith projection.
- double GetHeight (int)

  Return the height for ith projection.
- double GetRange ()

  Get the maximum range of the curve.
- double GetRangeZ ()

  Get the maximum range in z direction.
- NumCurve & GetCurve (int)

  Return ith curve.
- NumCurve & GetFilteredCurve (int)

  Return filtered curve.
- void PushBack (double, const NumCurve &)

  PushBack for 2D reconstruction.
- void PushBack (double, double, const NumCurve &)

  PushBack for 3D reconstruction.
- void ConvolveWithKernal (double(*kernal)(int, double)=Hamming)

  Convole all elements with kernal.
- void Print ()

  Default print method.
- void PrintFiltered ()

  Print all filtered curve.
- void PrintSinogram (double spacing=0.01)

  Print out the sinogram.

Private Attributes

- std::vector< NumCurve > _curve

  Stores the projection.
- std::vector< NumCurve > _filtered

  Stores the projection after convolution.
- std::vector< double > _angle

  Each angle at which projection is taken.
- std::vector< double > _height

  Each height at which projection is taken.
- int _size

  Total number of view.
- int _slice

  Total number of horizontal slice.

7.7.1 Detailed Description

This is a container class that contains objects of type NumCurve. The curves can be projections taken at various angles and heights, stored as vectors in the class. This object will be passed on to reconstructor such as Filtered↩ BackProjection.

Definition at line 14 of file ImageArray.h.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 ImageArray::ImageArray ( )

Definition at line 4 of file ImageArray.cpp.

```
4 :_size(0){}
```

7.7.2.2 ImageArray::∼ImageArray ( )

Definition at line 5 of file ImageArray.cpp.

```
6 {
7     _angle.clear();
8     _filtered.clear();
9     _curve.clear();
10     _height.clear();
11 }
```

7.7.3 Member Function Documentation

7.7.3.1 void ImageArray::ConvolveWithKernal (double(∗)(int, double) *kernal* = Hamming)

Convole all elements with kernal.

< Nyquist frequency

< beginning convolution.

Definition at line 109 of file ImageArray.cpp.

```
110 {
111     double _ran = this->GetRange();
112     for(int i=0;i<_size;i++){
113         int Npt = _curve[i].GetSize();
114         double tau = 2*_ran/(Npt-1);
115         for(int j=0;j<Npt;j++){
116             _filtered[i][j] = 0;
117             for(int k=j-Npt+1;k<j+1;k++)
118                 _filtered[i][j] += tau * kernal(k,tau)*(_curve[i])[j-k];
119         }
120     }
121 }
```

7.7.3.2 double ImageArray::GetAngle (int *i*)

Get the angle for ith projection.

Definition at line 28 of file ImageArray.cpp.

```
29 {
30     return _angle[i];
31 }
```

7.7.3.3 NumCurve & ImageArray::GetCurve (int *i*)

Return ith curve.

Definition at line 38 of file ImageArray.cpp.

```
39 {
40     return _curve[i];
41 }
```

7.7.3.4 NumCurve & ImageArray::GetFilteredCurve (int *i*)

Return filtered curve.

Definition at line 123 of file ImageArray.cpp.

```
124 {
125     return _filtered[i];
126 }
```

7.7.3.5 double ImageArray::GetHeight (int *i*)

Return the height for ith projection.

Definition at line 33 of file ImageArray.cpp.

```
34 {
35     return _height[i];
36 }
```

7.7.3.6 double ImageArray::GetRange ( )

Get the maximum range of the curve.

Definition at line 95 of file ImageArray.cpp.

```
96 {
97     double r = 0;
98     for(int i=0;i<_size;i++) r = max<double>(r,_curve[i].GetRange());
99     return r;
100 }
```

7.7.3.7 double ImageArray::GetRangeZ ( )

Get the maximum range in z direction.

Definition at line 102 of file ImageArray.cpp.

```
103 {
104     double r = 0;
105     for(int i=0;i<_size;i++) r = max<double>(r,_height[i]);
106     return r;
107 }
```

7.7.3.8 int ImageArray::GetSize ( )

Return the total number of elem in the vector. Number of view.

Definition at line 13 of file ImageArray.cpp.

```
14 {
15     return _size;
16 }
```

7.7.3.9 int ImageArray::GetSlice ( )

Get the total number of horizontal slice.

Definition at line 23 of file ImageArray.cpp.

```
24 {
25     return _slice;
26 }
```

7.7.3.10 void ImageArray::Print ( )

Default print method.

Definition at line 60 of file ImageArray.cpp.

```
61 {
62     for(int i=0;i<_size;i++){
63         _curve[i].Print();
64 
65     }
66 }
```

7.7.3.11 void ImageArray::PrintFiltered ( )

Print all filtered curve.

Definition at line 68 of file ImageArray.cpp.

```
69 {
70     for(int i=0;i<_size;i++){
71         _filtered[i].Print();
72         printf("\n");
73     }
74 }
```

7.7.3.12 void ImageArray::PrintSinogram (double *spacing* = 0.01)

Print out the sinogram.

Definition at line 76 of file ImageArray.cpp.

```
77 {
78     bool same_size = true;
79     for(int i=0;i<_size-1;i++)
80         if(_curve[i].GetSize()!=_curve[i+1].GetSize()) {same_size = false; break;
      }
81     if(same_size)
82     for(int i=_size-1;i>=0;i--){
83          for(int j=0;j<_curve[i].GetSize();j++)
84              printf("%.8f ",(_curve[i])[j]);
85          printf("\n");
86     }
87     else
88     for(int i=_size-1;i>=0;i++){
89          for(double j=-_curve[i].GetRange();j<_curve[i].GetRange();j+=spacing)
90              printf("%.8f ",(_curve[i])(j,0));
91          printf("\n");
92     }
93 }
```

7.7.3.13 void ImageArray::PushBack (double *a*, const NumCurve & *c*)

PushBack for 2D reconstruction.

Definition at line 43 of file ImageArray.cpp.

```
44 {
45     _curve.push_back(c);
46     _filtered.push_back(c);
47     _angle.push_back(a);
48     _size++;
49 }
```

7.7.3.14 void ImageArray::PushBack (double *a*, double *h*, const NumCurve & *c*)

PushBack for 3D reconstruction.

Parameters

c	PushBack method for 3D objects.Note that _size will be the total number of NumCurves in the whole 3D domain.

Definition at line 51 of file ImageArray.cpp.

```
52 {
53     _curve.push_back(c);
54     _filtered.push_back(c);
55     _angle.push_back(a);
56     _height.push_back(h);
57     _size++;
58 }
```

7.7.3.15 void ImageArray::SetSlice (int *slice*)

Set the total number of horizontal slice.

Definition at line 18 of file ImageArray.cpp.

```
19 {
20     _slice = slice;
21 }
```

7.7.4 Member Data Documentation

7.7.4.1 std::vector<double> ImageArray::_angle [private]

Each angle at which projection is taken.

Definition at line 39 of file ImageArray.h.

7.7.4.2 std::vector<NumCurve> ImageArray::_curve [private]

Stores the projection.

Definition at line 37 of file ImageArray.h.

7.7.4.3 std::vector<NumCurve> ImageArray::_filtered [private]

Stores the projection after convolution.

Definition at line 38 of file ImageArray.h.

7.7.4.4 std::vector<double> ImageArray::_height [private]

Each height at which projection is taken.

Definition at line 40 of file ImageArray.h.

7.7.4.5 int ImageArray::_size [private]

Total number of view.

Definition at line 41 of file ImageArray.h.

7.7.4.6 int ImageArray::_slice [private]

Total number of horizontal slice.

Definition at line 42 of file ImageArray.h.

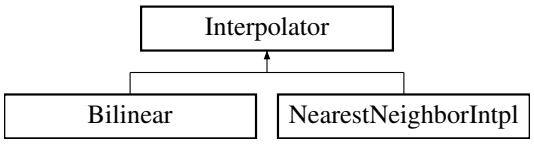
The documentation for this class was generated from the following files:

- include/ImageArray.h
- src/ImageArray.cpp

7.8 Interpolator Class Reference

`#include <Interpolator.h>`

Inheritance diagram for Interpolator:



Public Member Functions

- Interpolator ()

  constructor
- ~Interpolator ()

  destructor
- void set_values (int, double *, double *)
- void set_values (int, int, double *, double *, double **)
- void set_values (int, int, int, double *, double *, double *, double ***)

  3D interpolator set value method. This function must be called before interpolation.
- virtual double Interpolate (double)=0

  Interpolate method for NumCurve.
- virtual double Interpolate (double, double)=0

  Interpolate method for NumSurface.
- virtual double Interpolate (double, double, double)=0

  Interpolate method for NumVolume.

Protected Attributes

- int _sizex
- int _sizey
- int _sizez
- double * _xptr
- double * _yptr
- double * _zptr
- double ** _zzptr
- double *** _wptr

7.8.1 Detailed Description

Interpolator class.

Definition at line 13 of file Interpolator.h.

7.8.2 Constructor & Destructor Documentation

7.8.2.1 Interpolator::Interpolator ( )

constructor

Definition at line 10 of file Interpolator.cpp.

```
10 {}
```

7.8.2.2 Interpolator::~Interpolator ( )

destructor

Definition at line 12 of file Interpolator.cpp.

```
12 {}
```

7.8.3 Member Function Documentation

7.8.3.1 virtual double Interpolator::Interpolate (double) [pure virtual]

Interpolate method for NumCurve.

Implemented in Bilinear.

7.8.3.2 virtual double Interpolator::Interpolate (double , double) [pure virtual]

Interpolate method for NumSurface.

Implemented in Bilinear, and NearestNeighborIntpl.

7.8.3.3 virtual double Interpolator::Interpolate (double , double , double) [pure virtual]

Interpolate method for NumVolume.

Implemented in Bilinear, and NearestNeighborIntpl.

7.8.3.4 void Interpolator::set_values (int *sizex*, double * *xptr*, double * *vptr*)

Set necessary values for interpolator. This method is called before interpolation is performed. Interpolator has access to the private data of respective classes, and access that information idrectly through pointers. This is for 1D.

Definition at line 17 of file Interpolator.cpp.

```
18 {
19     _xptr = xptr;
20     _yptr = vptr;
21     _sizex = sizex;
22 }
```

7.8.3.5 void Interpolator::set_values (int *sizex*, int *sizey*, double * *xptr*, double * *yptr*, double ** *vptr*)

Set necessary values for interpolator. This method is called before interpolation is performed. Interpolator has access to the private data of respective classes, and access that information idrectly through pointers. This is for 2D.

Definition at line 27 of file Interpolator.cpp.

```
28 {
29     _xptr = xptr;
30     _yptr = yptr;
31     _sizex = sizex;
32     _sizey = sizey;
33     _zzptr = vptr;
34 }
```

7.8.3.6 void Interpolator::set_values (int *sizex*, int *sizey*, int *sizez*, double * *xptr*, double * *yptr*, double * *zptr*, double *** *wptr*)

3D interpolator set value method. This function must be called before interpolation.

Definition at line 37 of file Interpolator.cpp.

```
38 {
39     _xptr = xptr;
40     _yptr = yptr;
41     _zptr = zptr;
42     _sizex = sizex;
43     _sizey = sizey;
44     _sizez = sizez;
45     _wptr = wptr;
46 }
```

7.8.4 Member Data Documentation

7.8.4.1 int Interpolator::_sizex [protected]

Definition at line 34 of file Interpolator.h.

7.8.4.2 int Interpolator::_sizey [protected]

Definition at line 36 of file Interpolator.h.

7.8.4.3 int Interpolator::_sizez [protected]

Definition at line 38 of file Interpolator.h.

7.8.4.4 double*** Interpolator::_wptr [protected]

Definition at line 48 of file Interpolator.h.

7.8.4.5 double* Interpolator::_xptr [protected]

Definition at line 40 of file Interpolator.h.

7.8.4.6 double* Interpolator::_yptr [protected]

Definition at line 42 of file Interpolator.h.

7.8.4.7 double* Interpolator::_zptr [protected]

Definition at line 44 of file Interpolator.h.

7.8.4.8 double** Interpolator::_zzptr [protected]

Definition at line 46 of file Interpolator.h.

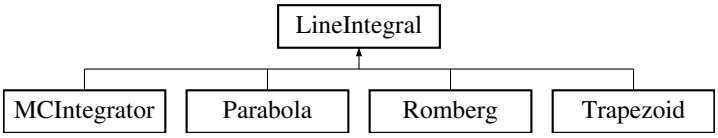
The documentation for this class was generated from the following files:

- include/Interpolator.h
- src/Interpolator.cpp

7.9 LineIntegral Class Reference

`#include <LineIntegral.h>`

Inheritance diagram for LineIntegral:



Public Member Functions

- LineIntegral ()
- virtual ~LineIntegral ()
- virtual double Integrate (std::function< double(double)>, double xmin, double xmax, double N)=0

  Performs integration from xmin to xmax, with N steps to the function object.

7.9.1 Detailed Description

LineIntegral base class. It has only one method that performs line integral according to a function f. This function f should be the parameterised function from a class.

Definition at line 10 of file LineIntegral.h.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 LineIntegral::LineIntegral ( )

Definition at line 3 of file LineIntegral.cpp.

```
3 {}
```

7.9.2.2 LineIntegral::~LineIntegral ( ) [virtual]

Definition at line 5 of file LineIntegral.cpp.

```
5 {}
```

7.9.3 Member Function Documentation

7.9.3.1 virtual double LineIntegral::Integrate (std::function< double(double)> , double *xmin,* double *xmax,* double *N*) [pure virtual]

Performs integration from xmin to xmax, with N steps to the function object.

Implemented in MCIntegrator, Parabola, Trapezoid, and Romberg.

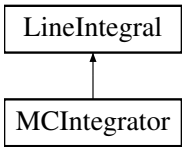
The documentation for this class was generated from the following files:

- include/LineIntegral.h
- src/LineIntegral.cpp

7.10 MCIntegrator Class Reference

`#include <MCIntegrator.h>`

Inheritance diagram for MCIntegrator:



Public Member Functions

- MCIntegrator ()
- ~MCIntegrator ()
- double Integrate (std::function< double(double)>, double xmin, double xmax, double N)

  Performs sampling using Monte-Carlo sampling.

7.10.1 Detailed Description

Monte-Carlo integrator. Performs integration using Monte-Carlo sampling.

Definition at line 11 of file MCIntegrator.h.

7.10.2 Constructor & Destructor Documentation

7.10.2.1 MCIntegrator::MCIntegrator ( )

Definition at line 5 of file MCIntegrator.cpp.

```
5 {}
```

7.10.2.2 MCIntegrator::∼MCIntegrator ( )

Definition at line 7 of file MCIntegrator.cpp.

```
7 {}
```

7.10.3 Member Function Documentation

7.10.3.1 double MCIntegrator::Integrate (std::function< double(double)> *f*, double *xmin*, double *xmax*, double *N*) [virtual]

Performs sampling using Monte-Carlo sampling.

< approximately equivalent number of required samples.

Implements LineIntegral.

Definition at line 9 of file MCIntegrator.cpp.

```
9                                                                                                    {
10      int N = 10*(xmax-xmin)/step;
11      double sum = 0;
12      for(int i=0;i<N;i++){
13      sum += f( xmin+1.0*rand()/RAND_MAX*(xmax-xmin));
14      }
15      return (xmax-xmin)*sum/N;
16 }
```

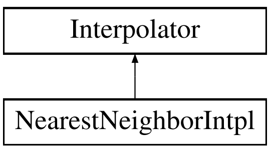
The documentation for this class was generated from the following files:

- include/MCIntegrator.h
- src/MCIntegrator.cpp

7.11 NearestNeighborIntpl Class Reference

```
#include <NearestNeighborIntpl.h>
```

Inheritance diagram for NearestNeighborIntpl:



Public Member Functions

- NearestNeighborIntpl ()

- ~NearestNeighborIntpl ()
- double Interpolate (double x, double y)

  Interpolation for 2D.
- double Interpolate (double x, double y, double z)

  Interpolation for 3D.

Additional Inherited Members

7.11.1 Detailed Description

Interpolate a function using nearest neighbour method.

Definition at line 11 of file NearestNeighborIntpl.h.

7.11.2 Constructor & Destructor Documentation

7.11.2.1 NearestNeighborIntpl::NearestNeighborIntpl ( )

Definition at line 5 of file NearestNeighborIntpl.cpp.

```
5 {}
```

7.11.2.2 NearestNeighborIntpl::~NearestNeighborIntpl ( )

Definition at line 6 of file NearestNeighborIntpl.cpp.

```
6 {}
```

7.11.3 Member Function Documentation

7.11.3.1 double NearestNeighborIntpl::Interpolate (double *x*, double *y*) [virtual]

Interpolation for 2D.

Implements Interpolator.

Definition at line 8 of file NearestNeighborIntpl.cpp.

```
8                                                       {
9     return 0;
10 }
```

7.11.3.2 double NearestNeighborIntpl::Interpolate (double *x*, double *y*, double *z*) [virtual]

Interpolation for 3D.

Implements Interpolator.

Definition at line 11 of file NearestNeighborIntpl.cpp.

```
11                                                                  {
12     return 0;
13 }
```

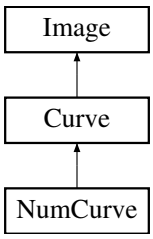
The documentation for this class was generated from the following files:

- include/NearestNeighborIntpl.h
- src/NearestNeighborIntpl.cpp

7.12 NumCurve Class Reference

`#include <NumCurve.h>`

Inheritance diagram for NumCurve:



Public Member Functions

- NumCurve ()

  Default constructor, everything null and 0.
- NumCurve (int size)

  Default constructor, _size set and everything else null and 0./Users/qili/Downloads/apc524_CT/include/NumCurve.h.
- NumCurve (int size, double r)

  Default constructor, _size and range set.
- NumCurve (int size, double *x, double *y)

  Initialize with a given x and y array.
- NumCurve (int size, double r, double *y)

  Initialize with a radius and an array.
- NumCurve (const NumCurve &)

  Copy constructor, same type as NumCurve.
- NumCurve & operator= (const NumCurve &)

  Assignment constructor for same type.
- NumCurve (int size, const Curve &)

  Copy constr. for general Curve obj. Needs size info.
- NumCurve (const char *file)
- void Copy (int size, const Curve &)

  !< Constructor from HDF5 file.
- ~NumCurve ()

  Destructor, has to delete stored data.
- double operator() (double, Interpolator *intpl=0) const

  Operator () to access lvalues, argument double will be rounded to nearest intger and be used to access.
- double & operator() (int)
- double & operator[ ] (int)

  This method can be used to set values at the integer nodes.
- void Print ()

  Default method, print out everything as two columns.
- void Print (double, double, int)

  Print for a given range.
- double * GetXPtr ()

  Returns a pointer to the array of coordinates for faster access.
- double * GetYPtr ()

  Returns a pointer to the array of values for faster access.
- int GetSize ()

  Return size of the data array.

Protected Attributes

- double * _datax

  X-Coordinates of the points.
- double * _datay

  Values at X-Coordinates.
- int _size

  size of the array.

7.12.1 Detailed Description

Definition at line 14 of file NumCurve.h.

7.12.2 Constructor & Destructor Documentation

7.12.2.1 NumCurve::NumCurve ( )

Default constructor, everything null and 0.

Implements numerical curve.

Default constructor, everything to NULL.

Definition at line 10 of file NumCurve.cpp.

```
10                     : Curve(0)
11 {
12     _datax = 0;
13     _datay = 0;
14     _size = 0;
15 }
```

7.12.2.2 NumCurve::NumCurve (int *size*)

Default constructor, _size set and everything else null and 0./Users/qili/Downloads/apc524_CT/include/Num↩Curve.h.

Constructor with a size input.

Definition at line 18 of file NumCurve.cpp.

```
18                             : Curve(0), _size(size)
19 {
20     _datax = new double[_size];
21     _datay = new double[_size];
22     for(int i=0;i<size;i++){
23         _datax[i]=0; _datay[i]=0;
24     }
25 }
```

7.12.2.3 NumCurve::NumCurve (int *size,* double *r*)

Default constructor, _size and range set.

Constructor with a size and a range.

Definition at line 27 of file NumCurve.cpp.

```
27                                       : Curve(r), _size(size)
28 {
29     _datax = new double[_size];
```

```
30     _datay = new double[_size];
31     for(int i=0;i<size;i++){
32         _datax[i]=-r+i*(2*r)/size;
33         _datay[i]=0;
34     }
35 }
```

7.12.2.4 NumCurve::NumCurve (int *size*, double * *x*, double * *y*)

Initialize with a given x and y array.

Constructor with a given array. < symmetrize the given array. Center them at 0.

Definition at line 38 of file NumCurve.cpp.

```
38                                                  : Curve(0), _size(size)
39 {
40         double avg = 0;
41     for(int i=0;i<_size;i++){avg += x[i];}
42     avg /= _size;
43     _datax = new double[_size];
44     _datay = new double[_size];
45     for(int i=0;i<_size;i++){
46         _datax[i] = x[i]-avg;    _datay[i] = y[i];
47     }
48     _r = fabs(_datax[0]) > fabs(_datax[_size-1]) ? fabs(_datax[0]) : fabs(
      _datax[_size-1]);
49 }
```

7.12.2.5 NumCurve::NumCurve (int *size*, double *r*, double * *y*)

Initialize with a radius and an array.

Constructor with a size, a range and a set of y-values. < convention: 0th point is -r, (n-1)th point is r. Includes both end-points.

Definition at line 52 of file NumCurve.cpp.

```
52                                                  : Curve(r), _size(size)
53 {
54     _datax = new double[_size];
55     _datay = new double[_size];
56     for(int i=0;i<_size;i++){
57         _datay[i] = y[i];
58         _datax[i] = -_r + i*(2.0*_r)/(_size-1);
59     }
60 }
```

7.12.2.6 NumCurve::NumCurve (const NumCurve & *f*)

Copy constructor, same type as NumCurve.

Copy constructor that takes in the same type. < Performs a deep copy.

Definition at line 63 of file NumCurve.cpp.

```
63                                      : Curve(f._r)
64 {
65     _size = f._size;
66     _datax = new double[_size];    _datay = new double[_size];
67     for(int i=0;i<_size;i++){
68         _datax[i] = f._datax[i];
69         _datay[i] = f._datay[i];
70     }
71 }
```

7.12.2.7 NumCurve::NumCurve (int *size,* const Curve & *f*)

Copy constr. for general Curve obj. Needs size info.

Constructor with a size and a Curve object. Use operator () to initialize. < Evaluate at _datax{} and assign the value to the new obj.

Definition at line 89 of file NumCurve.cpp.

```
89                                          : Curve(0)
90 {
91     _size = size;
92     _r = f.GetRange();
93     _datax = new double[_size];    _datay = new double[_size];
94     for(int i=0;i<_size;i++){
95         _datax[i] = -_r + i*(2*_r)/(_size-1);
96         _datay[i] = f(_datax[i],0);
97     }
98 }
```

7.12.2.8 NumCurve::NumCurve (const char * *file*)

7.12.2.9 NumCurve::~NumCurve ( )

Destructor, has to delete stored data.

Definition at line 113 of file NumCurve.cpp.

```
114 {
115     if(_datax!=0) delete [] _datax;
116     if(_datay!=0) delete [] _datay;
117 }
```

7.12.3 Member Function Documentation

7.12.3.1 void NumCurve::Copy (int *size,* const Curve & *f*)

!< Constructor from HDF5 file.

Assignment operator for construction.

Copy operator for general Curve, will use previous size information. < free memory if previously contains objects.

< Evaluate at _datax{} and assign the value to the new obj.

Definition at line 101 of file NumCurve.cpp.

```
102 {
103     if(_datax!=0) delete [] _datax;
104     if(_datay!=0) delete [] _datay;
105     _size = size; _r = f.GetRange();
106     _datax = new double[_size];    _datay = new double[_size];
107     for(int i=0;i<_size;i++){
108         _datax[i] = -_r + i*(2*_r)/(_size-1);
109         _datay[i] = f(_datax[i]);
110     }
111 }
```

7.12.3.2 int NumCurve::GetSize ( )

Return size of the data array.

Definition at line 200 of file NumCurve.cpp.

```
201 {
202     return _size;
203 }
```

7.12.3.3 double * NumCurve::GetXPtr ( )

Returns a pointer to the array of coordinates for faster access.

Definition at line 143 of file NumCurve.cpp.

```
144 {
145     return _datax;
146 }
```

7.12.3.4 double * NumCurve::GetYPtr ( )

Returns a pointer to the array of values for faster access.

Definition at line 148 of file NumCurve.cpp.

```
149 {
150     return _datay;
151 }
```

7.12.3.5 double NumCurve::operator() (double *x*, Interpolator * *intpl* = 0) const [virtual]

Operator () to access lvalues, argument double will be rounded to nearest intger and be used to access.

Implements Curve.

Definition at line 119 of file NumCurve.cpp.

```
120 {
121     intpl->set_values(_size,_datax,_datay);
122     return intpl->Interpolate(x);
123 }
```

7.12.3.6 double & NumCurve::operator() (int *index*)

Definition at line 134 of file NumCurve.cpp.

```
135 {
136     if(index<0 || index>_size-1){
137         fprintf(stderr,"Error: NumCurve::operator[] index %d out of range (%d,%d)\n",index,0,
      _size-1);
138         index = 0;
139     }
140     return _datay[index];
141 }
```

7.12.3.7 NumCurve & NumCurve::operator= (const NumCurve & *f*)

Assignment constructor for same type.

Copy assignment, used when modifying existing objects, so have to take care of memories. < free memory if previously contains objects.

< Performs a deep copy.

Definition at line 74 of file NumCurve.cpp.

```
75 {
76     if(_datax!=0) delete [] _datax;
77     if(_datay!=0) delete [] _datay;
78     _size = f._size;
79     _r = f._r;
```

```
80      _datax = new double[_size];     _datay = new double[_size];
81      for(int i=0;i<_size;i++){
82          _datax[i] = f._datax[i];
83          _datay[i] = f._datay[i];
84      }
85      return (*this);
86  }
```

7.12.3.8 double & NumCurve::operator[ ] (int *index*)

This method can be used to set values at the integer nodes.

Definition at line 125 of file NumCurve.cpp.

```
126 {
127     if(index<0 || index>_size-1){
128         fprintf(stderr,"Error: NumCurve::operator[] index %d out of range (%d,%d)\n",index,0,
      _size-1);
129         index = 0;
130     }
131     return _datay[index];
132 }
```

7.12.3.9 void NumCurve::Print ( ) [virtual]

Default method, print out everything as two columns.

Reimplemented from Curve.

Definition at line 153 of file NumCurve.cpp.

```
154 {
155     for(int i=0; i<_size; i++) printf("%.9f %.9f\n",_datax[i],_datay[i]);
156 }
```

7.12.3.10 void NumCurve::Print (double , double , int)

Print for a given range.

7.12.4 Member Data Documentation

7.12.4.1 double* NumCurve::_datax [protected]

X-Coordinates of the points.

Definition at line 45 of file NumCurve.h.

7.12.4.2 double* NumCurve::_datay [protected]

Values at X-Coordinates.

Definition at line 46 of file NumCurve.h.

7.12.4.3 int NumCurve::_size [protected]

size of the array.

Definition at line 47 of file NumCurve.h.

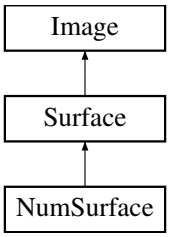
The documentation for this class was generated from the following files:

- include/NumCurve.h
- src/NumCurve.cpp

7.13 NumSurface Class Reference

`#include <NumSurface.h>`

Inheritance diagram for NumSurface:



Public Member Functions

- NumSurface ()

  Default constructor, everything null and 0.
- NumSurface (int sizex, int sizey)

  Default constructor, _size set and everything else null and 0.
- NumSurface (int sizex, double ∗x, int sizey, double ∗y, double ∗∗z)

  Initialize with a given x,y,z array.
- NumSurface (int sizex, double rx, int sizey, double ry, double ∗∗z)

  Initialize with a radius and an array.
- NumSurface (int sizex, double rx, int sizey, double ry)

  Initialize with a radius and range.
- NumSurface (const NumSurface &)

  Copy constructor, same type as NumSurface.
- NumSurface & operator= (const NumSurface &)

  Assignment constructor for same type.
- NumSurface (int sizex, int sizey, const Surface &)

  Copy constr. for general Surface obj. Needs size info.
- NumSurface (const char ∗file)

  constructor from hdf5 file.
- void Copy (int sizex, int sizey, const Surface &)

  Copy operator for general Surface, will use previous size information.
- ∼NumSurface ()

  Destructor, has to delete stored data.
- double operator() (double, double, Interpolator ∗) const

  Operator () to access lvalues, argument double will be rounded to nearest intger and be used to access.
- double & operator() (int, int)

  This method can be used to set values at the integer nodes.
- void Print ()

  Default method, print out everything as three columns.
- void Print (double, double, int, double, double, int)

  Print out image.
- double ∗ GetXPtr ()

Returns a pointer to x coordinates.

- double * GetYPtr ()

  Returns a pointer to y coordinates.

- double ** GetZPtr ()

  Returns a pointer to values on xy coordinates.

- int GetSizeX ()

  Returns the size in x direction.

- int GetSizeY ()

  Returns the size in y direction.

Protected Attributes

- double * _datax

  X-Coordinates of the points.

- double * _datay

  Y-Coordinates of the points.

- double ** _dataz

  *Z-Coordinates of the points. Since dataz will be like dataz[][], it will be double** instead of double*.*

- int _sizex

  size of the array in x direction.

- int _sizey

  size of the array in y direction.

7.13.1 Detailed Description

This class implements the Surface abstract class with numerical points. The values are obtained by interpolation.

Definition at line 14 of file NumSurface.h.

7.13.2 Constructor & Destructor Documentation

7.13.2.1 NumSurface::NumSurface ( )

Default constructor, everything null and 0.

Default constructor, everything to 0 or Null.

Definition at line 13 of file NumSurface.cpp.

```
13                          : Surface(0,0)
14 {
15     _datax = 0;
16     _datay = 0;
17     _dataz = 0;
18     _sizex = 0;
19     _sizey = 0;
20 }
```

7.13.2.2 NumSurface::NumSurface (int *sizex*, int *sizey*)

Default constructor, _size set and everything else null and 0.

Constructor with a size input. Range is still 0 so subsequently range must be adjusted.

Definition at line 23 of file NumSurface.cpp.

```
23                                          : Surface(0,0), _sizex(sizex),
      _sizey(sizey)
24 {
25     _datax = new double[_sizex];
26     _datay = new double[_sizey];
27     _dataz = new double*[_sizex];
28     for(int i=0;i<_sizex;i++)
29         _dataz[i] = new double[_sizey];
30 }
```

7.13.2.3 NumSurface::NumSurface (int *sizex*, double * *x*, int *sizey*, double * *y*, double ** *z*)

Initialize with a given x,y,z array.

Constructor with a given array. Range is read from the maximum of the array element. < x-direction, symmetrize the given array by subtracting the average.

< y-direction, symmetrize the given array. Center them at 0.

< Find the largest coordinate as the range.

< Don't forget to set the range.

Definition at line 33 of file NumSurface.cpp.

```
34 : Surface(0,0), _sizex(sizex), _sizey(sizey)
35 {
36     double avg = 0;
37     for(int i=0;i<_sizex;i++){avg += x[i];}
38     avg /= _sizex;
39     _datax = new double[_sizex];
40     for(int i=0;i<_sizex;i++){
41         _datax[i] = x[i]-avg;
42     }
43
44     avg = 0;
45     for(int i=0;i<_sizey;i++){avg += y[i];}
46     avg /= _sizey;
47     _datay = new double[_sizey];
48     for(int i=0;i<_sizey;i++){
49         _datay[i] = y[i]-avg;
50     }
51
52     _dataz = new double*[_sizex];
53     for(int i=0;i<_sizex;i++)
54         _dataz[i] = new double[_sizey];
55     for(int i=0;i<_sizex;i++)
56         for(int j=0;j<_sizey;j++)
57             _dataz[i][j] = z[i][j];
58
59     _rx = fabs(_datax[0]) > fabs(_datax[_sizex-1]) ? fabs(_datax[0]) : fabs(
      _datax[_sizex-1]);
60     _ry = fabs(_datay[0]) > fabs(_datay[_sizey-1]) ? fabs(_datay[0]) : fabs(
      _datay[_sizey-1]);
62     _r = sqrt(_rx*_rx+_ry*_ry);
63 }
```

7.13.2.4 NumSurface::NumSurface (int *sizex*, double *rx*, int *sizey*, double *ry*, double ** *z*)

Initialize with a radius and an array.

Constructor with a given size, range and a set of z-values. Assumes equal spacing. < include the endpoints.

< include endpoints.

Definition at line 66 of file NumSurface.cpp.

```
67 : Surface(rx,ry), _sizex(sizex), _sizey(sizey)
68 {
69     _datax = new double[_sizex];
70     _datay = new double[_sizey];
71     _dataz = new double*[_sizex];
72     for(int i=0;i<_sizex;i++){
73         _dataz[i] = new double[_sizey];
74         _datax[i] = -_rx + i*(2*_rx)/(_sizex-1);
```

```
75      }
76      for(int i=0;i<_sizey;i++){
77          _datay[i] = -_ry + i*(2*_ry)/(_sizey-1);
78      }
79      for(int i=0;i<_sizex;i++)
80          for(int j=0;j<_sizey;j++)
81              _dataz[i][j] = z[i][j];
82  }
```

7.13.2.5 NumSurface::NumSurface (int *sizex*, double *rx*, int *sizey*, double *ry*)

Initialize with a radius and range.

Constructor with a given size, range and a set of z-values. Coordinates are assumed to be uniformly spaced, and field values are initialized to 0. < include endpoints

< include endpoints

< **z is not specified, therefore initialize to 0.

Definition at line 85 of file NumSurface.cpp.

```
86  : Surface(rx,ry), _sizex(sizex), _sizey(sizey)
87  {
88      _datax = new double[_sizex];
89      _datay = new double[_sizey];
90      _dataz = new double*[_sizex];
91      for(int i=0;i<_sizex;i++){
92          _dataz[i] = new double[_sizey];
93          _datax[i] = -_rx + i*(2*_rx)/(_sizex-1);
94      }
95      for(int i=0;i<_sizey;i++){
96          _datay[i] = -_ry + i*(2*_ry)/(_sizey-1);
97      }
98      for(int i=0;i<_sizex;i++)
99          for(int j=0;j<_sizey;j++)
100             _dataz[i][j] = 0;
102 }
```

7.13.2.6 NumSurface::NumSurface (const NumSurface & *f*)

Copy constructor, same type as NumSurface.

Copy constructor that takes in the same type. Size and the array will be both read from the rhs. < read size from the NumSurface.

< Performs a deep copy.

Definition at line 105 of file NumSurface.cpp.

```
105                                         : Surface(f._rx, f._ry)
106 {
107     _sizex = f._sizex; _sizey = f._sizey;
109     _datax = new double[_sizex];
110     _datay = new double[_sizey];
111     _dataz = new double*[_sizex];
112     for (int i=0;i<_sizex;i++){
113         _datax[i] = f._datax[i];
114         _dataz[i] = new double[_sizey];
115     }
116     for (int i=0;i<_sizey;i++){
117         _datay[i] = f._datay[i];
118     }
119     for(int i=0;i<_sizex;i++){
120         for(int j=0;j<_sizey;j++){
121             _dataz[i][j] = f._dataz[i][j];
122         }
123     }
124 }
```

7.13.2.7 NumSurface::NumSurface (int *sizex*, int *sizey*, const Surface & *f*)

Copy constr. for general Surface obj. Needs size info.

Constructor with a size and a Surface object. Use operator () of Surface class to initialize. < used default interpolation method.

Definition at line 156 of file NumSurface.cpp.

```
156                                                                 : Surface(0,0)
157 {
158     _sizex = sizex;
159     _sizey = sizey;
160     _rx = f.GetRangeX();
161     _ry = f.GetRangeY();
162     _r = f.GetRange();
163     _datax = new double[_sizex];
164     _datay = new double[_sizey];
165     _dataz = new double*[_sizex];
166     for(int i=0;i<_sizex;i++){
167         _datax[i] = -_rx + i*(2*_rx)/(_sizex-1);
168         _dataz[i] = new double[_sizey];
169     }
170     for(int i=0;i<_sizey;i++){
171         _datay[i] = -_ry + i*(2*_ry)/(_sizey-1);
172     }
173     for(int i=0;i<_sizex;i++){
174         for(int j=0;j<_sizey;j++){
175             _dataz[i][j] = f(_datax[i],_datay[j],0);
177         }
178     }
179 }
```

7.13.2.8 NumSurface::NumSurface (const char * *file*)

constructor from hdf5 file.

7.13.2.9 NumSurface::~NumSurface ( )

Destructor, has to delete stored data.

Destructor. Must delete memory allocated.

Definition at line 210 of file NumSurface.cpp.

```
211 {
212     if(_datax!=0) delete [] _datax;
213     if(_datay!=0) delete [] _datay;
214     if(_dataz!=0) {
215         for(int i=0;i<_sizex;i++) delete [] _dataz[i];
216         delete [] _dataz;
217     }
218 }
```

7.13.3 Member Function Documentation

7.13.3.1 void NumSurface::Copy (int *sizex*, int *sizey*, const Surface & *f*)

Copy operator for general Surface, will use previous size information.

Copy method. This will copy values from Surface object specified in the argument. < Don't forget range.

Definition at line 182 of file NumSurface.cpp.

```
183 {
184     if(_datax!=0) delete [] _datax;
185     if(_datay!=0) delete [] _datay;
186     if(_dataz!=0) delete [] _dataz;
187     _sizex = sizex;
188     _sizey  = sizey;
```

```
189     _rx = f.GetRangeX();
190     _ry = f.GetRangeY();
191     _r = f.GetRange();
192     _datax = new double[_sizex];
193     _datay = new double[_sizey];
194     _dataz = new double*[_sizex];
195     for(int i=0;i<_sizex;i++){
196         _datax[i] = -_rx + i*(2*_rx)/(_sizex-1);
197         _dataz[i] = new double[_sizey];
198     }
199     for(int i=0;i<_sizey;i++){
200         _datay[i] = -_ry + i*(2*_ry)/(_sizey-1);
201     }
202     for(int i=0;i<_sizex;i++){
203         for(int j=0;j<_sizey;j++){
204             _dataz[i][j] = f(_datax[i],_datay[j],0);
205         }
206     }
207 }
```

7.13.3.2 int NumSurface::GetSizeX ( )

Returns the size in x direction.

Definition at line 256 of file NumSurface.cpp.

```
257 {
258     return _sizex;
259 }
```

7.13.3.3 int NumSurface::GetSizeY ( )

Returns the size in y direction.

Definition at line 261 of file NumSurface.cpp.

```
262 {
263     return _sizey;
264 }
```

7.13.3.4 double * NumSurface::GetXPtr ( )

Returns a pointer to x coordinates.

Return pointer to x-coordinates for fast access.

Definition at line 239 of file NumSurface.cpp.

```
240 {
241     return _datax;
242 }
```

7.13.3.5 double * NumSurface::GetYPtr ( )

Returns a pointer to y coordinates.

Return pointer to y-coordinates for fast access.

Definition at line 245 of file NumSurface.cpp.

```
246 {
247     return _datay;
248 }
```

7.13.3.6 double ** NumSurface::GetZPtr ( )

Returns a pointer to values on xy coordinates.

Return pointer to z-coordinates for fast access.

Definition at line 251 of file NumSurface.cpp.

```
252 {
253     return _dataz;
254 }
```

7.13.3.7 double NumSurface::operator() (double *x*, double *y*, Interpolator * *intpl*) const [virtual]

Operator () to access lvalues, argument double will be rounded to nearest intger and be used to access.

operator (double,double) will return values using the interpolation method used.

Implements Surface.

Definition at line 221 of file NumSurface.cpp.

```
222 {
223     intpl->set_values(_sizex,_sizey,_datax,_datay,
    _dataz);
224     return intpl->Interpolate(x,y);
225 }
```

7.13.3.8 double & NumSurface::operator() (int *indexX*, int *indexY*)

This method can be used to set values at the integer nodes.

operator(int,int) will retern REFERENCE to the function value at the given indexes. print out of range error to stderror.

Definition at line 228 of file NumSurface.cpp.

```
229 {
230     if(indexX < 0 || indexX >_sizex-1 || indexY <0 || indexY >_sizey-1){
231         fprintf(stderr,"Error: NumSurface::operator(,) index %d %d out of range.\n",indexX,indexY);
233         indexX=0; indexY=0;
234     }
235     return _dataz[indexX][indexY];
236 }
```

7.13.3.9 NumSurface & NumSurface::operator= (const NumSurface & *f*)

Assignment constructor for same type.

Copy assignment, used when modifying existing objects. If currently holds memory, must free it first. < if with memory, free it first.

< don't forget to copy range.

< Performs a deep copy.

Definition at line 127 of file NumSurface.cpp.

```
128 {
129     if(_datax!=0) delete [] _datax;
130     if(_datay!=0) delete [] _datay;
131     if(_dataz!=0) delete [] _dataz;
132     _sizex = f._sizex;
133     _sizey = f._sizey;
134     _rx = f._rx;
135     _ry = f._ry;
```

```
136     _r = f._r;
137     _datax = new double[_sizex];
138     _datay = new double[_sizey];
139     _dataz = new double*[_sizex];
140     for (int i=0;i<_sizex;i++){
141         _datax[i] = f._datax[i];
142         _dataz[i] = new double[_sizey];
143     }
144     for (int i=0;i<_sizey;i++){
145         _datay[i] = f._datay[i];
146     }
147     for(int i=0;i<_sizex;i++){
148         for(int j=0;j<_sizey;j++){
149             _dataz[i][j] = f._dataz[i][j];
150         }
151     }
152     return (*this);
153 }
```

7.13.3.10 void NumSurface::Print ( ) [virtual]

Default method, print out everything as three columns.

Default print method. Print out function values at the nodes.

Reimplemented from Surface.

Definition at line 267 of file NumSurface.cpp.

```
268 {
269     for(int j=_sizey-1;j>=0;j--){
270         for(int i=0;i<_sizex;i++)
271             printf("%.9f\t",_dataz[i][j]);
272         printf("\n");
273     }
274 }
```

7.13.3.11 void NumSurface::Print (double *xi*, double *xf*, int *Nx*, double *yi*, double *yf*, int *Ny*)

Print out image.

Print out the surface values within the specified range.

Definition at line 277 of file NumSurface.cpp.

```
278 {
279
280 }
```

7.13.4 Member Data Documentation

7.13.4.1 double* NumSurface::_datax [protected]

X-Coordinates of the points.

Definition at line 65 of file NumSurface.h.

7.13.4.2 double* NumSurface::_datay [protected]

Y-Coordinates of the points.

Definition at line 67 of file NumSurface.h.

7.13.4.3 double** NumSurface::_dataz [protected]

Z-Coordinates of the points. Since dataz will be like dataz[][], it will be double** instead of double*.

Definition at line 69 of file NumSurface.h.

7.13.4.4 int NumSurface::_sizex [protected]

size of the array in x direction.

Definition at line 71 of file NumSurface.h.

7.13.4.5 int NumSurface::_sizey [protected]

size of the array in y direction.

Definition at line 73 of file NumSurface.h.

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

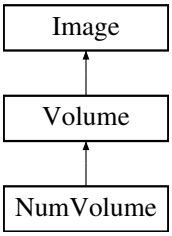
- include/NumSurface.h
- src/NumSurface.cpp

7.14 NumVolume Class Reference

This file defines numerical images whose data points are defined by discrete points.

`#include <NumVolume.h>`

Inheritance diagram for NumVolume:



Public Member Functions

- NumVolume ()

  Default constructor, everything null and 0.
- NumVolume (int sizex, int sizey, int sizez)

  Default constructor, _size set and everything else null and 0.
- NumVolume (int sizex, double *x, int sizey, double *y, int sizez, double *z, double ***w)

  Initialize with a given x,y,z array.
- NumVolume (int sizex, double rx, int sizey, double ry, int sizez, double rz, double ***w)

  Initialize with a radius and an array.
- NumVolume (int sizex, double rx, int sizey, double ry, int sizez, double rz)

  Initialize with a radius. W unknown.
- NumVolume (const NumVolume &)

  Copy constructor, same type as NumVolume.
- NumVolume & operator= (const NumVolume &)

Assignment constructor for same type.

- NumVolume (int sizex, int sizey, int sizez, const Volume &)

  Copy constr. for general Surface obj. Needs size info.
- void Copy (int sizex, int sizey, int sizez, const Volume &)

  Copy operator for general Surface, will use previous size information.
- NumVolume (const char *file)

  Constructor from a HDF5 file.
- ~NumVolume ()

  Destructor, has to delete stored data.
- double operator() (double, double, double, Interpolator *) const

  Operator () to access Ivalues, argument double will be rounded to nearest intger and be used to access.
- double & operator() (int, int, int)

  This method can be used to set values at the integer nodes.
- void Print ()

  Default method, print out everything as three columns.
- void Print (double, double, int, double, double, int)

  Print out image in the specified range.
- double * GetXPtr ()

  Return a pointer to x coordinates.
- double * GetYPtr ()

  Return a pointer to y coordinates.
- double * GetZPtr ()

  Return a pointer to z coordinates.
- int GetSizeX ()

  Returns the size in x direction.
- int GetSizeY ()

  Returns the size in y direction.
- int GetSizeZ ()

  Returns the size in z direction.
- double *** GetWPtr ()

  Return a pointer containing the data.

Protected Attributes

- double * _datax

  X-Coordinates of the points.
- double * _datay

  Y-Coordinates of the points.
- double * _dataz

  Z-Coordinates of the points.
- double *** _dataw

  dataw is a 3D array.
- int _sizex

  size of the array in x direction.
- int _sizey

  size of the array in y direction.
- int _sizez

  size of the array in z direction.

7.14.1 Detailed Description

This file defines numerical images whose data points are defined by discrete points.

This class implements the Surface abstract class with numerical points. The values are obtained by interpolation.

Definition at line 14 of file NumVolume.h.

7.14.2 Constructor & Destructor Documentation

7.14.2.1 NumVolume::NumVolume ( )

Default constructor, everything null and 0.

Implements numerical images.

Default constructor, everything to Null

Definition at line 12 of file NumVolume.cpp.

```
12                      : Volume(0,0,0)
13 {
14     _datax = 0;
15     _datay = 0;
16     _dataz = 0;
17     _dataw = 0;
18     _sizex = 0;
19     _sizey = 0;
20     _sizez = 0;
21 }
```

7.14.2.2 NumVolume::NumVolume (int *sizex*, int *sizey*, int *sizez*)

Default constructor, _size set and everything else null and 0.

Constructor with a size input.

Definition at line 24 of file NumVolume.cpp.

```
24                                                  : Volume(0,0,0), _sizex(sizex),
    _sizey(sizey),_sizez(sizez)
25 {
26     _datax = new double[_sizex];
27     _datay = new double[_sizey];
28     _dataz = new double[_sizez];
29     _dataw = new double**[_sizex];
30     for(int i=0;i<_sizex;i++){
31         _dataw[i] = new double*[_sizey];
32         for(int j=0;j<_sizey;j++){
33             _dataw[i][j] = new double[_sizez];
34         }
35     }
36 }
```

7.14.2.3 NumVolume::NumVolume (int *sizex*, double * *x*, int *sizey*, double * *y*, int *sizez*, double * *z*, double *** *w*)

Initialize with a given x,y,z array.

Constructor with a given array. < x-direction, symmetrize the given array. Center them at 0.

< y-direction,symmetrize the given array. Center them at 0.

< z-direction,symmetrize the given array. Center them at 0.

Definition at line 39 of file NumVolume.cpp.

```
40 : Volume(0,0,0), _sizex(sizex), _sizey(sizey), _sizez(sizez)
41 {
```

```
42     double avgx = 0;
43     for(int i=0;i<_sizex;i++){avgx += x[i];}
44     avgx /= _sizex;
45     _datax = new double[_sizex]; _dataw = new double**[_sizex];
46     for(int i=0;i<_sizex;i++){
47         _datax[i] = x[i]-avgx; _dataw[i] = new double*[_sizey];
48     }
49     double avgy = 0;
50     for(int i=0;i<_sizey;i++){avgy += y[i];}
51     avgy /= _sizey;
52     _datay = new double[_sizey];
53     for(int i=0;i<_sizey;i++){
54         _datay[i] = y[i]-avgy;
55     }
56     double avgz = 0;
57     for(int i=0;i<_sizez;i++){avgz += z[i];}
58     avgz /= _sizez;
59     _dataz = new double[_sizez];
60     for(int i=0;i<_sizez;i++){
61         _dataz[i] = z[i]-avgz;
62     }
63
64     for(int i=0;i<_sizex;i++){
65         for(int j=0;j<_sizey;j++){
66             for(int k=0;k<_sizez;k++){
67             _dataw[i][j][k] = w[i][j][k];
68             }
69         }
70     }
71     _rx = fabs(_datax[0]) > fabs(_datax[_sizex-1]) ? fabs(_datax[0]) : fabs(
       _datax[_sizex-1]);
72     _ry = fabs(_datay[0]) > fabs(_datay[_sizey-1]) ? fabs(_datay[0]) : fabs(
       _datay[_sizey-1]);
73     _rz = fabs(_dataz[0]) > fabs(_dataz[_sizez-1]) ? fabs(_dataz[0]) : fabs(
       _dataz[_sizez-1]);
74     _r = sqrt(_rx*_rx+_ry*_ry);
75 }
```

7.14.2.4 NumVolume::NumVolume (int *sizex*, double *rx*, int *sizey*, double *ry*, int *sizez*, double *rz*, double *** *w*)

Initialize with a radius and an array.

Constructor with a given size, range and a set of z-values.

Definition at line 78 of file NumVolume.cpp.

```
79 : Volume(rx,ry,rz), _sizex(sizex), _sizey(sizey),_sizez(sizez)
80 {
81     _datax = new double[_sizex];
82     _datay = new double[_sizey];
83     _dataz = new double[_sizez];
84     _dataw = new double**[_sizex];
85     for(int i=0;i<_sizex;i++){
86         _dataw[i] = new double*[_sizey];
87         _datax[i] = -_rx + i*(2*_rx)/(_sizex-1);
88     }
89     for(int i=0;i<_sizex;i++){
90         for(int j=0;j<_sizey;j++){
91             _dataw[i][j] = new double[_sizez];
92         }
93     }
94     for(int i=0;i<_sizey;i++){
95         _datay[i] = -_ry + i*(2*_ry)/(_sizey-1);
96     }
97     for(int i=0;i<_sizez;i++){
98         _dataz[i] = -_rz + i*(2*_rz)/(_sizez-1);
99     }
100     for(int i=0;i<_sizex;i++){
101         for(int j=0;j<_sizey;j++){
102             for(int k=0;k<_sizez;k++){
103                 _dataw[i][j][k] = w[i][j][k];
104             }
105         }
106     }
107 }
```

7.14.2.5 NumVolume::NumVolume (int *sizex*, double *rx*, int *sizey*, double *ry*, int *sizez*, double *rz*)

Initialize with a radius. W unknown.

Constructor with a given size, range but z-values are unknown.

Definition at line 110 of file NumVolume.cpp.

```
111 : Volume(rx,ry,rz), _sizex(sizex), _sizey(sizey),_sizez(sizez)
112 {
113     _datax = new double[_sizex];
114     _datay = new double[_sizey];
115     _dataz = new double[_sizez];
116     _dataw = new double**[_sizex];
117     for(int i=0;i<_sizex;i++){
118         _dataw[i] = new double*[_sizey];
119         _datax[i] = -_rx + i*(2*_rx)/(_sizex-1);
120     }
121     for(int i=0;i<_sizex;i++){
122         for(int j=0;j<_sizey;j++){
123             _dataw[i][j] = new double[_sizez];
124         }
125     }
126     for(int i=0;i<_sizey;i++){
127         _datay[i] = -_ry + i*(2*_ry)/(_sizey-1);
128     }
129     for(int i=0;i<_sizez;i++){
130         _dataz[i] = -_rz + i*(2*_rz)/(_sizez-1);
131     }
132 }
```

7.14.2.6 NumVolume::NumVolume (const NumVolume & *f*)

Copy constructor, same type as NumVolume.

Copy constructor that takes in the same type.

Definition at line 135 of file NumVolume.cpp.

```
135                                     : Volume(f._rx, f._ry, f._rz)
136 {
137     _sizex = f._sizex; _sizey = f._sizey; _sizez = f.
    _sizez;
138     _datax = new double[_sizex]; _datay = new double[_sizey];
    _dataz = new double[_sizez];
139     _dataw = new double**[_sizex];
140     for (int i=0;i<_sizex;i++){
141         _datax[i] = f._datax[i];
142         _dataw[i] = new double*[_sizey];
143     }
144     for(int i=0;i<_sizex;i++){
145         for(int j=0;j<_sizey;j++){
146             _dataw[i][j] = new double[_sizez];
147         }
148     }
149
150     for (int i=0;i<_sizey;i++){
151         _datay[i] = f._datay[i];
152     }
153     for (int i=0;i<_sizez;i++){
154         _dataz[i] = f._dataz[i];
155     }
156     for(int i=0;i<_sizex;i++){
157         for(int j=0;j<_sizey;j++){
158             for(int k=0;k<_sizez;k++){
159             _dataw[i][j][k] = f._dataw[i][j][k];
160             }
161         }
162     }
163 }
```

7.14.2.7 NumVolume::NumVolume (int *sizex*, int *sizey*, int *sizez*, const Volume & *f*)

Copy constr. for general Surface obj. Needs size info.

Constructor with a size and a Volume object. Use operator () to initialize.

Definition at line 204 of file NumVolume.cpp.

```
204                                                                     : Volume(0,0,0)
205 {
206     _sizex = sizex;
207     _sizey = sizey;
208     _sizez = sizez;
209     _rx = f.GetRangeX(); _ry = f.GetRangeY(); _rz = f.
    GetRangeZ();
210     _r = sqrt(_rx*_rx+_ry*_ry);
211     _datax = new double[_sizex];
212     _datay = new double[_sizey];
213     _dataz = new double[_sizez];
214     _dataw = new double**[_sizex];
215     for(int i=0;i<_sizex;i++){
216         _dataw[i] = new double*[_sizey];
217         _datax[i] = -_rx + i*(2*_rx)/(_sizex-1);
218     }
219     for(int i=0;i<_sizey;i++){
220         _datay[i] = -_ry + i*(2*_ry)/(_sizey-1);
221     }
222     for(int i=0;i<_sizez;i++){
223         _dataz[i] = -_rz + i*(2*_rz)/(_sizez-1);
224     }
225     for(int i=0;i<_sizex;i++){
226         for(int j=0;j<_sizey;j++){
227             _dataw[i][j] = new double[_sizez];
228         }
229     }
230     for(int i=0;i<_sizex;i++){
231         for(int j=0;j<_sizey;j++){
232             for(int k=0;k<_sizez;k++){
233             _dataw[i][j][k] = f(_datax[i],_datay[j],_dataz[k],0);
234             }
235         }
236     }
237 }
```

7.14.2.8 NumVolume::NumVolume (const char * *file*)

Constructor from a HDF5 file.

7.14.2.9 NumVolume::~NumVolume ( )

Destructor, has to delete stored data.

Numerical Volume destructor, frees memory.

Definition at line 283 of file NumVolume.cpp.

```
284 {
285     if(_datax!=0) delete [] _datax;
286     if(_datay!=0) delete [] _datay;
287     if(_dataz!=0) delete [] _dataz;
288     if(_dataw!=0){
289     for (int i = 0; i < _sizex; ++i) {
290         for (int j = 0; j < _sizey; ++j)
291             delete [] _dataw[i][j];
292         delete [] _dataw[i];
293     }
294     delete [] _dataw;
295       }
296 }
```

7.14.3 Member Function Documentation

7.14.3.1 void NumVolume::Copy (int *sizex*, int *sizey*, int *sizez*, const Volume & *f*)

Copy operator for general Surface, will use previous size information.

Assignment operator for construction.

Definition at line 240 of file NumVolume.cpp.

```
241 {
```

```
242     if(_datax!=0) delete [] _datax;
243     if(_datay!=0) delete [] _datay;
244     if(_dataz!=0) delete [] _dataz;
245     if(_dataw!=0) {
246         for (int i = 0; i < _sizex; ++i) {
247             for (int j = 0; j < _sizey; ++j)
248                 delete [] _dataw[i][j];
249             delete [] _dataw[i];
250         }
251         delete [] _dataw;
252     }
253     _sizex = sizex; _sizey  = sizey; _sizez = sizez;
254     _rx = f.GetRangeX(); _ry = f.GetRangeY();_rz = f.
      GetRangeZ();
255     _datax = new double[_sizex]; _datay = new double[_sizey];
256     _dataz = new double[_sizez]; _dataw = new double**[_sizex];
257
258     for(int i=0;i<_sizex;i++){
259         _dataw[i] = new double*[_sizey];
260         _datax[i] = -_rx + i*(2*_rx)/(_sizex-1);
261     }
262     for(int i=0;i<_sizey;i++){
263         _datay[i] = -_ry + i*(2*_ry)/(_sizey-1);
264     }
265     for(int i=0;i<_sizez;i++){
266         _dataz[i] = -_rz + i*(2*_rz)/(_sizez-1);
267     }
268     for(int i=0;i<_sizex;i++){
269         for(int j=0;j<_sizey;j++){
270             _dataw[i][j] = new double[_sizez];
271         }
272     }
273     for(int i=0;i<_sizex;i++){
274         for(int j=0;j<_sizey;j++){
275             for(int k=0;k<_sizez;k++){
276                 _dataw[i][j][k]= f(_datax[i],_datay[j],_dataz[k],0);
277             }
278         }
279     }
280 }
```

7.14.3.2 int NumVolume::GetSizeX ( )

Returns the size in x direction.

Definition at line 348 of file NumVolume.cpp.

```
349 {
350     return _sizex;
351 }
```

7.14.3.3 int NumVolume::GetSizeY ( )

Returns the size in y direction.

Definition at line 353 of file NumVolume.cpp.

```
354 {
355     return _sizey;
356 }
```

7.14.3.4 int NumVolume::GetSizeZ ( )

Returns the size in z direction.

Definition at line 358 of file NumVolume.cpp.

```
359 {
360     return _sizez;
361 }
```

7.14.3.5 double *** NumVolume::GetWPtr ( )

Return a pointer containing the data.

Definition at line 362 of file NumVolume.cpp.

```
363 {
364     return _dataw;
365 }
```

7.14.3.6 double * NumVolume::GetXPtr ( )

Return a pointer to x coordinates.

Definition at line 333 of file NumVolume.cpp.

```
334 {
335     return _datax;
336 }
```

7.14.3.7 double * NumVolume::GetYPtr ( )

Return a pointer to y coordinates.

Definition at line 338 of file NumVolume.cpp.

```
339 {
340     return _datay;
341 }
```

7.14.3.8 double * NumVolume::GetZPtr ( )

Return a pointer to z coordinates.

Definition at line 343 of file NumVolume.cpp.

```
344 {
345     return _dataz;
346 }
```

7.14.3.9 double NumVolume::operator() (double *x*, double *y*, double *z*, Interpolator * *intpl*) const [virtual]

Operator () to access lvalues, argument double will be rounded to nearest intger and be used to access.

Operator for returning value at a given point.

Implements Volume.

Definition at line 299 of file NumVolume.cpp.

```
300 {
301       intpl->set_values(_sizex,_sizey,_sizez,_datax,
    _datay,_dataz,_dataw);
302       return intpl->Interpolate(x,y,z);
303 }
```

7.14.3.10 double & NumVolume::operator() (int *indexX*, int *indexY*, int *indexZ*)

This method can be used to set values at the integer nodes.

Operator returning a reference to the data at the specified index.

Definition at line 306 of file NumVolume.cpp.

```
307 {
308     if(indexX < 0 || indexX >_sizex-1 || indexY <0 || indexY >_sizey-1|| indexZ <0 || indexZ >
    _sizez-1)
309     printf("Index out of range");
310     return _dataw[indexX][indexY][indexZ];
311 }
```

7.14.3.11 NumVolume & NumVolume::operator= (const NumVolume & *f*)

Assignment constructor for same type.

Copy assignment, used when modifying existing objects.

Definition at line 166 of file NumVolume.cpp.

```
167 {
168     if(_datax!=0) delete [] _datax;
169     if(_datay!=0) delete [] _datay;
170     if(_dataz!=0) delete [] _dataz;
171     if(_dataw!=0) delete [] _dataw;
172     _sizex = f._sizex; _sizey = f._sizey; _sizez = f.
    _sizez;
173     _rx = f._rx; _ry = f._ry; _rz = f._rz; _r = f._r;
174     _datax = new double[_sizex];
175     _datay = new double[_sizey];
176     _dataz = new double[_sizez];
177     _dataw = new double**[_sizex];
178     for (int i=0;i<_sizex;i++){
179         _datax[i] = f._datax[i];
180         _dataw[i] = new double*[_sizey];
181     }
182     for(int i=0;i<_sizex;i++){
183         for(int j=0;j<_sizey;j++){
184             _dataw[i][j] = new double[_sizez];
185         }
186     }
187     for (int i=0;i<_sizey;i++){
188         _datay[i] = f._datay[i];
189     }
190     for (int i=0;i<_sizez;i++){
191         _dataz[i] = f._dataz[i];
192     }
193     for(int i=0;i<_sizex;i++){
194         for(int j=0;j<_sizey;j++){
195             for(int k=0;k<_sizez;k++){
196                 _dataw[i][j][k] = f._dataw[i][j][k];
197             }
198         }
199     }
200     return (*this);
201 }
```

7.14.3.12 void NumVolume::Print ( ) [virtual]

Default method, print out everything as three columns.

Default print method.

Reimplemented from Volume.

Definition at line 314 of file NumVolume.cpp.

```
315 {
316     for (int k=0;k<_sizez;k++){
317         for(int j=_sizey-1;j>=0;j--){
```

```
318             for(int i=0;i<_sizex;i++){
319                 printf("%.9f\t",_dataw[i][j][k]);
320             }
321             printf("\n");
322         }
323     }
324
325 }
```

7.14.3.13 void NumVolume::Print (double *xi,* double *xf,* int *Nx,* double *yi,* double *yf,* int *Ny*)

Print out image in the specified range.

Print data in the range.

Definition at line 328 of file NumVolume.cpp.

```
328 {}
```

7.14.4 Member Data Documentation

7.14.4.1 double*** NumVolume::_dataw [protected]

dataw is a 3D array.

Definition at line 56 of file NumVolume.h.

7.14.4.2 double* NumVolume::_datax [protected]

X-Coordinates of the points.

Definition at line 53 of file NumVolume.h.

7.14.4.3 double* NumVolume::_datay [protected]

Y-Coordinates of the points.

Definition at line 54 of file NumVolume.h.

7.14.4.4 double* NumVolume::_dataz [protected]

Z-Coordinates of the points.

Definition at line 55 of file NumVolume.h.

7.14.4.5 int NumVolume::_sizex [protected]

size of the array in x direction.

Definition at line 57 of file NumVolume.h.

7.14.4.6 int NumVolume::_sizey [protected]

size of the array in y direction.

Definition at line 58 of file NumVolume.h.

7.14.4.7 int NumVolume::_sizez [protected]

size of the array in z direction.

Definition at line 59 of file NumVolume.h.

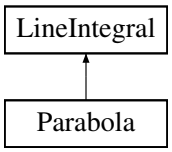
The documentation for this class was generated from the following files:

- include/NumVolume.h
- src/NumVolume.cpp

7.15 Parabola Class Reference

`#include <Parabola.h>`

Inheritance diagram for Parabola:



Public Member Functions

- Parabola ()
- ~Parabola ()
- double Integrate (std::function< double(double)>, double xmin, double xmax, double step)

  Implements the Integrate method.

7.15.1 Detailed Description

Parabola integration.

Definition at line 10 of file Parabola.h.

7.15.2 Constructor & Destructor Documentation

7.15.2.1 Parabola::Parabola ( )

Definition at line 3 of file Parabola.cpp.

```
3 {}
```

7.15.2.2 Parabola::~Parabola ( )

Definition at line 5 of file Parabola.cpp.

```
5 {}
```

7.15.3 Member Function Documentation

7.15.3.1 double Parabola::Integrate (std::function< double(double)> *f*, double *xmin*, double *xmax*, double *step*) [virtual]

Implements the Integrate method.

Implements LineIntegral.

Definition at line 7 of file Parabola.cpp.

```
7                                                                                                                    {
8      double sum = 0;
9      for (double i=xmin; i<xmax ; i += step){
10          sum += step*(f(i)+4*f(i+step/2)+f(i+step))/6;
11      }
12      return sum;
13 }
```

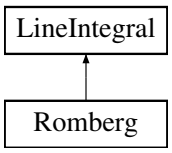
The documentation for this class was generated from the following files:

- include/Parabola.h
- src/Parabola.cpp

7.16 Romberg Class Reference

`#include <Romberg.h>`

Inheritance diagram for Romberg:



Public Member Functions

- Romberg ()
- ~Romberg ()
- double Integrate (std::function< double(double)>, double xmin, double xmax, double epsilon)

  Implements Integrate method from LineIntegral class.

7.16.1 Detailed Description

Romberg integration method.

Definition at line 9 of file Romberg.h.

7.16.2 Constructor & Destructor Documentation

7.16.2.1 Romberg::Romberg ( )

Definition at line 4 of file Romberg.cpp.

```
4 {}
```

7.16.2.2 Romberg::∼Romberg ( )

Definition at line 6 of file Romberg.cpp.

```
6 {}
```

7.16.3 Member Function Documentation

7.16.3.1 double Romberg::Integrate (std::function< double(double)> *f*, double *xmin*, double *xmax*, double *epsilon*) [virtual]

Implements Integrate method from LineIntegral class.

Implements LineIntegral.

Definition at line 8 of file Romberg.cpp.

```
8                                                                                                 {
9      double h = (xmax - xmin) / 2;
10      int k=1;
11      int n=1;
12      int m;
13      double **T;
14      T = new double*[50];
15      for(int j=0 ; j<50 ; j++){
16          T[j] = new double[100];
17      }
18      T[0][0] = h * (f(xmin) + f(xmax));
19      do{
20          double F = 0.0;
21          for(int i=1; i<=n; i++){
22              F += f(xmin + (2.0*i-1)*h);
23          }
24          double temp = T[0][k-1];
25          T[0][k] = temp/2.0 + h*F;
26
27          for(m=1; m<=k; m++){
28              double temp1 = T[m-1][k-m+1];
29              double temp2 = T[m-1][k-m];
30              T[m][k-m] = (pow(4.0,(double)m)*temp1 - temp2) / (pow(4.0,(double)m) - 1.0);
31          }
32          h /= 2.0;
33          n *= 2;
34          k++;
35
36      }while(fabs(T[m-1][0]-T[m-2][0]) > epsilon);
37
38      double I = T[m-1][0];
39
40      for(int j=0; j<50; j++){
41          delete[] T[j];
42      }
43      delete[] T;
44
45      return I;
46 }
```

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

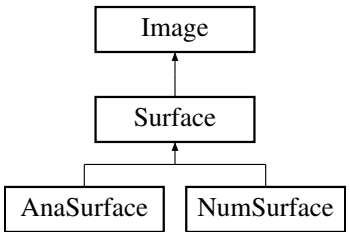
- include/Romberg.h
- src/Romberg.cpp

7.17 Surface Class Reference

Forward declaration.

`#include <Surface.h>`

Inheritance diagram for Surface:



Public Member Functions

- Surface (double, double)

  Constructor. Argument is the HALF-length in X and Y direction.
- Surface (double, double, double)

  Constructor for Romberg Integration Method. Last argument is epsilon.
- virtual ~Surface ()

  Virtual destructor, in case someone calls delete derived.
- virtual double operator() (double x, double y, Interpolator *intpl=0) const =0

  Returns image value at the argument point.
- virtual void Print ()

  Implement Image::Print. Note it is another virtual function.
- virtual void Print (double xmin, double xmax, int Nx, double ymin, double ymax, int Ny, Interpolator *intpl=0)

  Print out the field in the specified range.
- void SetRange (double rx, double ry)

  Sets symmetrized range in X and Y direction.
- double GetRangeX () const

  Returns symmetrized range in X direction.
- double GetRangeY () const

  Returns symmetrized range in Y direction.
- double GetRange () const

  Returns the smallest radius that would enclose the figure.
- void SetIntegralStep (double epsilon)

  Sets the stepsize to be used in the line integral.
- double GetIntegralStep () const

  Returns the step length of line integral.
- NumCurve GetProjection (LineIntegral *l, double angle=0, double spacing=0.01, Interpolator *intpl=0)

  Given angle, spacing of lines and an integration method, performs line integral. LineIntegral method is mandatory, and angle and spacing has default parameters. Angle starts from X-axis and moves counter-clockwise.
- double GetProjectionAtAngle (LineIntegral *l, double angle=0, double distance=0, Interpolator *intpl=0)

  Given angle, performs line integral. LineIntegral method is mandatory.Angle starts from X-axis and moves counter-clockwise.

Protected Attributes

- double _rx

  Range in X-direction.
- double _ry

  Range in Y-direction.
- double _r

  Smallest radius that would enclose the XY cross-section.
- double _step

  Integration step size in obtaining the projection.

7.17.1 Detailed Description

Forward declaration.

Definition at line 18 of file Surface.h.

7.17.2 Constructor & Destructor Documentation

7.17.2.1 Surface::Surface (double *rx*, double *ry*)

Constructor. Argument is the HALF-length in X and Y direction.

Definition at line 10 of file Surface.cpp.

```
10                                      : Image(Dim2){
11     _rx = rx; _ry = ry;
12     _r = sqrt(_rx*_rx+_ry*_ry);
13     _step = 0.001;
14 }
```

7.17.2.2 Surface::Surface (double , double , double)

Constructor for Romberg Integration Method. Last argument is epsilon.

7.17.2.3 Surface::~Surface ( ) [virtual]

Virtual destructor, in case someone calls delete derived.

Definition at line 16 of file Surface.cpp.

```
16 {}
```

7.17.3 Member Function Documentation

7.17.3.1 double Surface::GetIntegralStep ( ) const

Returns the step length of line integral.

7.17.3.2 NumCurve Surface::GetProjection (LineIntegral * *l*, double *angle* = 0, double *spacing* = 0.01, Interpolator * *intpl* = 0)

Given angle, spacing of lines and an integration method, performs line integral. LineIntegral method is mandatory, and angle and spacing has default parameters. Angle starts from X-axis and moves counter-clockwise.

Definition at line 26 of file Surface.cpp.

```
26                                                                                  {
27     int N = int(2*_r/spacing)+1;
28     NumCurve ret(N,_r);
29     double *x = ret.GetXPtr();
30     double *y = ret.GetYPtr();
31
32     for(int i=0; i < N; i++)
33         y[i] = this->GetProjectionAtAngle(l,angle,x[i],intpl);
34     return ret;
35 }
```

7.17.3.3 double Surface::GetProjectionAtAngle (LineIntegral * *l*, double *angle* = 0, double *distance* = 0, Interpolator * *intpl* = 0)

Given angle, performs line integral. LineIntegral method is mandatory.Angle starts from X-axis and moves counter-clockwise.

< t goes from -range to +range

< Uses lambda expression to pass a parameterised function to integral method in the LineIntegral class. Since lambda with [capture] cannot be used as function pointers, have to wrap it with std::function. Parameter angle is in radian. For a given angle, the function computes line integral along parallel lines with spacing spacing. The parameter t in the lambda is the dummy integration variable moving along the parameterised line. Integration starts and end at the edge defined by _r. d in the loop is the distance of the line to the reference line passing through the origin. S

Definition at line 41 of file Surface.cpp.

```
41                                                                                                  {
42     double ri = sqrt(_r*_r-d*d);
43         std::function<double (double)> fptr = [angle,d,ri,intpl,this](double t) -> double{
44             double temp = (*this)(-sin(angle)*(t-ri)+d*cos(angle),cos(angle)*(t-ri)+d*sin(angle),intpl);
45             return temp;
46         };
47         return l->Integrate(fptr, 0 , 2*ri, _step);
51 }
```

7.17.3.4 double Surface::GetRange ( ) const

Returns the smallest radius that would enclose the figure.

Definition at line 24 of file Surface.cpp.

```
24 { return _r; }
```

7.17.3.5 double Surface::GetRangeX ( ) const

Returns symmetrized range in X direction.

Definition at line 20 of file Surface.cpp.

```
20 { return _rx; }
```

7.17.3.6 double Surface::GetRangeY ( ) const

Returns symmetrized range in Y direction.

Definition at line 22 of file Surface.cpp.

```
22 { return _ry; }
```

7.17.3.7 virtual double Surface::operator() (double *x*, double *y*, Interpolator * *intpl* = 0) const [pure virtual]

Returns image value at the argument point.

Implemented in NumSurface, and AnaSurface.

7.17.3.8 void Surface::Print ( ) [virtual]

Implement Image::Print. Note it is another virtual function.

Implements Image.

Reimplemented in NumSurface.

Definition at line 53 of file Surface.cpp.

```
54 {
55   this->Print(-_rx,_rx,200,-_ry,_ry,200);
56 }
```

7.17.3.9 void Surface::Print (double *xmin,* double *xmax,* int *Nx,* double *ymin,* double *ymax,* int *Ny,* Interpolator * *intpl* = 0) [virtual]

Print out the field in the specified range.

Definition at line 58 of file Surface.cpp.

```
59 {
60     double stepx = (xmax-xmin)/Nx; double stepy = (ymax-ymin)/Ny;
61     for( double y = ymax; y >= ymin; y -= stepy){
62         for( double x = xmin; x <= xmax; x += stepx)
63             printf(" %.9f", (*this)(x,y,intpl));
64         printf("\n");
65     }
66 }
```

7.17.3.10 void Surface::SetIntegralStep (double *epsilon*)

Sets the stepsize to be used in the line integral.

Definition at line 37 of file Surface.cpp.

```
37                                          {
38     _step = epsilon;
39 }
```

7.17.3.11 void Surface::SetRange (double *rx,* double *ry*)

Sets symmetrized range in X and Y direction.

Definition at line 18 of file Surface.cpp.

```
18 { _rx = rx; _ry = ry; _r = sqrt(_rx*_rx+_ry*_ry); }
```

7.17.4 Member Data Documentation

7.17.4.1 double Surface::_r [protected]

Smallest radius that would enclose the XY cross-section.

Definition at line 63 of file Surface.h.

7.17.4.2 double Surface::_rx [protected]

Range in X-direction.

Definition at line 59 of file Surface.h.

7.17.4.3 double Surface::_ry [protected]

Range in Y-direction.

Definition at line 61 of file Surface.h.

7.17.4.4 double Surface::_step [protected]

Integration step size in obtaining the projection.

Definition at line 65 of file Surface.h.

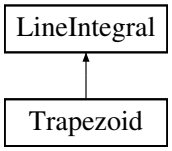
The documentation for this class was generated from the following files:

- include/Surface.h
- src/Surface.cpp

7.18 Trapezoid Class Reference

`#include <Trapezoid.h>`

Inheritance diagram for Trapezoid:



Public Member Functions

- Trapezoid ()
- ~Trapezoid ()
- double Integrate (std::function< double(double)>, double xmin, double xmax, double step)

  Implements LineIntegral::Integrate method.

7.18.1 Detailed Description

Trapezoid integration class.

Definition at line 9 of file Trapezoid.h.

7.18.2 Constructor & Destructor Documentation

7.18.2.1 Trapezoid::Trapezoid ( )

Definition at line 3 of file Trapezoid.cpp.

```
3 {}
```

7.18.2.2 Trapezoid::~Trapezoid ( )

Definition at line 5 of file Trapezoid.cpp.

```
5 {}
```

7.18.3 Member Function Documentation

7.18.3.1 double Trapezoid::Integrate (std::function< double(double)> *f*, double *xmin*, double *xmax*, double *step*) [virtual]

Implements LineIntegral::Integrate method.

Implements LineIntegral.

Definition at line 7 of file Trapezoid.cpp.

```
7                                                                                                    {
8     double sum = 0;
9     for(double i=xmin;i<xmax;i+=step){
10          sum += step*(f(i)+f(i+step))/2;
11     }
12     return sum;
13 }
```

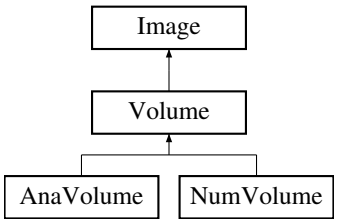
The documentation for this class was generated from the following files:

- include/Trapezoid.h
- src/Trapezoid.cpp

7.19 Volume Class Reference

`#include <Volume.h>`

Inheritance diagram for Volume:



Public Member Functions

- Volume (double, double, double)
- virtual ~Volume ()

  Constructor. Argument is the symmetrical range along each dim.
- virtual double operator() (double x, double y, double z, Interpolator *intpl) const =0

  Virtual destructor, in case someone calls delete derived.
- virtual void Print ()

  Returns image value at that integer point. S.
- virtual void Print (double xmin, double xmax, int Nx, double ymin, double ymax, int Ny, double z, Interpolator *intpl=0)

  Print the cross-section at z.
- void SetRange (double rx, double ry, double rz)

  Sets symmetrized range in X direction.
- double GetRangeX () const

  Returns symmetrized range in X direction.
- double GetRangeY () const

  Returns symmetrized range in Y direction.

- double GetRangeZ () const

  Returns symmetrized range in Z direction.
- double GetRadius () const

  Returns smallest radius that would enclose the X-Y cross-section. S.
- void SetIntegralStep (double epsilon)

  Sets integration stepsize.
- NumSurface GetProjection3D (LineIntegral *l, double angle=0, double spacingr=0.01, double spacingz=0.01, Interpolator *intpl=0)

  Given angle, spacing of lines and an integration method, performs line integral. LineIntegral method is mandatory, and angle and spacing has default parameters. Angle starts from X-axis and moves counter-clockwise.
- NumCurve GetProjection (LineIntegral *l, double angle=0, double spacingr=0.01, double z=0, Interpolator *intpl=0)

  Returns the projection along a particular angle and at a particular height.
- double GetProjectionAtAngle (LineIntegral *l, double angle_arg=0, double d=0, double z=0, Interpolator *intpl=0)

  Given angle, performs line integral at height z. LineIntegral method is mandatory. Angle starts from X-axis and moves counter-clockwise.

Protected Attributes

- double _rx
- double _ry
- double _rz
- double _r
- double _step

7.19.1 Detailed Description

Volume is an Image with dimension 3. A few further functions to consider, such as projections along different angles and creating a volume from surfaces.

Definition at line 17 of file Volume.h.

7.19.2 Constructor & Destructor Documentation

7.19.2.1 Volume::Volume (double *rx*, double *ry*, double *rz*)

Definition at line 7 of file Volume.cpp.

```
7                                                :Image(Dim3)
8 {
9     _rx = rx; _ry = ry; _rz = rz;
10     _r = sqrt(_rx*_rx+_ry*_ry);
11     _step = 0.001;
12 }
```

7.19.2.2 Volume::~Volume ( ) [virtual]

Constructor. Argument is the symmetrical range along each dim.

Definition at line 14 of file Volume.cpp.

```
14 {}
```

7.19.3 Member Function Documentation

7.19.3.1 NumCurve Volume::GetProjection (LineIntegral * *l*, double *angle* = 0, double *spacingr* = 0.01, double *z* = 0, Interpolator * *intpl* = 0)

Returns the projection along a particular angle and at a particular height.

Returns projection curve at a particular angle and height.

Definition at line 34 of file Volume.cpp.

```
35 {
36     int N = int(2*_r/spacing)+1;
37     NumCurve ret(N,_r);
38     double *x = ret.GetXPtr();
39     double *y = ret.GetYPtr();
40 
41     for(int i=0; i < N; i++){
42         y[i] = this->GetProjectionAtAngle(l,angle,x[i],z,intpl);
43          }
44     return ret;
45 }
```

7.19.3.2 NumSurface Volume::GetProjection3D (LineIntegral * *l*, double *angle* = 0, double *spacingr* = 0.01, double *spacingz* = 0.01, Interpolator * *intpl* = 0)

Given angle, spacing of lines and an integration method, performs line integral. LineIntegral method is mandatory, and angle and spacing has default parameters. Angle starts from X-axis and moves counter-clockwise.

7.19.3.3 double Volume::GetProjectionAtAngle (LineIntegral * *l*, double *angle_arg* = 0, double *d* = 0, double *z* = 0, Interpolator * *intpl* = 0)

Given angle, performs line integral at height z. LineIntegral method is mandatory. Angle starts from X-axis and moves counter-clockwise.

Returns the projection at a certain angle, distance and height. < t goes from -range to +range

Definition at line 48 of file Volume.cpp.

```
49 {
50     double ri = sqrt(_r*_r-d*d);
52     std::function<double (double)> fptr = [angle,d,z,ri,intpl,this](double t) -> double
53     {
54         double temp = (*this)((ri-t)*sin(angle)+d*cos(angle),(t-ri)*cos(angle)+d*sin(angle),
      z,intpl);
55         return temp;
56     };
57     return l->Integrate(fptr, 0 , 2*ri, _step);
58 }
```

7.19.3.4 double Volume::GetRadius ( ) const

Returns smallest radius that would enclose the X-Y cross-section. S.

Definition at line 25 of file Volume.cpp.

```
25 { return _r; }
```

7.19.3.5 double Volume::GetRangeX ( ) const

Returns symmetrized range in X direction.

Definition at line 22 of file Volume.cpp.

```
22 { return _rx; }
```

7.19.3.6 double Volume::GetRangeY ( ) const

Returns symmetrized range in Y direction.

Definition at line 23 of file Volume.cpp.

```
23 { return _ry; }
```

7.19.3.7 double Volume::GetRangeZ ( ) const

Returns symmetrized range in Z direction.

Definition at line 24 of file Volume.cpp.

```
24 { return _rz; }
```

7.19.3.8 virtual double Volume::operator() (double *x*, double *y*, double *z*, Interpolator * *intpl*) const [pure virtual]

Virtual destructor, in case someone calls delete derived.

Implemented in AnaVolume, and NumVolume.

7.19.3.9 void Volume::Print ( ) [virtual]

Returns image value at that integer point. S.

Default print method, will call overloaded print method.

Implements Image::Print().

Implements Image.

Reimplemented in NumVolume.

Definition at line 61 of file Volume.cpp.

```
62 { this->Print(-_rx,_rx,200,-_ry,_ry,200,0.0); }
```

7.19.3.10 void Volume::Print (double *xmin*, double *xmax*, int *Nx*, double *ymin*, double *ymax*, int *Ny*, double *z*, Interpolator * *intpl* = 0) [virtual]

Print the cross-section at z.

Print the volume information in the range specified.

Definition at line 65 of file Volume.cpp.

```
66 {
67     double stepx = (xmax-xmin)/Nx; double stepy = (ymax-ymin)/Ny;
68     for( double y = ymax; y >= ymin; y -= stepy){
69         for( double x = xmin; x <= xmax; x += stepx)
70             printf(" %.9f", (*this)(x,y,z,intpl));
71         printf("\n");
72     }
73 }
```

7.19.3.11 void Volume::SetIntegralStep (double *epsilon*)

Sets integration stepsize.

This is the integration step size. For Romberg method this is the precision.

Definition at line 28 of file Volume.cpp.

```
29 {
30     _step = epsilon;
31 }
```

7.19.3.12 void Volume::SetRange (double *rx*, double *ry*, double *rz*)

Sets symmetrized range in X direction.

Definition at line 16 of file Volume.cpp.

```
17 {
18     _rx = rx; _ry = ry; _rz = rz;
19     _r = sqrt(_rx*_rx+_ry*_ry);
20 }
```

7.19.4 Member Data Documentation

7.19.4.1 double Volume::_r [protected]

Definition at line 57 of file Volume.h.

7.19.4.2 double Volume::_rx [protected]

Definition at line 57 of file Volume.h.

7.19.4.3 double Volume::_ry [protected]

Definition at line 57 of file Volume.h.

7.19.4.4 double Volume::_rz [protected]

Definition at line 57 of file Volume.h.

7.19.4.5 double Volume::_step [protected]

Definition at line 57 of file Volume.h.

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

- include/Volume.h
- src/Volume.cpp

Chapter 8

File Documentation

8.1 demo/demoAna2D.cpp File Reference

```
#include "AnaImage.h"
#include "FilteredBackProjection.h"
#include "Trapezoid.h"
#include "TestFunctions.h"
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
```

Functions

- int main (int argc, char ∗argv[ ])

8.1.1 Function Documentation

8.1.1.1 int main (int *argc,* char ∗ *argv[ ]*)

demoAna2D is a demo for 2D Analytical functions to do X-ray simulation and reconstruction. Five analytical functions can be chosen. They are "Batman", Gauss2D, Rectangle, Circle and Triangle. USAGE: ./bin/demoAna2D (number) number is 0∼5 0: Batman, 1: Gauss2D, 2: Circle, 3: Rectangle, 4: Triangle < range of the geometry

< number of view

< resolution/ N of point in the projected curve.

< array containing size angles.

< since 180 symmetry, do not include endpoint.

< a 2D function.

< integ. method

< Num Surf to contain reconstructed result.

Definition at line 13 of file demoAna2D.cpp.

```
13                                {
14
15     int choice = -1;
16     if (argc > 1) choice = atoi(argv[1]);
17
```

```
18      double range = 8;
19      const int size=100;
20      const int Nres=400;
21      double angle[size];
22
23      ImageArray array;
24      for(int i=0;i<size;i++) angle[i] = 0 + i*pi/size;
26      AnaSurface* gauss;
27      if(choice < 0 || choice > 4){
28          printf("USAGE: %s (number)\n number is 0~4\n 0: Batman, 1: Gauss2D, 2: Circle, 3: Rectangle, 4:
        Triangle\n",argv[0]);
29          return -1;
30      }
31
32      if(choice == 0) gauss = new AnaSurface (Batman, range, range);
33      if(choice == 1) gauss = new AnaSurface (Gauss2D, range, range);
34      if(choice == 2) gauss = new AnaSurface (Circle, range, range);
35      if(choice == 3) gauss = new AnaSurface (Rectangle, range, range);
36      if(choice == 4) gauss = new AnaSurface (Triangle, range, range);
38      LineIntegral* l;
39      Trapezoid t; l = &t;
40      NumSurface *sf;
41
42      for(int i=0; i<size; i++){
43          cerr<<"Projecting at angle "<< angle[i]<<endl;
44          array.PushBack(angle[i], gauss->GetProjection(l,angle[i],0.1));
45      }
46      sf = (FilteredBackProjection(array,Nres,Hamming));
47  #ifdef USE_HDF
48      sf->ExportHDF("outAna2D.h5");
49  #endif
50      delete sf;
51      delete gauss;
52      return 0;
53  }
```

8.2 demo/demoAna3D.cpp File Reference

```
#include "AnaImage.h"
#include "Trapezoid.h"
#include "TestFunctions.h"
#include "FilteredBackProjection.h"
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
```

Functions

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

8.2.1 Function Documentation

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

demoAna3D is a demo for 3D analytical functions to do X-ray simulation and reconstruction. Four analytical functions can be chosen. They are Gauss3D, heart, sphere, cube. USAGE: ./bin/demoAna3D (number) number is 0∼3 0: Gauss3D, 1: Heart, 2: Sphere, 3: Cube < range of the geometry

< number of view per slice

< number of projected horizontal slice

< total number of view

< resolution/ N of point in the projected curve.

< array containing sizeT angles.

< array containing height.

< distance between each projected horizontal slice.

< set the number of projected horizontal slice in array.

< set the spacing in z to obtain projection.

< a 3D function.

< integ. method

< Num Surf to contain reconstructed result.

Definition at line 13 of file demoAna3D.cpp.

```
13                                {
14     int choice = -1;
15     if(argc > 1) choice = atoi(argv[1]);
16 
17     double range = 2;
18     const int size= 30;
19     const int slice=50;
20     const int sizeT = size*slice;
21     const int Nres=50;
22     double angle[sizeT];
23     double height[slice];
24     double spacingz;
25     ImageArray array;
26     array.SetSlice(slice);
27     spacingz = 2*range/slice;
28     for(int k=0;k<slice;k++){
29         height[k] = -range+spacingz*k;
30         for(int i=0;i<size;i++) {
31             angle[i+k*size] = 0 + i*pi/size;
32         }
33     }
34     Volume* gauss;
35     if(choice == -1){
36         printf("USAGE: %s (number)\n number is 0~3\n 0: Gauss3D, 1: Heart, 2: Sphere, 3: Cube\n", argv[0]);
37         return -1;
38     }
39     if(choice == 0) gauss = new AnaVolume (Gauss3D, range, range, range);
40     if(choice == 1) gauss = new AnaVolume (Heart, range, range, range);
41     if(choice == 2) gauss = new AnaVolume (Sphere, range, range, range);
42     if(choice == 3) gauss = new AnaVolume (Cube, range, range, range);
44     LineIntegral* l;
45     Trapezoid t; l = &t;
46     NumVolume *sf;
47 
48     for(int k=0;k<slice;k++){
49         cerr<<"Projecting at height "<<height[k]<<endl;
50         for(int i=0; i<size; i++){
51             cerr<<"Projecting at angle "<< angle[i+k*size]<<endl;
52             array.PushBack(angle[i+k*size], height[k], gauss->
      GetProjection(l,angle[i+k*size],0.05,height[k]));
53         }
54     }
55     sf = (FilteredBackProjection3D(array,Nres,Hamming));
56     cerr<<"Done running FBP3D"<<endl;
57 #ifdef USE_HDF
58     sf->ExportHDF("outAna3D.h5");
59 #endif
60     delete sf;
61     delete gauss;
62     return 0;
63 }
```

8.3 demo/demoNum2D.cpp File Reference

```
#include "Trapezoid.h"
#include "FilteredBackProjection.h"
#include "TestFunctions.h"
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
```

Functions

- int main (int argc, char ∗argv[ ])

8.3.1 Function Documentation

8.3.1.1 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 10 of file demoNum2D.cpp.

```
10                                {
11 #ifdef USE_HDF
12     double range = .5;  // range of the geometry
13     const int size=100; // number of view
14     const int Nres=400;// resolution/ N of point in the projected curve.
15     double angle[size]; // array containing size angles.
16
17     ImageArray array;
18     for(int i=0;i<size;i++) angle[i] = 0 + i*pi/size;
19             //  since 180 symmetry, do not include endpoint.
20     NumSurface object;
21     if (argc == 1){
22         object=NumSurface("./input/spine.h5");
23     }
24     else{
25         object=NumSurface(argv[1]);    //argv[1] is the path of the input file. e.g.
       argv[1]="./input/spine.h5"
26     }
27
28     NumSurface* gauss = &object;
29     LineIntegral* l;
30     Trapezoid t; l = &t;   // integ. method
31     NumSurface *sf;   // Num Surf to contain reconstructed result.
32
33     Bilinear intpl_nnb;
34     Interpolator* intpl = &intpl_nnb;
35     for(int i=0; i<size; i++){
36         cerr<<"Projecting at angle "<< angle[i]<<endl;
37         array.PushBack(angle[i], gauss->GetProjection(l,angle[i],0.01,intpl));
38     }
39     sf = (FilteredBackProjection(array,Nres,Hamming));
40     // filtered back-projection
41 // The file will automatically be stored in the output directory
42     sf->ExportHDF("out2D.h5");
43     delete sf;
44 #else
45     fprintf(stderr,"This demo requires HDF5 libraries. Please enable them by\n");
46     fprintf(stderr,"\n\t\tmake USE_HDF=1\n\n");
47 #endif
48     return 0;
49 }
```

8.4 demo/demoNum3D.cpp File Reference

```
#include "Trapezoid.h"
#include "FilteredBackProjection.h"
#include "TestFunctions.h"
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
```

Functions

- int main (int argc, char ∗argv[ ])

8.4.1 Function Documentation

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

Definition at line 10 of file demoNum3D.cpp.

```
10                                {
11 #ifdef USE_HDF
12     double range = 0.5; // range of the geometry
13     const int size= 100;    // number of view per slice
14     const int slice=30; // number of projected horizontal slice
15     const int sizeT = size*slice; // total number of view
16     const int Nres=128;// resolution/ N of point in the projected curve.
17     double angle[sizeT]; // array containing sizeT angles.
18     double height[slice]; // array containing height.
19     double spacingz; // distance between each projected horizontal slice.
20     ImageArray array;
21     array.SetSlice(slice); // set the number of projected horizontal slice in array.
22     spacingz = 2*range/slice; // set the spacing in z to obtain projection.
23     for(int k=0;k<slice;k++){
24         height[k] = -range+spacingz*k;
25         for(int i=0;i<size;i++) {
26             angle[i+k*size] = 0 + i*pi/size;
27         }
28     } //  since 180 symmetry, do not include endpoint.
29
30     NumVolume object;
31     if (argc == 1){
32         object=NumVolume("./input/brain.h5");
33     }
34     else{
35         object=NumVolume(argv[1]);    //argv[1] is the path of the input file. e.g.
       argv[1]="./input/brain.h5"
36     }
37     NumVolume* gauss = &object;
38     Bilinear intpl_nnb;
39     Interpolator* intpl = &intpl_nnb;
40
41     LineIntegral* l;
42     Trapezoid t; l = &t;   // integ. method
43     NumVolume *sf; // Num Surf to contain reconstructed result.
44
45     for(int k=0;k<slice;k++){
46         cerr<<"Projecting at height "<<height[k]<<endl;
47         for(int i=0; i<size; i++){
48             cerr<<"Projecting at angle "<< angle[i+k*size]<<endl;
49             NumCurve gauss_tmp;
50             gauss_tmp = gauss->GetProjection(l,angle[i+k*size],0.01,height[k],intpl); //
     spacingr =0.01
51             array.PushBack(angle[i+k*size], height[k], gauss_tmp);
52         }
53     }
54     cerr<<"running FBP3D"<<endl;
55     sf = (FilteredBackProjection3D(array,Nres,Hamming));
56     cerr<<"Done running FBP3D"<<endl;
57 // File will automatically be stored in output directory
58     sf->ExportHDF("out3D.h5");
59     cerr<<"doneHDF"<<endl;
60     delete sf;
61 #else
62     fprintf(stderr,"This demo requires HDF5 libraries. Please enable them by\n");
63     fprintf(stderr,"\n\t\tmake USE_HDF=1\n\n");
64 #endif
65     return 0;
66 }
```

8.5 include/AnaImage.h File Reference

```
#include "Image.h"
#include "Curve.h"
#include "Surface.h"
#include "Volume.h"
#include "Interpolator.h"
#include "globals.h"
```

Classes

- class AnaCurve
- class AnaSurface
- class AnaVolume

8.6 include/Bilinear.h File Reference

```
#include "Interpolator.h"
#include <vector>
```

Classes

- class Bilinear

8.7 include/Curve.h File Reference

Curves represent the projection of surfaces along some direction.

```
#include "Image.h"
#include "Interpolator.h"
```

Classes

- class Curve

8.7.1 Detailed Description

Curves represent the projection of surfaces along some direction.

8.8 include/FilteredBackProjection.h File Reference

```
#include "Image.h"
#include "Curve.h"
#include "NumSurface.h"
#include "NumVolume.h"
#include "ImageArray.h"
#include "globals.h"
```

Functions

- NumSurface * FilteredBackProjection (ImageArray &, int Nres=100, double(*kernal)(int, double)=Hamming)

  From input ImageArray, performs filtered back-projection to reconstruct. Nres is the resolution of the final image / number of point along one axis. kernal is used to convolve the projection image. By default it is the Hamming function.
- NumVolume * FilteredBackProjection3D (ImageArray &, int Nres=100, double(*kernal)(int, double)=Hamming)

From input ImageArray, performs filtered back-projection in 3D to reconstruct. Nres is the resolution of the final image / number of point along one axis. kernal is used to convolve the projection image. By default it is the Hamming function. ImageArray should contain more than one horizontal slice.

8.8.1 Function Documentation

8.8.1.1 NumSurface* FilteredBackProjection (ImageArray & , int *Nres* = 100, double(*)(int, double) *kernal* = Hamming)

From input ImageArray, performs filtered back-projection to reconstruct. Nres is the resolution of the final image / number of point along one axis. kernal is used to convolve the projection image. By default it is the Hamming function.

Nres is the resolution for the reconstructed surface.

Definition at line 8 of file FilteredBackProjection.cpp.

```
9 {
11     const double range = array.GetRange();
12     Bilinear bilin;
13     array.ConvolveWithKernal(kernal);   // Array filters each curve with the kernal.
14     std::cerr<<"FBP: Convolution done."<<std::endl;
15     NumSurface* rec = new NumSurface(Nres,range,Nres,range);    // NumSurface to store
       the reconstructed obj.
16     int Nangle = array.GetSize();
17     std::cerr<<"Size is."<< Nangle <<std::endl;
18     for(int ll=0; ll<Nangle; ll++){ // Iterate over all angle of view
19         double angle = array.GetAngle(ll);
20         std::cerr<<"Angle is."<< angle <<std::endl;
21         for( int i=0; i<Nres; i++){ // Loop over X-coordinates of final surface.
22             double x = -range + i*2*range/(Nres-1);
23             for( int j=0; j<Nres; j++){ // Loop over y-coordinate of final surface.
24                 double y = -range + j*2*range/(Nres-1);
25                 double t = x*cos(angle) + y*sin(angle); // Distance from (x,y) to origin at angle ll.
26                 //std::cerr<<"t is."<< t <<" "<<i<<" "<<j<< "range is "<< range<<std::endl;
27                 (*rec)(i,j) += (array.GetFilteredCurve(ll))(t,&bilin)*pi/Nangle; // Superpose all values,
       assuming uniform grid.
28             }
29         }
30         std::cerr<<"running" <<std::endl;
31 //        #ifdef USE_HDF
32 //        char file[100]; sprintf(file,"output/batman_rec%d.h5",ll);
33 //        rec->ExportHDF(file);
34 //        #endif
35     }
36     return rec;
37 }
```

8.8.1.2 NumVolume* FilteredBackProjection3D (ImageArray & , int *Nres* = 100, double(*)(int, double) *kernal* = Hamming)

From input ImageArray, performs filtered back-projection in 3D to reconstruct. Nres is the resolution of the final image / number of point along one axis. kernal is used to convolve the projection image. By default it is the Hamming function. ImageArray should contain more than one horizontal slice.

Definition at line 39 of file FilteredBackProjection.cpp.

```
40 {
41     const double range = array.GetRange(); // This is the same as 2D since we are considering slice by
       slice.
42     const double rangeZ = array.GetRangeZ(); // This is the maximum domain height.
43     //std::cerr << "range is " << range <<std::endl;
44     //std::cerr << "rangeZ is " << rangeZ<<std::endl;
45     Bilinear bilin;
46     int Nslice = array.GetSlice(); // Nslice is the slice number of horizontal slice.
47     std::cerr << "FBP: Nslice is " << Nslice<<std::endl;
48     array.ConvolveWithKernal(kernal); // Filter each projection with kernal.
49     std::cerr << "FBP: Convolution done" << std::endl;
50
51     NumVolume* rec = new NumVolume(Nres,range,Nres,range,Nslice,rangeZ);
52     double*** w = rec->GetWPtr();
53     std::cerr << "run1" <<std::endl;
```

```
54      int Nangle = array.GetSize(); // Nangle is slice*size number of angle views
55      std::cerr <<"Nangle" << Nangle << std::endl;
56      int NviewPerslice = Nangle/Nslice; // NviewPerslice is the total number of angle view per slice.
        Assuming each slice has the same number of angle view for now.
57      std::cerr <<"Nviewperslice" << NviewPerslice << std::endl;
58      double sum[Nres][Nres];
59      for(int k=0;k<Nslice;k++){ //iterate over number of horizontal slice
60          for(int i=0;i<Nres;i++){
61              for(int j=0;j<Nres;j++){
62                  sum[i][j] =0;
63              }
64          }
65          for(int ll=0; ll<NviewPerslice; ll++){ // iterate over angle of view per slice
66              double angle = array.GetAngle(ll+k*NviewPerslice);
67              for( int i=0; i<Nres; i++){ // loop over x-coordinate
68                  double x = -range + i*2*range/(Nres-1);
69                  for( int j=0; j<Nres; j++){ // Loop over y-coordinate
70                      double y = -range + j*2*range/(Nres-1);
71                      double t = x*cos(angle) + y*sin(angle); // distance to origin for angle ll.
72                      NumCurve temp_Curve = (array.GetFilteredCurve(ll+k*NviewPerslice));
73                      sum[i][j] += (temp_Curve)(t,&bilin)*pi/NviewPerslice;
74                  } // i loop
75              }// j loop
76          }    // ll loop
77          for( int ii=0; ii<Nres; ii++){  // loop over x-coordinate
78              for( int jj=0; jj<Nres; jj++){  // loop over y-coordinate
79                  w[ii][jj][k] = sum[ii][jj]/Nslice;
80              }
81          }
82      }// k loop
83      return rec;
84  }
```

8.9 include/globals.h File Reference

Definitions for some constants.

`#include <stdlib.h>`

Macros

- #define pi 3.14159265357

Typedefs

- typedef double(∗ f1D) (double)

  1-D function pointer.
- typedef double(∗ f2D) (double, double)

  2-D function pointer.
- typedef double(∗ f3D) (double, double, double)

  3-D function pointer.

Enumerations

- enum Dimension { Dim0 =0, Dim1, Dim2, Dim3 }

Functions

- template<typename T >
  T max (T a, T b)
- template<typename T >
  T min (T a, T b)

- double Hamming (int, double)

  Default convolution kernal used by filtered back-projection.
- int ArryIndexFloor (double x, double *array, int size)

  Find the floor index number of the double in the given array.
- int ArryIndexRoof (double x, double *array, int size)

  Find the roof index number of the double in the given array.

8.9.1 Detailed Description

Definitions for some constants.

8.9.2 Macro Definition Documentation

8.9.2.1 #define pi 3.14159265357

Definition at line 7 of file globals.h.

8.9.3 Typedef Documentation

8.9.3.1 typedef double(* f1D) (double)

1-D function pointer.

Definition at line 12 of file globals.h.

8.9.3.2 typedef double(* f2D) (double, double)

2-D function pointer.

Definition at line 13 of file globals.h.

8.9.3.3 typedef double(* f3D) (double, double, double)

3-D function pointer.

Definition at line 14 of file globals.h.

8.9.4 Enumeration Type Documentation

8.9.4.1 enum Dimension

Enumerator

Dim0

Dim1

Dim2

Dim3

Definition at line 10 of file globals.h.

```
10 {Dim0=0, Dim1, Dim2, Dim3};
```

8.9.5 Function Documentation

8.9.5.1 int ArryIndexFloor (double *x*, double * *array*, int *size*)

Find the floor index number of the double in the given array.

Definition at line 13 of file globals.cpp.

```
14 {
15
16     int indx = int((size-1)*((x-array[0])/(array[size-1]-array[0])));
17     if(indx>=size-1){
18 //         fprintf(stderr,"Error: index %d is already at the end.\n",indx);
19         return -1;
20     }
21     else if(indx<0){
22 //         fprintf(stderr,"Error: index %d is below 0.\n",indx);
23         return -2;
24     }
25     if(x>=array[indx] && x<array[indx+1]){
26         return indx;
27     }
28     else{
29 //    the following two line for considering non-equally spaced arrays.
30 //         while(indx>0 && x>array[indx+1]) indx--;
31 //         while(indx<size-1 && x<array[indx]) indx++;
32         return -1;
33     }
34 }
```

8.9.5.2 int ArryIndexRoof (double *x*, double * *array*, int *size*)

Find the roof index number of the double in the given array.

Definition at line 36 of file globals.cpp.

```
37 {
38     int indx = ArryIndexFloor(x,array,size);
39     if(indx<size-1) return indx+1;
40     else return -1;
41 }
```

8.9.5.3 double Hamming (int , double)

Default convolution kernal used by filtered back-projection.

Definition at line 4 of file globals.cpp.

```
5 {
6     // Hamming window function, tau is real space spacing.
7     double t = tau*tau;
8     if (n==0) return 1.0/(4*t);
9     else if (abs(n)%2==0) return 0;
10     else return -1.0/(n*n*pi*pi*t);
11 }
```

8.9.5.4 template<typename T > T max (T *a*, T *b*)

Definition at line 17 of file globals.h.

```
17 {return a>b?a:b;}
```

8.9.5.5 template<typename T > T min (T *a*, T *b*)

Definition at line 20 of file globals.h.

```
20 {return a<b?a:b;}
```

8.10 include/Image.h File Reference

Abstract Image class from which curves, surfaces and volumes will be derived.

```
#include "globals.h"
#include "LineIntegral.h"
```

Classes

- class Image

8.10.1 Detailed Description

Abstract Image class from which curves, surfaces and volumes will be derived.

8.11 include/ImageArray.h File Reference

```
#include <vector>
#include "globals.h"
#include "NumCurve.h"
#include <stdio.h>
#include <iostream>
```

Classes

- class ImageArray

8.12 include/Interpolator.h File Reference

```
#include <vector>
```

Classes

- class Interpolator

8.13 include/LineIntegral.h File Reference

```
#include <math.h>
#include <functional>
```

Classes

- class LineIntegral

8.14 include/MCIntegrator.h File Reference

```
#include "MCIntegrator.h"
#include "LineIntegral.h"
#include <functional>
```

Classes

- class MCIntegrator

8.15 include/NearestNeighborIntpl.h File Reference

```
#include "Interpolator.h"
#include <vector>
```

Classes

- class NearestNeighborIntpl

8.16 include/NumCurve.h File Reference

```
#include "Curve.h"
#include "Interpolator.h"
```

Classes

- class NumCurve

8.17 include/NumSurface.h File Reference

This file defines numerical images whose data points are defined by discrete points.

```
#include "Interpolator.h"
#include "Surface.h"
```

Classes

- class NumSurface

8.17.1 Detailed Description

This file defines numerical images whose data points are defined by discrete points.

8.18 include/NumVolume.h File Reference

```
#include "Interpolator.h"
#include "Volume.h"
```

Classes

- class NumVolume

  This file defines numerical images whose data points are defined by discrete points.

8.19 include/Parabola.h File Reference

```
#include "LineIntegral.h"
#include <functional>
```

Classes

- class Parabola

8.20 include/Romberg.h File Reference

```
#include "LineIntegral.h"
#include <functional>
```

Classes

- class Romberg

8.21 include/Surface.h File Reference

Abstract Image class for Two-dimensional CT images called surface.

```
#include "LineIntegral.h"
#include "Image.h"
#include "Interpolator.h"
#include "NumCurve.h"
```

Classes

- class Surface

  Forward declaration.

8.21.1 Detailed Description

Abstract Image class for Two-dimensional CT images called surface.

8.22 include/TestFunctions.h File Reference

Contains frequently used functions and features.

```
#include <math.h>
#include <stdio.h>
#include "NumCurve.h"
```

Functions

- double Gauss1D (double x)

  1D Gauss function
- double Gauss2D (double x, double y)

  2D Gauss
- double Gauss3D (double x, double y, double z)

  3D Gauss
- double Sphere (double x, double y, double z)
- double Heart (double x, double y, double z)

  Heart function.
- double Circle (double x, double y)

  Circle. Returns 1 or 0.
- double Rectangle (double x, double y)

  Rectangle Returns 1 or 0.
- double Cube (double x, double y, double z)

  Rectangle Returns 1 or 0.
- double Triangle (double x, double y)

  Triangle. Returns 1 or 0.
- bool assertArrayEqual (double *, double *, int, double precision=1e-8)

  Check if two arrays agree within the precision.
- bool assertEqual (NumCurve, NumCurve, double precision=1e-8)

  Check if two NumCurve agrees with each other.
- bool assertEqual (NumCurve *, NumCurve *, double precision=1e-8)

  Check if two NumCurves pointed by the pointer agrees with each other.
- double Batman (double, double)

  If a point lies in a batman symbol return 1, else return 0.
- double Heaviside (double x)

8.22.1 Detailed Description

Contains frequently used functions and features.

8.22.2 Function Documentation

8.22.2.1 bool assertArrayEqual (double *, double *, int, double *precision* = 1e-8)

Check if two arrays agree within the precision.

Definition at line 44 of file TestFunctions.cpp.

```
44                                                                          {
45
46     for(int i=0;i<n;i++){
47         double diff=fabs(x[i]-y[i]);
48         if(diff>precision){
49             fprintf(stderr,"Test failed: array didn't match at index %d by %.5f\n",i,diff);
50             return false;
51         }
52     }
53     return true;
54 }
```

8.22.2.2 bool assertEqual (NumCurve , NumCurve , double *precision* = 1e-8)

Check if two NumCurve agrees with each other.

Definition at line 56 of file TestFunctions.cpp.

```
56                                                          {
57
58     if(a.GetXPtr()==b.GetXPtr()){
59         fprintf(stderr,"Test warning: the two have the same X pointer address.\n");
60     }
61     if(a.GetYPtr()==b.GetYPtr()){
62         fprintf(stderr,"Test warning: the two have the same Y pointer address.\n");
63     }
64
65     if(a.GetSize()!=b.GetSize()){
66         fprintf(stderr,"Test failed: different size.\n");
67         return false;
68     }
69     double size = a.GetSize();
70     double diff=fabs(a.GetRange()-b.GetRange());
71     if(diff>precision){
72         fprintf(stderr,"Test failed: range difference %.10f greater than precision %.10f\n",diff,precision)
       ;
73         return false;
74     }
75
76     if(!assertArrayEqual(a.GetXPtr(),b.GetXPtr(),size,precision)) return false;
77     if(!assertArrayEqual(a.GetYPtr(),b.GetYPtr(),size,precision)) return false;
78
79     return true;
80 }
```

8.22.2.3 bool assertEqual (NumCurve *, NumCurve *, double *precision* = 1e-8)

Check if two NumCurves pointed by the pointer agrees with each other.

Definition at line 82 of file TestFunctions.cpp.

```
82                                                          {
83     return assertEqual(*a,*b,precision);
84 }
```

8.22.2.4 double Batman (double , double)

If a point lies in a batman symbol return 1, else return 0.

Definition at line 86 of file TestFunctions.cpp.

```
87 {
88     if(y>0 and fabs(x)>3 and x*x/49+y*y/9-1<0)
89         return 1;
90     if(fabs(x)>4 and y < 0 and x*x/49+y*y/9-1<0)
91         return 1;
92     if (y<0 and fabs(x/2)-(3*sqrt(33)-7)/112.*x*x-3+sqrt(1-(fabs(fabs(x)-2)-1)*(fabs(fabs(
       x)-2)-1))-y <0 and fabs(x)<4)
93         return 1;
94     if (fabs(x)>0.75 and fabs(x)<1 and y < 9-8*fabs(x) and y>0)
95         return 1;
```

```
96     if (fabs(x)>0.5 and fabs(x)<0.75 and y>0 and y<3*fabs(x)+0.75)
97         return 1;
98     if (y<2.25 and y>0 and fabs(x)<0.5)
99         return 1;
100     if (y>0 and fabs(x)>1 and fabs(x)<3 and 6.*sqrt(10)/7 + (1.5-0.5*fabs(x))-6.0*sqrt(10)/14.*sqrt(4-(
      fabs(x)-1)*(fabs(x)-1)) -y > 0  )
101         return 1;
102     else return 0;
103 }
```

8.22.2.5 double Circle (double *x*, double *y*)

Circle. Returns 1 or 0.

Definition at line 19 of file TestFunctions.cpp.

```
19                                  {
20     return (x*x+y*y > 16)?0:1;
21 }
```

8.22.2.6 double Cube (double *x*, double *y*, double *z*)

Rectangle Returns 1 or 0.

Definition at line 32 of file TestFunctions.cpp.

```
32                                              {
33     double val = 0;
34     if(fabs(x)<1 && fabs(y)<1 && fabs(z)<1) val = 1;
35     return val;
36 }
```

8.22.2.7 double Gauss1D (double *x*)

1D Gauss function

Definition at line 3 of file TestFunctions.cpp.

```
3                           {
4     return exp(-x*x);
5 }
```

8.22.2.8 double Gauss2D (double *x*, double *y*)

2D Gauss

Definition at line 7 of file TestFunctions.cpp.

```
7                                       {
8     return exp(-x*x/3-y*y/5)*sin(x);
9 }
```

8.22.2.9 double Gauss3D (double *x*, double *y*, double *z*)

3D Gauss

Definition at line 11 of file TestFunctions.cpp.

```
11                                                  {
12     return exp(-x*x/6 -y*y/3 + x*y/4 - z*z/3 + z*y/4);
13 }
```

8.22.2.10 double Heart (double *x*, double *y*, double *z*)

Heart function.

Definition at line 15 of file TestFunctions.cpp.

```
15                                              {
16     return (pow((x*x+9*y*y/4+z*z-1),3)-x*x*z*z*z-9*y*y*z*z*z/80 >0)?0:1;
17 }
```

8.22.2.11 double Heaviside (double *x*)

Definition at line 105 of file TestFunctions.cpp.

```
105                             {
106     return x>0?1.0:0.0;
107 }
```

8.22.2.12 double Rectangle (double *x*, double *y*)

Rectangle Returns 1 or 0.

Definition at line 27 of file TestFunctions.cpp.

```
27                                      {
28     double val = 0;
29     if(fabs(x)<3 && fabs(y)<3) val = 1;
30     return val;
31 }
```

8.22.2.13 double Sphere (double *x*, double *y*, double *z*)

Definition at line 23 of file TestFunctions.cpp.

```
23                                              {
24     return (x*x+y*y+z*z> 1)?0:1;
25 }
```

8.22.2.14 double Triangle (double *x*, double *y*)

Triangle. Returns 1 or 0.

Definition at line 37 of file TestFunctions.cpp.

```
37                                      {
38     if(y<-1) return 0;
39     if(2*x+y>1) return 0;
40     if(2*x-y<-1) return 0;
41     return sin(x);
42 }
```

8.23 include/Trapezoid.h File Reference

```
#include "LineIntegral.h"
#include <functional>
```

Classes

- class Trapezoid

8.24 include/Volume.h File Reference

Abstract class for 3-dimensional image called Volume.

```
#include "Image.h"
#include "Interpolator.h"
#include "NumSurface.h"
#include "NumCurve.h"
#include "LineIntegral.h"
```

Classes

- class Volume

8.24.1 Detailed Description

Abstract class for 3-dimensional image called Volume.

8.25 output/movieVolume.py File Reference

Namespaces

- movieVolume

Functions

- def movieVolume.animate

Variables

- list movieVolume.fname = sys.argv[1]
- tuple movieVolume.infile = h5py.File(fname, 'r')
- tuple movieVolume.x = np.array((infile["x"]))
- tuple movieVolume.y = np.array((infile["y"]))
- tuple movieVolume.z = np.array((infile["z"]))
- tuple movieVolume.data = np.array((infile["data"]))
- tuple movieVolume.dmin = data.min()
- tuple movieVolume.dmax = data.max()
- tuple movieVolume.fig = plt.figure()
- tuple movieVolume.ani = manimation.FuncAnimation(fig, animate, frames = z.shape[0], interval = 50)

8.26 output/plotCurve.py File Reference

Namespaces

- plotCurve

Variables

- list plotCurve.fname = sys.argv[1]
- tuple plotCurve.infile = h5py.File(fname + ".h5", 'r')
- list plotCurve.x = infile["x"]
- list plotCurve.data = infile["data"]

8.27 output/plotSurface.py File Reference

Namespaces

- plotSurface

Variables

- list plotSurface.fname = sys.argv[1]
- tuple plotSurface.infile = h5py.File(fname, 'r')
- tuple plotSurface.x = np.array((infile["x"]))
- tuple plotSurface.y = np.array((infile["y"]))
- tuple plotSurface.data = np.array((infile["data"]))

8.28 README.md File Reference

8.29 src/AnaImage.cpp File Reference

```
#include "AnaImage.h"
#include <iostream>
#include <stdio.h>
```

8.30 src/Bilinear.cpp File Reference

```
#include "Bilinear.h"
#include "globals.h"
```

8.31 src/Curve.cpp File Reference

Source code for 1D function (Curve).

```
#include "Curve.h"
#include <stdio.h>
#include <string.h>
```

8.31.1 Detailed Description

Source code for 1D function (Curve).

8.32 src/FilteredBackProjection.cpp File Reference

```
#include "FilteredBackProjection.h"
#include "Bilinear.h"
#include "globals.h"
#include <math.h>
#include <stdio.h>
#include <iostream>
```

Functions

- NumSurface * FilteredBackProjection (ImageArray &array, int Nres, double(*kernal)(int, double))

  From input ImageArray, performs filtered back-projection to reconstruct. Nres is the resolution of the final image / number of point along one axis. kernal is used to convolve the projection image. By default it is the Hamming function.

- NumVolume * FilteredBackProjection3D (ImageArray &array, int Nres, double(*kernal)(int, double))

  From input ImageArray, performs filtered back-projection in 3D to reconstruct. Nres is the resolution of the final image / number of point along one axis. kernal is used to convolve the projection image. By default it is the Hamming function. ImageArray should contain more than one horizontal slice.

8.32.1 Function Documentation

8.32.1.1 NumSurface* FilteredBackProjection (ImageArray & *array*, int *Nres*, double(*)(int, double) *kernal*)

From input ImageArray, performs filtered back-projection to reconstruct. Nres is the resolution of the final image / number of point along one axis. kernal is used to convolve the projection image. By default it is the Hamming function.

Nres is the resolution for the reconstructed surface.

Definition at line 8 of file FilteredBackProjection.cpp.

```
9 {
11     const double range = array.GetRange();
12     Bilinear bilin;
13     array.ConvolveWithKernal(kernal); // Array filters each curve with the kernal.
14     std::cerr<<"FBP: Convolution done."<<std::endl;
15     NumSurface* rec = new NumSurface(Nres,range,Nres,range);    // NumSurface to store
       the reconstructed obj.
16     int Nangle = array.GetSize();
17     std::cerr<<"Size is."<< Nangle <<std::endl;
18     for(int ll=0; ll<Nangle; ll++){ // Iterate over all angle of view
19         double angle = array.GetAngle(ll);
20         std::cerr<<"Angle is."<< angle <<std::endl;
21         for( int i=0; i<Nres; i++){ // Loop over X-coordinates of final surface.
22             double x = -range + i*2*range/(Nres-1);
23             for( int j=0; j<Nres; j++){ // Loop over y-coordinate of final surface.
24                 double y = -range + j*2*range/(Nres-1);
25                 double t = x*cos(angle) + y*sin(angle); // Distance from (x,y) to origin at angle ll.
26                 //std::cerr<<"t is."<< t <<" "<<i<<" "<<j<< "range is "<< range<<std::endl;
27                 (*rec)(i,j) += (array.GetFilteredCurve(ll))(t,&bilin)*
       pi/Nangle; // Superpose all values, assuming uniform grid.
28             }
29         }
30         std::cerr<<"running" <<std::endl;
31 //        #ifdef USE_HDF
32 //        char file[100]; sprintf(file,"output/batman_rec%d.h5",ll);
33 //        rec->ExportHDF(file);
34 //        #endif
35     }
36     return rec;
37 }
```

8.32.1.2 NumVolume* FilteredBackProjection3D (ImageArray & *array*, int *Nres*, double(*)(int, double) *kernal*)

From input ImageArray, performs filtered back-projection in 3D to reconstruct. Nres is the resolution of the final image / number of point along one axis. kernal is used to convolve the projection image. By default it is the Hamming function. ImageArray should contain more than one horizontal slice.

Definition at line 39 of file FilteredBackProjection.cpp.

```
40 {
41     const double range = array.GetRange(); // This is the same as 2D since we are considering slice
       by slice.
42     const double rangeZ = array.GetRangeZ(); // This is the maximum domain height.
43     //std::cerr << "range is " << range <<std::endl;
44     //std::cerr << "rangeZ is " << rangeZ<<std::endl;
45     Bilinear bilin;
46     int Nslice = array.GetSlice(); // Nslice is the slice number of horizontal slice.
47     std::cerr << "FBP: Nslice is " << Nslice<<std::endl;
48     array.ConvolveWithKernal(kernal); // Filter each projection with kernal.
49     std::cerr << "FBP: Convolution done" << std::endl;
50 
51     NumVolume* rec = new NumVolume(Nres,range,Nres,range,Nslice,rangeZ);
52     double*** w = rec->GetWPtr();
53     std::cerr << "run1" <<std::endl;
54     int Nangle = array.GetSize(); // Nangle is slice*size number of angle views
55     std::cerr <<"Nangle" << Nangle << std::endl;
56     int NviewPerslice = Nangle/Nslice; // NviewPerslice is the total number of angle view per slice.
       Assuming each slice has the same number of angle view for now.
57     std::cerr <<"Nviewperslice" << NviewPerslice << std::endl;
58     double sum[Nres][Nres];
59     for(int k=0;k<Nslice;k++){ //iterate over number of horizontal slice
60         for(int i=0;i<Nres;i++){
61             for(int j=0;j<Nres;j++){
62                 sum[i][j] =0;
63             }
64         }
65         for(int ll=0; ll<NviewPerslice; ll++){ // iterate over angle of view per slice
66             double angle = array.GetAngle(ll+k*NviewPerslice);
67             for( int i=0; i<Nres; i++){ // loop over x-coordinate
68                 double x = -range + i*2*range/(Nres-1);
69                 for( int j=0; j<Nres; j++){ // Loop over y-coordinate
70                     double y = -range + j*2*range/(Nres-1);
71                     double t = x*cos(angle) + y*sin(angle); // distance to origin for angle ll.
72                     NumCurve temp_Curve = (array.GetFilteredCurve(ll+k*
       NviewPerslice));
73                     sum[i][j] += (temp_Curve)(t,&bilin)*pi/NviewPerslice;
74                 } // i loop
75             }// j loop
76         }    // ll loop
77         for( int ii=0; ii<Nres; ii++){  // loop over x-coordinate
78             for( int jj=0; jj<Nres; jj++){  // loop over y-coordinate
79                 w[ii][jj][k] = sum[ii][jj]/Nslice;
80             }
81         }
82     }// k loop
83     return rec;
84 }
```

8.33 src/globals.cpp File Reference

```
#include "globals.h"
#include <stdio.h>
```

Functions

- double Hamming (int n, double tau)

  Default convolution kernal used by filtered back-projection.
- int ArryIndexFloor (double x, double *array, int size)

  Find the floor index number of the double in the given array.
- int ArryIndexRoof (double x, double *array, int size)

  Find the roof index number of the double in the given array.

8.33.1 Function Documentation

8.33.1.1 int ArryIndexFloor (double *x*, double * *array*, int *size*)

Find the floor index number of the double in the given array.

Definition at line 13 of file globals.cpp.

```
14 {
15
16     int indx = int((size-1)*((x-array[0])/(array[size-1]-array[0])));
17     if(indx>=size-1){
18 //         fprintf(stderr,"Error: index %d is already at the end.\n",indx);
19         return -1;
20     }
21     else if(indx<0){
22 //         fprintf(stderr,"Error: index %d is below 0.\n",indx);
23         return -2;
24     }
25     if(x>=array[indx] && x<array[indx+1]){
26         return indx;
27     }
28     else{
29 //    the following two line for considering non-equally spaced arrays.
30 //         while(indx>0 && x>array[indx+1]) indx--;
31 //         while(indx<size-1 && x<array[indx]) indx++;
32         return -1;
33     }
34 }
```

8.33.1.2 int ArryIndexRoof (double *x*, double * *array*, int *size*)

Find the roof index number of the double in the given array.

Definition at line 36 of file globals.cpp.

```
37 {
38     int indx = ArryIndexFloor(x,array,size);
39     if(indx<size-1) return indx+1;
40     else return -1;
41 }
```

8.33.1.3 double Hamming (int *n*, double *tau*)

Default convolution kernal used by filtered back-projection.

Definition at line 4 of file globals.cpp.

```
5 {
6     // Hamming window function, tau is real space spacing.
7     double t = tau*tau;
8     if (n==0) return 1.0/(4*t);
9     else if (abs(n)%2==0) return 0;
10     else return -1.0/(n*n*pi*pi*t);
11 }
```

8.34 src/Image.cpp File Reference

```
#include "Image.h"
```

8.35 src/ImageArray.cpp File Reference

```
#include "ImageArray.h"
```

```
#include "globals.h"
```

8.36 src/Interpolator.cpp File Reference

```
#include "Interpolator.h"
```

8.37 src/LineIntegral.cpp File Reference

```
#include "LineIntegral.h"
```

8.38 src/MCIntegrator.cpp File Reference

```
#include "MCIntegrator.h"
#include <stdlib.h>
```

8.39 src/NearestNeighborIntpl.cpp File Reference

```
#include "NearestNeighborIntpl.h"
#include <math.h>
#include <vector>
```

8.40 src/NumCurve.cpp File Reference

```
#include "NumCurve.h"
#include <stdio.h>
#include <string.h>
```

8.41 src/NumSurface.cpp File Reference

Implementation for numerical surfaces.

```
#include "NumSurface.h"
#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <string.h>
```

8.41.1 Detailed Description

Implementation for numerical surfaces.

8.42 src/NumVolume.cpp File Reference

```
#include "NumVolume.h"
#include <stdio.h>
#include <iostream>
#include <string.h>
```

8.43 src/Parabola.cpp File Reference

```
#include "Parabola.h"
```

8.44 src/Romberg.cpp File Reference

```
#include "Romberg.h"
#include "math.h"
```

8.45 src/Surface.cpp File Reference

```
#include "Surface.h"
#include <stdio.h>
#include <math.h>
#include <string.h>
```

8.46 src/TestFunctions.cpp File Reference

```
#include "TestFunctions.h"
```

Functions

- double Gauss1D (double x)

  1D Gauss function
- double Gauss2D (double x, double y)

  2D Gauss
- double Gauss3D (double x, double y, double z)

  3D Gauss
- double Heart (double x, double y, double z)

  Heart function.
- double Circle (double x, double y)

  Circle. Returns 1 or 0.
- double Sphere (double x, double y, double z)
- double Rectangle (double x, double y)

  Rectangle Returns 1 or 0.
- double Cube (double x, double y, double z)

Rectangle Returns 1 or 0.

- double Triangle (double x, double y)

  Triangle. Returns 1 or 0.
- bool assertArrayEqual (double *x, double *y, int n, double precision)

  Check if two arrays agree within the precision.
- bool assertEqual (NumCurve a, NumCurve b, double precision)

  Check if two NumCurve agrees with each other.
- bool assertEqual (NumCurve *a, NumCurve *b, double precision)

  Check if two NumCurves pointed by the pointer agrees with each other.
- double Batman (double x, double y)

  If a point lies in a batman symbol return 1, else return 0.
- double Heaviside (double x)

8.46.1 Function Documentation

8.46.1.1 bool assertArrayEqual (double * *x*, double * *y*, int *n*, double *precision*)

Check if two arrays agree within the precision.

Definition at line 44 of file TestFunctions.cpp.

```
44                                                                          {
45
46     for(int i=0;i<n;i++){
47         double diff=fabs(x[i]-y[i]);
48         if(diff>precision){
49             fprintf(stderr,"Test failed: array didn't match at index %d by %.5f\n",i,diff);
50             return false;
51         }
52     }
53     return true;
54 }
```

8.46.1.2 bool assertEqual (NumCurve *a*, NumCurve *b*, double *precision*)

Check if two NumCurve agrees with each other.

Definition at line 56 of file TestFunctions.cpp.

```
56                                                          {
57
58     if(a.GetXPtr()==b.GetXPtr()){
59         fprintf(stderr,"Test warning: the two have the same X pointer address.\n");
60     }
61     if(a.GetYPtr()==b.GetYPtr()){
62         fprintf(stderr,"Test warning: the two have the same Y pointer address.\n");
63     }
64
65     if(a.GetSize()!=b.GetSize()){
66         fprintf(stderr,"Test failed: different size.\n");
67         return false;
68     }
69     double size = a.GetSize();
70     double diff=fabs(a.GetRange()-b.GetRange());
71     if(diff>precision){
72         fprintf(stderr,"Test failed: range difference %.10f greater than precision %.10f\n",diff,precision)
    ;
73         return false;
74     }
75
76     if(!assertArrayEqual(a.GetXPtr(),b.GetXPtr(),size,precision)) return
    false;
77     if(!assertArrayEqual(a.GetYPtr(),b.GetYPtr(),size,precision)) return
    false;
78
79     return true;
80 }
```

8.46.1.3 bool assertEqual (NumCurve * *a*, NumCurve * *b*, double *precision*)

Check if two NumCurves pointed by the pointer agrees with each other.

Definition at line 82 of file TestFunctions.cpp.

```
82                                                          {
83     return assertEqual(*a,*b,precision);
84 }
```

8.46.1.4 double Batman (double *x*, double *y*)

If a point lies in a batman symbol return 1, else return 0.

Definition at line 86 of file TestFunctions.cpp.

```
87 {
88     if(y>0 and fabs(x)>3 and x*x/49+y*y/9-1<0)
89         return 1;
90     if(fabs(x)>4 and y < 0 and x*x/49+y*y/9-1<0)
91         return 1;
92     if (y<0 and fabs(x/2)-(3*sqrt(33)-7)/112.*x*x-3+sqrt(1-(fabs(fabs(x)-2)-1)*(fabs(fabs(
      x)-2)-1))-y <0 and fabs(x)<4)
93         return 1;
94     if (fabs(x)>0.75 and fabs(x)<1 and y < 9-8*fabs(x) and y>0)
95         return 1;
96     if (fabs(x)>0.5 and fabs(x)<0.75 and y>0 and y<3*fabs(x)+0.75)
97         return 1;
98     if (y<2.25 and y>0 and fabs(x)<0.5)
99         return 1;
100     if (y>0 and fabs(x)>1 and fabs(x)<3 and 6.*sqrt(10)/7 + (1.5-0.5*fabs(x))-6.0*sqrt(10)/14.*sqrt(4-(
      fabs(x)-1)*(fabs(x)-1)) -y > 0  )
101         return 1;
102     else return 0;
103 }
```

8.46.1.5 double Circle (double *x*, double *y*)

Circle. Returns 1 or 0.

Definition at line 19 of file TestFunctions.cpp.

```
19                                  {
20     return (x*x+y*y > 16)?0:1;
21 }
```

8.46.1.6 double Cube (double *x*, double *y*, double *z*)

Rectangle Returns 1 or 0.

Definition at line 32 of file TestFunctions.cpp.

```
32                                          {
33     double val = 0;
34     if(fabs(x)<1 && fabs(y)<1 && fabs(z)<1) val = 1;
35     return val;
36 }
```

8.46.1.7 double Gauss1D (double *x*)

1D Gauss function

Definition at line 3 of file TestFunctions.cpp.

```
3                           {
4     return exp(-x*x);
5 }
```

8.46.1.8 double Gauss2D (double *x*, double *y*)

2D Gauss

Definition at line 7 of file TestFunctions.cpp.

```
7                                   {
8     return exp(-x*x/3-y*y/5)*sin(x);
9 }
```

8.46.1.9 double Gauss3D (double *x*, double *y*, double *z*)

3D Gauss

Definition at line 11 of file TestFunctions.cpp.

```
11                                                  {
12     return exp(-x*x/6 -y*y/3 + x*y/4 - z*z/3 + z*y/4);
13 }
```

8.46.1.10 double Heart (double *x*, double *y*, double *z*)

Heart function.

Definition at line 15 of file TestFunctions.cpp.

```
15                                                  {
16     return (pow((x*x+9*y*y/4+z*z-1),3)-x*x*z*z*z-9*y*y*z*z*z/80 >0)?0:1;
17 }
```

8.46.1.11 double Heaviside (double *x*)

Definition at line 105 of file TestFunctions.cpp.

```
105                             {
106     return x>0?1.0:0.0;
107 }
```

8.46.1.12 double Rectangle (double *x*, double *y*)

Rectangle Returns 1 or 0.

Definition at line 27 of file TestFunctions.cpp.

```
27                                      {
28     double val = 0;
29     if(fabs(x)<3 && fabs(y)<3) val = 1;
30     return val;
31 }
```

8.46.1.13 double Sphere (double *x*, double *y*, double *z*)

Definition at line 23 of file TestFunctions.cpp.

```
23                                                  {
24     return (x*x+y*y+z*z> 1)?0:1;
25 }
```

8.46.1.14 double Triangle (double *x*, double *y*)

Triangle. Returns 1 or 0.

Definition at line 37 of file TestFunctions.cpp.

```
37                                      {
38     if(y<-1) return 0;
39     if(2*x+y>1) return 0;
40     if(2*x-y<-1) return 0;
41     return sin(x);
42 }
```

8.47 src/Trapezoid.cpp File Reference

```
#include "Trapezoid.h"
```

8.48 src/Volume.cpp File Reference

```
#include "Volume.h"
#include <math.h>
#include <stdio.h>
#include <iostream>
#include <string.h>
```

8.49 test/test_Intpl.cpp File Reference

```
#include "Image.h"
#include "AnaImage.h"
#include "NumCurve.h"
#include "NumSurface.h"
#include "Surface.h"
#include "Trapezoid.h"
#include "Interpolator.h"
#include "NearestNeighborIntpl.h"
#include "MCIntegrator.h"
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
```

Functions

- double gauss_1D (double x)
- double gauss_2D (double x, double y)
- double gauss_3D (double x, double y, double z)
- double cylinder (double x, double y)
- double box (double x, double y)
- int main (int argc, char ∗argv[ ])

8.49.1 Function Documentation

8.49.1.1 double box (double *x*, double *y*)

Definition at line 31 of file test_Intpl.cpp.

```
31                                {
32     double val = 0;
33     if(fabs(x)<2 && fabs(y)<2) val = 1;
34     return val;
35 }
```

8.49.1.2 double cylinder (double *x*, double *y*)

Definition at line 27 of file test_Intpl.cpp.

```
27                                      {
28     return (x*x+y*y > 4)?0:1;
29 }
```

8.49.1.3 double gauss_1D (double *x*)

Definition at line 14 of file test_Intpl.cpp.

```
14                            {
15     return exp(-x*x/6);
16 }
```

8.49.1.4 double gauss_2D (double *x*, double *y*)

Definition at line 18 of file test_Intpl.cpp.

```
18                                      {
19 //    return exp(-x*x/6 -y*y/3 + x*y/4)+ x*y/100;
20     return exp(-x*x/3-y*y/15);
21 }
```

8.49.1.5 double gauss_3D (double *x*, double *y*, double *z*)

Definition at line 23 of file test_Intpl.cpp.

```
23                                                 {
24     return exp(-x*x/6 -y*y/3 + x*y/4 - z*z/3 + z*y/4);
25 }
```

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

Definition at line 37 of file test_Intpl.cpp.

```
37                                  {
38
39 //  creates a nonsymmetric gaussian on 20 x 20 region.
40     Image* gauss  = new AnaSurface( gauss_2D, 10, 10);
41
42 // creates a numerical line using gauss_1D.
43       //Curve* gauss1 = new AnaCurve( gauss_1D,10);
44       //Image* num_gauss1 = new NumCurve(100,*gauss1);
45
```

```
46 // creates a numerical surface using gass_2D.
47     Surface* gauss2 = new AnaSurface( gauss_2D,2,2);
48     NumSurface* num_gauss2 = new NumSurface(5,10,*gauss2);
49 //    NumSurface Baltimore=NumSurface("./output/BaltimoreDowntown.h5");
50 // defines trapezoid integration rule.
51     LineIntegral* l=new Trapezoid();
52
53 //  since Projection is defined
54    switch(argc)
55     {
56        case 1 : {num_gauss2->Print(); delete gauss2; delete num_gauss2;return 0;}
57        //case 1 : {num_gauss1->Print(); delete gauss1; delete num_gauss1;return 0;}
58        case 2 : {
59            //NumSurface a = *num_gauss2;
60            //a.SetRange(2,2);
61            //a.Print();
62 //      NumCurve* ptr = (NumCurve*)num_gauss1; //Image has no GetProjection, must downcast.
63 //          NumCurve a = *ptr;  //double a;
64          //printf("%.9f\n",a(-1.8,-1.8,0));
65 //         printf("%.9f\n",a(0,0));
66 //           double r = Baltimore.GetRange();
67           // printf("r is %f\n",r);
68 //         for(double rx=-r;rx<r;rx+=0.005){
69 //                for(double ry=-r;ry<r;ry+=0.005){
70            //printf("%.9f\n",a(1.8,1.8,intpl));
71            //printf("%.9f\n",a(0.8,0.8,intpl));
72            //printf("%.9f\n",a(0.1,0.1,intpl));
73         //         Interpolator* intpl=new NearestNeighborIntpl();
74 //                printf("%.9f\n",Baltimore(rx,ry,0));
75 //               }
76 //           }
77
78         delete num_gauss2; return 0;
79          }
80         default: {delete gauss; break;}
81     }
82
83     return 0;
84 }
```

8.50 test/testIntegration.cpp File Reference

```
#include "MCIntegrator.h"
#include "Parabola.h"
#include "Romberg.h"
#include "Trapezoid.h"
#include <stdio.h>
#include <stdlib.h>
```

Functions

- double f (double x)
- int main (int argc, char ∗argv[ ])

8.50.1 Function Documentation

8.50.1.1 double f (double *x*)

Definition at line 8 of file testIntegration.cpp.

```
8                   {
9     return 3*x*x;
10 }
```

8.50.1.2 int main (int *argc*, char ∗ *argv[ ]*)

Definition at line 12 of file testIntegration.cpp.

```
12                                {
13   /*
14     int choice = -1;
15     if (argc > 1) choice = atoi(argv[1]);
16
17     LineIntegral* l;
18     double step;
19
20     if(choice == -1){
21         printf("USAGE: %s (number)\n number is 0~3\n 0: MCIntegrator, 1: Parabola, 2: Romberg, 3:
       Trapezoid\n",argv[0]);
22         return -1;
23     }
24     if(choice == 0){ MCIntegrator t; l = &t; step = 0.001;}
25     if(choice == 1){ Parabola t; l = &t; step = 0.001;}
26     if(choice == 2){ Romberg t; l = &t; step = 0.00001;}
27     if(choice == 3){ Trapezoid t; l = &t; step = 0.001;}
28
29     printf("The result of integration is %f\n",l->Integrate(f, 0, 3, step));
30   */
31     LineIntegral* l1;
32     LineIntegral* l2;
33     LineIntegral* l3;
34     LineIntegral* l4;
35     MCIntegrator t1;
36     Parabola t2;
37     Romberg t3;
38     Trapezoid t4;
39     double step1 = 0.001;
40     double step2 = 0.00001;
41     l1 = &t1;
42     l2 = &t2;
43     l3 = &t3;
44     l4 = &t4;
45     printf("The result of MCIntegraor Method is %f\n",l1->Integrate(f, 0, 3, step1));
46     printf("The result of Parabola Method is %f\n",l2->Integrate(f, 0, 3, step1));
47     printf("The result of Romberg Method is %f\n",l3->Integrate(f, 0, 3, step2));
48     printf("The result of Trapezoid Method is %f\n",l4->Integrate(f, 0, 3, step1));
49
50 }
```

8.51 test/testInterpolation.cpp File Reference

```
#include "globals.h"
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <iostream>
```

Functions

- int main ()

8.51.1 Function Documentation

8.51.1.1 int main ( )

Definition at line 8 of file testInterpolation.cpp.

```
8           {
9     srand(time(NULL));
10 //    First test array index searching method.
11     int N = 200000;
12     int size = N;
13     double begin = -100;
14     double end = 100;
15     double* array = new double[N];
16     for(int i=0;i<N;i++)
17         array[i] = begin + i*(end-begin)/(N-1);
18
19     for(int i=0;i<1000000;i++){
```

```
20          double rnd = begin + (end-begin)*rand()/RAND_MAX;
21          int index = ArryIndexFloor(rnd,array,N);
22          if(index<0) return -1;
23          if(index != size-1)
24            if(rnd<array[index] || rnd>array[index+1]){
25              fprintf(stderr,"Test Failed at run %d: %.5f index %d is not in %.5f %.5f\n",i,rnd,index,array[
    index],array[index+1]);
26          }
27      }
28      delete [] array;
29      fprintf(stderr,"Test Passed.\n");
30      return 0;
31  }
```

8.52 test/testNumCurve.cpp File Reference

```
#include "AnaImage.h"
#include "NumCurve.h"
#include "Trapezoid.h"
#include "TestFunctions.h"
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
```

Functions

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

8.52.1 Function Documentation

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

Definition at line 9 of file testNumCurve.cpp.

```
9                                   {
10      int choice = -1;
11      if(argc>1) choice = atoi(argv[1]);
12
13  // default ctor()
14      NumCurve a;
15      NumCurve* aptr = &a;
16      if(!assertEqual(a,*aptr)) return -1;
17
18  // ctor with a size;
19      NumCurve b(100);
20      NumCurve* bptr = &b;
21
22      double range = 20;
23      const int N = 500;
24      double datax[N] = {0};
25      double datay[N] = {0};
26      for(int i=0;i<N;i++){
27          datax[i] = -range + i*2.0*range/(N-1);
28          datay[i] = exp(-fabs(datax[i]/(0.5*range)))*sin(datax[i]);
29      }
30
31  // ctor with data points
32      NumCurve* c = new NumCurve(N,range,datay);
33      Curve* cptr = c;
34  // copy ctor
35      NumCurve d(*c);
36      if(!assertEqual(c,&d)) return -1;
37
38  // test assignment
39      if(choice==0)aptr->Print();
40      if(choice==1)bptr->Print();
41      if(choice==2)c->Print();
42      if(choice==3)d.Print();
43      if(choice==4) {
44          cptr->Print(-2*range,2*range,100000);
```

```
45 #ifdef USE_HDF
46         cptr->ExportHDF("curve.h5");
47 #endif
48     }
49 // test assignment
50     a = *c;
51     b = d;
52     if(!assertEqual(a,*c)) return -1;
53     if(!assertEqual(b,d)) return -1;
54 
55     delete c;
56     return 0;
57 }
```

8.53 test/testNumSurface.cpp File Reference

```
#include "AnaImage.h"
#include "NumCurve.h"
#include "NumSurface.h"
#include "Trapezoid.h"
#include "TestFunctions.h"
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
```

Functions

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

8.53.1 Function Documentation

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

Definition at line 10 of file testNumSurface.cpp.

```
10                                 {
11     int choice = -1;
12     if(argc>1) choice = atoi(argv[1]);
13 
14 // default ctor()
15     NumSurface a;
16     NumSurface* aptr = &a;
17 //    if(!assertEqual(a,*aptr)) return -1;
18 
19 // ctor with a size;
20     Surface *b_temp = new AnaSurface(Gauss2D,10,10);
21     NumSurface b(150,100,*b_temp);
22     NumSurface* bptr = &b;
23 
24     double range = 20;
25     const int N = 500;
26     double datax[N] = {0};
27     double datay[N] = {0};
28     double** dataz = new double*[N];
29     for(int i=0;i<N;i++){
30         datax[i] = -range + i*2.0*range/(N-1);
31         datay[i] = datax[i];
32         dataz[i] = new double[N];
33     }
34 
35     for(int i=0;i<N;i++)
36         for(int j=0;j<N;j++)
37             dataz[i][j] = exp(-(datax[i])*(datax[i])-(datay[j])*(datay[j]))*sin(datax[i]);
38 
39 // ctor with data points
40     NumSurface* c = new NumSurface(N,datax,N,datay,dataz);
41     Surface* cptr = c;
42 // copy ctor
43     NumSurface d(*c);
```

```
44 //    if(!assertEqual(c,&d)) return -1;
45
46 // test assignment
47     if(choice==0)aptr->Print();
48     if(choice==1){bptr->Print();
49 #ifdef USE_HDF
50         bptr->ExportHDF("test.h5");
51 #endif
52     }
53     if(choice==2)c->Print();
54     if(choice==3)d.Print();
55     if(choice==4)cptr->Print(-range,range,200,-range,range,200);
56 #ifdef USE_HDF
57     if(choice==5){
58         NumSurface* e = new NumSurface("output/BaltimoreDowntown.h5");
59         e->ExportHDF("BaltimoreDowntown2.h5");
60     delete e;
61     }
62 #endif
63
64
65 // test assignment
66     a = *c;
67     b = d;
68 //    if(!assertEqual(a,*c)) return -1;
69 //    if(!assertEqual(b,d)) return -1;
70
71     delete c;
72     return 0;
73 }
```

8.54 test/testVolume.cpp File Reference

```
#include "Image.h"
#include "AnaImage.h"
#include "NumCurve.h"
#include "NumSurface.h"
#include "NumVolume.h"
#include "Volume.h"
#include "Surface.h"
#include "Trapezoid.h"
#include "Romberg.h"
#include "MCIntegrator.h"
#include "NearestNeighborIntpl.h"
#include "TestFunctions.h"
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
```

Functions

- double gauss_1D (double x)
- double gauss_2D (double x, double y)
- double gauss_3D (double x, double y, double z)
- double cylinder (double x, double y)
- double box (double x, double y)
- int main (int argc, char *argv[])

8.54.1 Function Documentation

8.54.1.1 double box (double *x*, double *y*)

Definition at line 35 of file testVolume.cpp.

```
35                                {
36     double val = 0;
37     if(fabs(x)<2 && fabs(y)<2) val = 1;
38     return val;
39 }
```

8.54.1.2 double cylinder (double *x*, double *y*)

Definition at line 31 of file testVolume.cpp.

```
31                                      {
32     return (x*x+y*y > 4)?0:1;
33 }
```

8.54.1.3 double gauss_1D (double *x*)

Definition at line 18 of file testVolume.cpp.

```
18                             {
19     return exp(-x*x/6);
20 }
```

8.54.1.4 double gauss_2D (double *x*, double *y*)

Definition at line 22 of file testVolume.cpp.

```
22                                      {
23     //    return exp(-x*x/6 -y*y/3 + x*y/4)+ x*y/100;
24     return exp(-x*x/3-y*y/15);
25 }
```

8.54.1.5 double gauss_3D (double *x*, double *y*, double *z*)

Definition at line 27 of file testVolume.cpp.

```
27                                              {
28     return exp(-x*x/6 -y*y/3 + x*y/4 - z*z/3 + z*y/4);
29 }
```

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

Definition at line 41 of file testVolume.cpp.

```
41                                    {
42     if(argc<2){
43         printf("usage: ./testVolume angle\n");
44         return -1;
45     }
46     //  creates a nonsymmetric gaussian on 20 x 20 region.
47     Image* gauss  = new AnaVolume(Heart, 2, 1.5, 2);
48    // Interpolator* intpl = new NearestNeighborIntpl();
49     // creates a numerical line using gauss_1D.
50     //    Curve* gauss1 = new AnaCurve( gauss_1D,10);
51     //    Image* num_gauss1 = new NumCurve(100,*gauss1);
52
53     // creates a numerical surface using gass_3D.
54     //Volume* gauss3 = new AnaVolume( gauss_3D,2,2,1);
55     //NumVolume* num_gauss3 = new NumVolume(50,50,25,*gauss3);
56     // defines trapezoid integration rule.
57 /*    LineIntegral* l=new Romberg();
58
```

```
59     //  since Projection is defined
60
61     Volume* ptr = (Volume*)gauss; //Image has no GetProjection, must downcast.
62     NumSurface a;
63     ptr->SetIntegralStep(0.0001);
64     a = ptr->GetProjection(1,atof(argv[1]),0.01,0.01);
65 */     //a.Print();
66 #ifdef USE_HDF
67     gauss->ExportHDF("test.h5");
68 #endif
69     delete gauss;
70  //   delete gauss3;
71  //   delete num_gauss3;
72     return 0;
73 }
```

Index