Xavier Initialisation

1

Generated by Doxygen 1.10.0

Chapter 1

Introduction

Welcome to XIIA: Xavier Initialisation Image Application!

1.1 How To Use

To utilize this tool effectively, users can follow the steps outlined below to navigate through the interfaces for 2D and 3D image processing.

1.1.1 General Steps

1. **Start the Program**: Run the executable to start the program. This brings up the main menu.
2. **Choose between 2D and 3D**: At the main menu, select whether you wish to work with 2D or 3D images. Input `1` for 2D and `2` for 3D processing. To exit the program, enter `0`.

1.1.2 For 2D Image Processing

1. **Enter the Path of the Input Image**: After selecting 2D processing, you'll be prompted to enter the path to your image file.
2. **Choose a Filter**: You will see a list of available filters (e.g., Grayscale, Brightness, Histogram Equalisation, etc.). Enter the number corresponding to the filter you wish to apply.
3. **Configure the Filter (if required)**: Some filters might require additional input, such as brightness value or kernel size. Follow the prompts to input these values.
4. **Save Processed Image**: After applying a filter, you'll be prompted to enter a path to save the processed image.
5. **Continue or Exit**: Decide if you want to apply another filter to the same image or finish processing. To add another filter, input `y` when asked; to finish, input `n`.

1.1.3 For 3D Image Processing

1. **Enter the Path of the Input Volume**: After selecting 3D processing, you'll first be prompted to enter the path to your 3D volume data.
2. **Choose a Filter or Operation**: You will see a menu for selecting a 3D filter (e.g., Gaussian Filter, Median Filter) or operation (e.g., Projection, Slice). Select the desired option by entering its corresponding number.
3. **Configure the Filter or Operation (if required)**: Depending on your choice, you might need to specify additional parameters, such as filter size or sigma for Gaussian filtering, or select slices for projection.
4. **Save Processed Volume or Slices**: After applying a filter or operation, you'll be prompted to enter a path to save the output.
5. **Continue, Back, or Exit**: After each operation, you can choose to apply another filter or operation, go back to the previous menu, or exit the program. Select the appropriate option as per your needs.

1.1.4 Exiting the Program

- **Exit at Any Time**: You can exit the program at any time by selecting the `Exit` or `Back` option available in the menus, eventually leading you back to the main menu where you can exit the program by entering `0`.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all documented 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 documented files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 helpers Namespace Reference

Namespace containing utility functions.

Functions

- void print (const std::string &message)

  Print a message to standard output.
- std::string centered (const std::string &, int)

  Center-aligns a string within a given width.
- std::string formatPrice (double)

  Formats a double value as a price string.
- void **clearScreen** ()

  Clears the console screen.
- void **pause** ()

  Pauses program execution and waits for the user to press Enter.
- void printLine (int len=80)

  Prints a horizontal line of a specified length.
- void printTitle (const std::string &, int, Align, int)

  Prints a title with specified alignment and optional border lines.
- int getInput (int &)

  Gets integer input from the user between 0 and 100.
- void **printSigniture** ()

  Prints signature details for the program or script.
- void ensure_directory_exists (const fs::path &path, bool verbose=false)

  Ensures that a specified directory exists, creating it if necessary.
- std::vector< int > get_vector (int size)

  Generates and returns a vector filled with a sequence of integers.
- std::set< int > get_set (int size)

  Generates and returns a set filled with a sequence of integers.

6.1.1 Detailed Description

Namespace containing utility functions.

6.1.2 Function Documentation

6.1.2.1 centered()

```
std::string helpers::centered (
            const std::string & ,
            int  )
```

Center-aligns a string within a given width.

Parameters

str	String to center.
width	Width of the field within which to center the string.

Returns

A new string that is centered.

6.1.2.2 ensure_directory_exists()

```
void helpers::ensure_directory_exists (
            const fs::path & path,
            bool verbose = false )
```

Ensures that a specified directory exists, creating it if necessary.

Parameters

path	The filesystem path to the directory.
verbose	Specifies whether to print messages about the operation.

6.1.2.3 formatPrice()

```
std::string helpers::formatPrice (
            double  )
```

Formats a double value as a price string.

Parameters

price	The price to format.

Returns

A formatted price string.

6.1.2.4 get_set()

```
std::set< int > helpers::get_set (
            int size )
```

Generates and returns a set filled with a sequence of integers.

Parameters

size	The size of the set to generate.

Returns

A set of integers.

6.1.2.5 get_vector()

```
std::vector< int > helpers::get_vector (
            int size )
```

Generates and returns a vector filled with a sequence of integers.

Parameters

size	The size of the vector to generate.

Returns

A vector of integers.

6.1.2.6 getInput()

```
int helpers::getInput (
            int &  )
```

Gets integer input from the user between 0 and 100.

Parameters

number	Reference to an integer to store the input.

Returns

The entered integer.

6.1.2.7 print()

```
void helpers::print (
            const std::string & message )
```

Print a message to standard output.

Parameters

message	The message to print.

6.1.2.8 printLine()

```
void helpers::printLine (
            int len = 80 )
```

Prints a horizontal line of a specified length.

Parameters

len	The length of the line to print. Defaults to 80 characters.

6.1.2.9 printTitle()

```
void helpers::printTitle (
            const std::string & ,
            int ,
            Align ,
            int  )
```

Prints a title with specified alignment and optional border lines.

Parameters

title	The title to print.
totalLength	The total length of the line.
align	The alignment of the title.
n	Specifies if borders should be printed. A value of 1 prints top and bottom borders.

Chapter 7

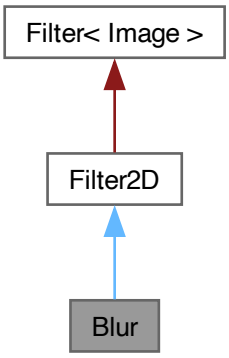
Class Documentation

7.1 Blur Class Reference

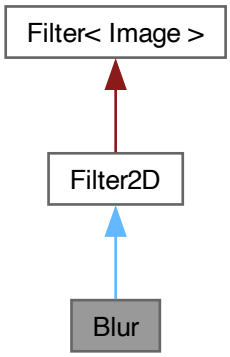
Class derived from Filter2D to apply blur effects on images.

`#include <Blur.h>`

Inheritance diagram for Blur:



Collaboration diagram for Blur:



Public Member Functions

- Blur (Kernel &kernel)

  Constructor that initializes the blur filter with a specified kernel.
- void apply (Image &image) override

  Applies the blur filter to a 2D image.
- const Kernel & getKernel () const

  Gets the kernel used by the blur filter.

Public Member Functions inherited from Filter2D

- Filter2D ()

  Default constructor.

7.1.1 Detailed Description

Class derived from Filter2D to apply blur effects on images.

The Blur class provides functionalities to apply a 2D blur filter using different types of kernels. It holds a Kernel object to be applied on Image objects.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 Blur()

```
Blur::Blur (
            Kernel & kernel )  [explicit]
```

Constructor that initializes the blur filter with a specified kernel.

Parameters

kernel	Reference to the kernel to be used for blurring.

7.1.3 Member Function Documentation

7.1.3.1 apply()

```
void Blur::apply (
            Image & image )  [override], [virtual]
```

Applies the blur filter to a 2D image.

Parameters

image	Reference to the image to apply the blur filter on.

Note

This function overrides the pure virtual function in the base class Filter2D.

Implements Filter2D.

7.1.3.2 getKernel()

```
const Kernel & Blur::getKernel ( ) const
```

Gets the kernel used by the blur filter.

Returns

A constant reference to the kernel.

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

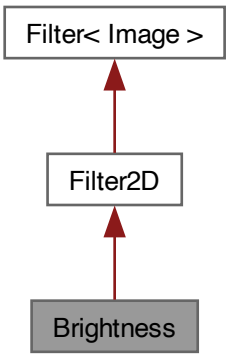
- include/Blur.h

7.2 Brightness Class Reference

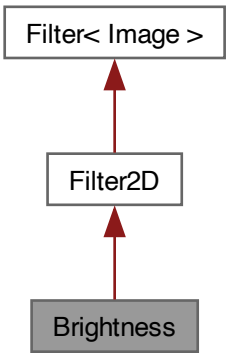
Implements brightness adjustment for images.

```
#include <Brightness.h>
```

Inheritance diagram for Brightness:



Collaboration diagram for Brightness:



Public Member Functions

- Brightness (int brightnessValue)

  Constructs a new Brightness filter object with a specified brightness adjustment value.
- void apply (Image &image)

  Applies brightness adjustment to the given image.

7.2.1 Detailed Description

Implements brightness adjustment for images.

The Brightness class is derived from Filter2D and allows for the brightness adjustment of an Image object. It modifies the brightness by adding a specified value to the pixel values of the image.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 Brightness()

```
Brightness::Brightness (
            int brightnessValue )
```

Constructs a new Brightness filter object with a specified brightness adjustment value.

Parameters

brightnessValue	The brightness adjustment value. Positive values increase brightness, while negative values decrease it.

7.2.3 Member Function Documentation

7.2.3.1 apply()

```
void Brightness::apply (
            Image & image )  [virtual]
```

Applies brightness adjustment to the given image.

Modifies the given Image object by adjusting its brightness. The method iterates over each pixel, adjusting the brightness by adding the specified brightness value to the pixel values, while ensuring that the resulting value remains within the valid range of [0, 255].

Parameters

image	Reference to the Image object to be processed.

Implements Filter2D.

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

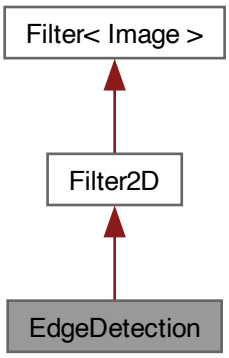
- include/Brightness.h

7.3 EdgeDetection Class Reference

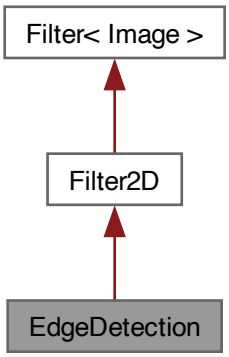
Implements edge detection algorithms for images.

```
#include <EdgeDetection.h>
```

Inheritance diagram for EdgeDetection:



Collaboration diagram for EdgeDetection:



Public Member Functions

- EdgeDetection (EdgeDetectionAlgorithm algorithm=EdgeDetectionAlgorithm::Sobel)

  Constructs an EdgeDetection object with a specified algorithm.
- void apply (Image &image)

  Applies the selected edge detection algorithm to an image.

7.3.1 Detailed Description

Implements edge detection algorithms for images.

The EdgeDetection class, derived from Filter2D, is designed to apply various edge detection algorithms to an Image object. It highlights the edges and contours within images by employing different convolution kernels.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 EdgeDetection()

```
EdgeDetection::EdgeDetection (
            EdgeDetectionAlgorithm algorithm = EdgeDetectionAlgorithm::Sobel )
```

Constructs an EdgeDetection object with a specified algorithm.

Parameters

algorithm	The edge detection algorithm to be applied. Defaults to Sobel if not specified.

7.3.3 Member Function Documentation

7.3.3.1 apply()

```
void EdgeDetection::apply (
            Image & image )  [virtual]
```

Applies the selected edge detection algorithm to an image.

Modifies the given Image object by applying the edge detection algorithm specified at construction. The function adjusts image pixels to highlight edges and contours based on the chosen algorithm.

Parameters

image	Reference to the Image object to be processed.

Implements Filter2D.

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

- include/EdgeDetection.h

7.4 Experiment2DBlur Class Reference

Class for running multiple blurring experiments on an image.

`#include <Experiment2DBlur.h>`

Public Member Functions

- Experiment2DBlur (int &numExperiments, std::string &imgFilePath, int &initKernelSize, std::string &kernel↩ Type, int &kernelSizeJumps, double &sigma)

  Constructor that sets up the experiment parameters.
- void runNExperiments ()

  Runs the specified number of experiments with different kernel sizes.
- const int & getNumExperiments () const

  Gets the number of experiments to be run.

7.4.1 Detailed Description

Class for running multiple blurring experiments on an image.

This class handles the setup and execution of a series of blurring experiments. Each experiment uses a different kernel size or type to apply a blur effect to a single image.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 Experiment2DBlur()

```
Experiment2DBlur::Experiment2DBlur (
            int & numExperiments,
            std::string & imgFilePath,
            int & initKernelSize,
            std::string & kernelType,
            int & kernelSizeJumps,
            double & sigma )
```

Constructor that sets up the experiment parameters.

Parameters

numExperiments	Number of experiments to run.
imgFilePath	Path to the image file.
initKernelSize	Initial size of the kernel.
kernelType	Type of the kernel to use for blurring.
kernelSizeJumps	Increments by which to increase the kernel size after each experiment.
sigma	Sigma value for Gaussian kernels.

7.4.3 Member Function Documentation

7.4.3.1 getNumExperiments()

```
const int & Experiment2DBlur::getNumExperiments ( ) const
```

Gets the number of experiments to be run.

Returns

The total number of experiments.

7.4.3.2 runNExperiments()

```
void Experiment2DBlur::runNExperiments ( )
```

Runs the specified number of experiments with different kernel sizes.

-

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

- include/Experiment2DBlur.h

7.5 Filter< T > Class Template Reference

Base class for different types of filters.

`#include <Filter.h>`

Public Member Functions

- Filter ()
  default constructor.
- virtual void apply (T &data)=0
  Pure virtual function for applying the filter to input data.
- virtual ~Filter ()=default
  Virtual destructor.

7.5.1 Detailed Description

template<typename T>
class Filter< T >

Base class for different types of filters.

The Filter class provides a common interface and foundational functionality for various filtering operations. It is intended to be subclassed by more specific filter types which implement actual filtering algorithms.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 Filter()

```
template<typename T >
Filter< T >::Filter ( )
```

default constructor.

-

7.5.2.2 ~Filter()

```
template<typename T >
virtual Filter< T >::~Filter ( )  [virtual], [default]
```

Virtual destructor.

-

7.5.3 Member Function Documentation

7.5.3.1 apply()

```
template<typename T >
virtual void Filter< T >::apply (
            T & data )  [pure virtual]
```

Pure virtual function for applying the filter to input data.

Parameters

data	The input data (e.g., Image or Volume) to apply the filter to.

This is a pure virtual function that must be implemented by derived classes to define the specific filtering operation for the corresponding data type.

Implemented in Brightness, EdgeDetection, Grayscale, HistogramEqualisation, SaltAndPepperNoise, Thresholding, Blur, Filter2D, GaussianFilter3D, MedianFilter3D, and Filter3D.

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

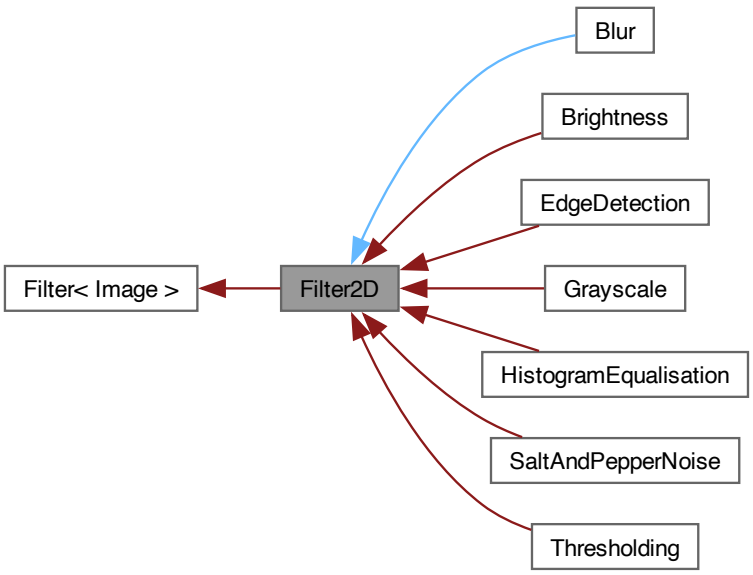
- include/Filter.h

7.6 Filter2D Class Reference

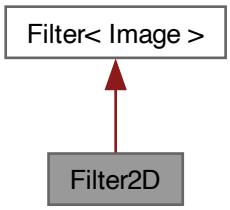
An abstract base class for 2D image filters. Class derived from Filter.

`#include <Filter2D.h>`

Inheritance diagram for Filter2D:



Collaboration diagram for Filter2D:



Public Member Functions

- Filter2D ()
  Default constructor.
- void apply (Image &image) override=0
  Pure virtual function for applying the filter to an image.

7.6.1 Detailed Description

An abstract base class for 2D image filters. Class derived from Filter.

The Filter2D class is an abstract base class that defines the common interface for 2D image filters. Derived classes must implement the apply() method to define the specific filtering operation.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 Filter2D()

```
Filter2D::Filter2D ( )
```

Default constructor.

-

7.6.3 Member Function Documentation

7.6.3.1 apply()

```
void Filter2D::apply (
            Image & image )  [override], [pure virtual]
```

Pure virtual function for applying the filter to an image.

Parameters

image	The image to apply the filter to.

This is a pure virtual function that must be implemented by derived classes to define the specific filtering operation.

Implements Filter< Image >.

Implemented in Brightness, EdgeDetection, Grayscale, HistogramEqualisation, SaltAndPepperNoise, Thresholding, and Blur.

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

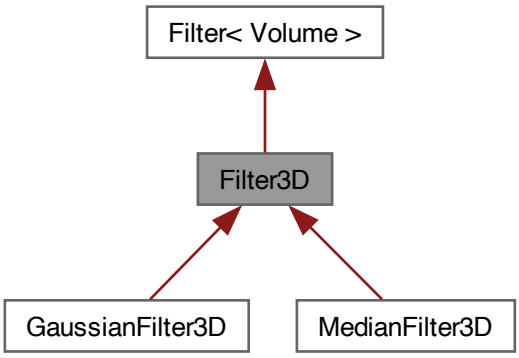
- include/Filter2D.h

7.7 Filter3D Class Reference

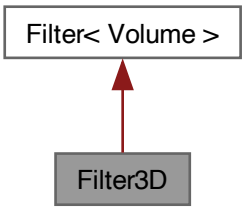
An abstract base class for 3D volume filters. Class derived from Filter.

`#include <Filter3D.h>`

Inheritance diagram for Filter3D:



Collaboration diagram for Filter3D:



Public Member Functions

- Filter3D ()

  Default constructor.
- ~Filter3D ()

  Destructor.
- void apply (Volume &volume) override=0

  Pure virtual function for applying the filter to a volume.
- virtual void info ()=0

  Pure virtual function for displaying information about the filter.

7.7.1 Detailed Description

An abstract base class for 3D volume filters. Class derived from Filter.

The Filter3D class is an abstract base class that defines the common interface for 3D volume filters. Derived classes must implement the apply() and info() methods to define the specific filtering operation and provide information about the filter, respectively.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 Filter3D()

```
Filter3D::Filter3D ( )
```

Default constructor.

-

7.7.2.2 ~Filter3D()

```
Filter3D::~Filter3D ( )
```

Destructor.

-

7.7.3 Member Function Documentation

7.7.3.1 apply()

```
void Filter3D::apply (
            Volume & volume )  [override], [pure virtual]
```

Pure virtual function for applying the filter to a volume.

Parameters

volume	The volume to apply the filter to.

This is a pure virtual function that must be implemented by derived classes to define the specific filtering operation for 3D volumes.

Implements Filter< Volume >.

Implemented in GaussianFilter3D, and MedianFilter3D.

7.7.3.2 info()

```
virtual void Filter3D::info ( )  [pure virtual]
```

Pure virtual function for displaying information about the filter.

This is a pure virtual function that must be implemented by derived classes to provide information about the specific filter implementation.

Implemented in GaussianFilter3D, and MedianFilter3D.

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

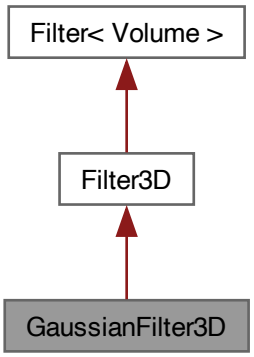
- include/Filter3D.h

7.8 GaussianFilter3D Class Reference

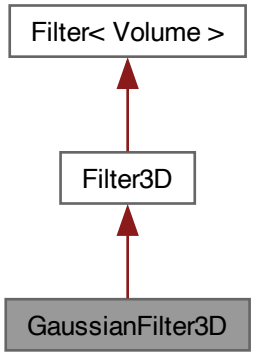
Implements a 3D Gaussian filter on volumetric data.

```
#include <GaussianFilter3D.h>
```

Inheritance diagram for GaussianFilter3D:



Collaboration diagram for GaussianFilter3D:



Public Member Functions

- **GaussianFilter3D** ()

  Default constructor that initializes a Gaussian filter with a kernel size of 3 and sigma of 1.0.
- GaussianFilter3D (int filterSize, double sigma)

  Constructor to initialize a Gaussian filter with a specified kernel size and sigma.
- ~**GaussianFilter3D** ()

  Default destructor.
- void apply (Volume &volume) override

  Applies the Gaussian filter to a given volume.
- void info () override

  Outputs information about the filter settings to the standard output.
- void setFilter (int filterSize, double sigma)

  Sets or updates the filter size and sigma of the Gaussian kernel.

7.8.1 Detailed Description

Implements a 3D Gaussian filter on volumetric data.

This class provides functionality to apply a Gaussian filter to a Volume object. It encapsulates the filter size and standard deviation (sigma) and uses them to create a Gaussian kernel for the filtering process.

7.8.2 Constructor & Destructor Documentation

7.8.2.1 GaussianFilter3D()

```
GaussianFilter3D::GaussianFilter3D (
            int filterSize,
            double sigma )
```

Constructor to initialize a Gaussian filter with a specified kernel size and sigma.

Parameters

filterSize	Size of the filter kernel.
sigma	Standard deviation for the Gaussian distribution.

7.8.3 Member Function Documentation

7.8.3.1 apply()

```
void GaussianFilter3D::apply (
            Volume & volume )  [override], [virtual]
```

Applies the Gaussian filter to a given volume.

Parameters

volume	Reference to the Volume object to be processed.

Implements Filter3D.

7.8.3.2 info()

```
void GaussianFilter3D::info ( )  [override], [virtual]
```

Outputs information about the filter settings to the standard output.

Implements Filter3D.

7.8.3.3 setFilter()

```
void GaussianFilter3D::setFilter (
            int filterSize,
            double sigma )
```

Sets or updates the filter size and sigma of the Gaussian kernel.

Parameters

filterSize	New size of the filter kernel.
sigma	New standard deviation for the Gaussian distribution.

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

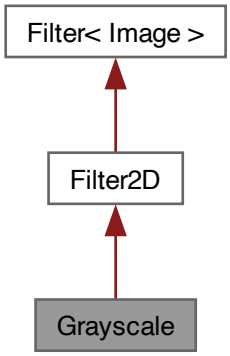
- include/GaussianFilter3D.h

7.9 Grayscale Class Reference

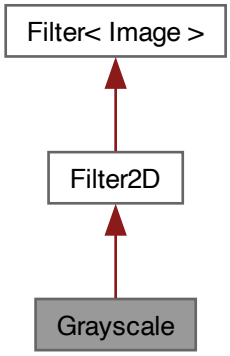
Implements image grayscaling.

`#include <Grayscale.h>`

Inheritance diagram for Grayscale:



Collaboration diagram for Grayscale:



Public Member Functions

- Grayscale ()=default

  Constructs a new Grayscale object.
- void apply (Image &image)

  Applies the grayscale filter to a specified Image object.

7.9.1 Detailed Description

Implements image grayscaling.

The Grayscale class inherits from Filter2D and offers functionality to convert a given Image object into a grayscale image. It overrides the apply method to apply the grayscale filter.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 Grayscale()

```
Grayscale::Grayscale ( )  [default]
```

Constructs a new Grayscale object.

The default constructor.

7.9.3 Member Function Documentation

7.9.3.1 apply()

```
void Grayscale::apply (
            Image & image )  [virtual]
```

Applies the grayscale filter to a specified Image object.

Takes an Image object as a parameter and applies grayscaling to it. This method modifies the Image object to reflect the conversion to grayscale, changing each pixel's color to its grayscale equivalent based on luminance.

Parameters

image	Reference to an Image object to be converted into grayscale.

Implements Filter2D.

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

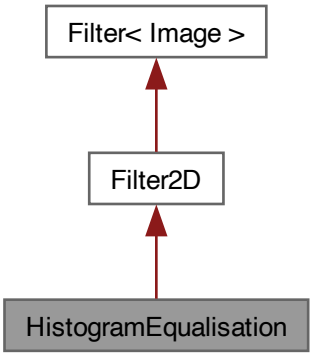
- include/Grayscale.h

7.10 HistogramEqualisation Class Reference

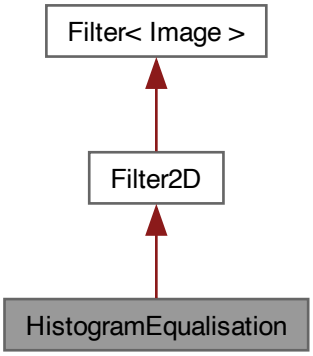
Implements histogram equalization for image contrast enhancement.

```
#include <HistogramEqualisation.h>
```

Inheritance diagram for HistogramEqualisation:



Collaboration diagram for HistogramEqualisation:



Public Member Functions

- HistogramEqualisation (bool isHSV)

  Constructs a new Histogram Equalisation filter object.
- void apply (Image &image)

  Applies histogram equalization to the given image.

7.10.1 Detailed Description

Implements histogram equalization for image contrast enhancement.

HistogramEqualisation is a derived class of Filter2D designed to apply histogram equalization on an Image object. This technique improves the contrast of images by stretching out the intensity distribution. The class supports both the RGB and HSV/HSL color models for equalization.

7.10.2 Constructor & Destructor Documentation

7.10.2.1 HistogramEqualisation()

```
HistogramEqualisation::HistogramEqualisation (
            bool isHSV )
```

Constructs a new Histogram Equalisation filter object.

Initializes the filter with the choice of using HSV or HSL. When set to true, for RGB images, the equalization will be performed on the V channel of the HSV color space. Otherwise, equalization will be applied on the L channel of the HSL color space.

Parameters

isHSV	Boolean flag to determine the color space for equalization (true for HSV, false for HSL).

7.10.3 Member Function Documentation

7.10.3.1 apply()

```
void HistogramEqualisation::apply (
            Image & image )  [virtual]
```

Applies histogram equalization to the given image.

Modifies the given Image object by applying histogram equalization to enhance its contrast. The method decides the color space for equalization based on the isHSV member variable.

Parameters

image	Reference to the Image object to be processed.

Implements Filter2D.

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

- include/HistogramEqualisation.h

7.11 Image Class Reference

Class for handling image loading, processing, and saving. The Image class used the STB image libraries are used to load, manipulate, and save images.

```
#include <Image.h>
```

Public Member Functions

- Image (std::string &filepath)

  Constructor that loads an image from a file path.
- Image (std::string &filepath, int &c)

  Constructor that loads an image from a filepath with the desired number of channels to work with.
- Image (std::string &filepath, int &h, int &w, int &c)

  Constructor that loads an image from a filepath with the desired height, width and channels to work with.
- Image ()

  default constructor.
- Image (const Image &imageCopy)

  copy constructor.
- ~Image ()

  destructor.
- int & operator() (int x, int y, int z)

  Override the () operator to access pixel value at specified height (x), width(y) and channel(z).
- void readImage ()

  Load image from file.
- void createImageData ()

  Initialize image data into a x, y, z tensor structure.
- void createPaddedImageData (int &pad)

  Overwrite the .data attr with a padded version of the raw image.
- void outputImage (const std::string &outputFilePath)

  Save image to file.
- void **setPixel** (int &h, int &w, int &c, int &value)
- void **setHeight** (int &h)
- void **setWidth** (int &w)
- void **setDataPtr** (unsigned char *&p)
- void **setChannels** (int &c)
- void **setDesiredChannels** (int &dc)
- unsigned char & **getPixel** (int &h, int &w, int &c)
- const int & **getHeight** () const
- const int & **getWidth** () const
- const int & **getRawHeight** () const
- const int & **getRawWidth** () const
- const int & **getChannels** () const
- const int & **getDesiredChannels** () const
- unsigned char *& **getDataPtr** ()

Friends

- std::ostream & operator<< (std::ostream &s, Image const &m)

  Overloads the << operator for easy printing of Image objects.

7.11.1 Detailed Description

Class for handling image loading, processing, and saving. The Image class used the STB image libraries are used to load, manipulate, and save images.

7.11.2 Constructor & Destructor Documentation

7.11.2.1 Image() [1/5]

```
Image::Image (
            std::string & filepath )  [explicit]
```

Constructor that loads an image from a file path.

Parameters

filepath	The path to the image file to be loaded.

7.11.2.2 Image() [2/5]

```
Image::Image (
            std::string & filepath,
            int & c )
```

Constructor that loads an image from a filepath with the desired number of channels to work with.

Parameters

filepath	Path to the image file.
c	Desired image channels.

7.11.2.3 Image() [3/5]

```
Image::Image (
            std::string & filepath,
            int & h,
            int & w,
            int & c )
```

Constructor that loads an image from a filepath with the desired height, width and channels to work with.

Parameters

filepath	Path to the image file.
h	Desired image height.
w	Desired image width.
c	Desired image channels.

7.11.2.4 Image() [4/5]

```
Image::Image ( )
```

default constructor.

-

7.11.2.5 Image() [5/5]

```
Image::Image (
            const Image & imageCopy )
```

copy constructor.

-

7.11.2.6 ~Image()

```
Image::~Image ( )
```

destructor.

-

7.11.3 Member Function Documentation

7.11.3.1 createImageData()

```
void Image::createImageData ( )
```

Initialize image data into a x, y, z tensor structure.

-

7.11.3.2 createPaddedImageData()

```
void Image::createPaddedImageData (
            int & pad )
```

Overwrite the .data attr with a padded version of the raw image.

-

7.11.3.3 operator()()

```
int & Image::operator() (
            int x,
            int y,
            int z )
```

Override the () operator to access pixel value at specified height (x), width(y) and channel(z).

Parameters

x	X-coordinate (height).
y	Y-coordinate (width).
z	Channel index.

Returns

Reference to the pixel value.

7.11.3.4 outputImage()

```
void Image::outputImage (
            const std::string & outputFilePath )
```

Save image to file.

Parameters

outputFilePath	Path to save the image.

7.11.3.5 readImage()

```
void Image::readImage ( )
```

Load image from file.

Returns

'unsigned char ∗' which points to the pixel data in a 1D buffer.

7.11.4 Friends And Related Symbol Documentation

7.11.4.1 operator<<

```
std::ostream & operator<< (
            std::ostream & s,
            Image const & m )  [friend]
```

Overloads the << operator for easy printing of Image objects.

-

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

- include/Image.h

7.12 Kernel Class Reference

Class for instantiating 2D kernels and applying them on images.

`#include <Kernel.h>`

Public Member Functions

- Kernel ()
  Default constructor.
- Kernel (int &kernelSize, std::string &kernelType)
  Constructor that initializes a kernel with a specified size and type.
- Kernel (int &kernelSize, std::string &kernelType, double &sigma)
  Constructor for a Gaussian kernel with a specified size, type, and sigma.
- void **setPadding** (int &p)
- void **setKernel** (std::vector< double > &k)
- const std::vector< double > & **getKernel** () const
- const int & **getKernelSize** () const
- const int & **getPadding** () const
- const double & **getSigma** () const
- const KernelType & **getKernelType** () const

Static Public Member Functions

- static std::vector< double > applyKernel (Image &image, Kernel &k, int &iPoint, int &jPoint, int &zPoint)
  Applies the kernel to a specific point in the image.
- static KernelType stringToKernelType (std::string &str)
  Converts a kernel type string to a KernelType enum.

7.12.1 Detailed Description

Class for instantiating 2D kernels and applying them on images.

The Kernel class provides functionalities to create different types of image processing kernels such as Gaussian, Box, and Median, and apply them to Image objects.

7.12.2 Constructor & Destructor Documentation

7.12.2.1 Kernel() [1/3]

```
Kernel::Kernel ( )
```

Default constructor.

-

7.12.2.2 Kernel() [2/3]

```
Kernel::Kernel (
            int & kernelSize,
            std::string & kernelType )
```

Constructor that initializes a kernel with a specified size and type.

Parameters

kernelSize	Size of the kernel.
kernelType	Type of the kernel as a string.

7.12.2.3 Kernel() [3/3]

```
Kernel::Kernel (
            int & kernelSize,
            std::string & kernelType,
            double & sigma )
```

Constructor for a Gaussian kernel with a specified size, type, and sigma.

Parameters

kernelSize	Size of the kernel.
kernelType	Type of the kernel as a string.
sigma	Sigma value for Gaussian kernels.

7.12.3 Member Function Documentation

7.12.3.1 applyKernel()

```
static std::vector< double > Kernel::applyKernel (
            Image & image,
            Kernel & k,
            int & iPoint,
            int & jPoint,
            int & zPoint )  [static]
```

Applies the kernel to a specific point in the image.

Parameters

image	Reference to the image.
k	Kernel to be applied.
iPoint	X-coordinate in the image.
jPoint	Y-coordinate in the image.
zPoint	Z-coordinate (channel) in the image.

Returns

A vector<double> containing the pixel and its neighbourhood values after applying the kernel.

7.12.3.2 stringToKernelType()

```
static KernelType Kernel::stringToKernelType (
```

```
std::string & str )  [static]
```

Converts a kernel type string to a KernelType enum.

Parameters

str	String representation of the kernel type.

Returns

The corresponding KernelType enum value.

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

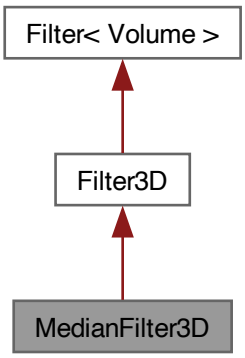
- include/Kernel.h

7.13 MedianFilter3D Class Reference

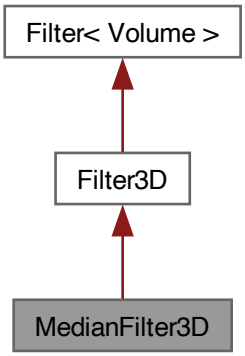
Implements median filtering for volumetric data.

```
#include <MedianFilter3D.h>
```

Inheritance diagram for MedianFilter3D:



Collaboration diagram for MedianFilter3D:



Public Member Functions

- **MedianFilter3D** ()

  Default constructor that initializes a median filter with a default kernel size.
- MedianFilter3D (int filterSize)

  Constructor that initializes a median filter with a specified kernel size.
- ~**MedianFilter3D** ()

  Destructor for the median filter.
- void apply (Volume &volume) override

  Applies the median filter to a given volume.
- void info () override

  Outputs information about the filter settings to the standard output.
- void setFilter (int filterSize)

  Sets the filter size for the median filter.
- unsigned char initHistogramAndFindMedian (const std::vector< unsigned char * > &data, int s, int y, int x, int c, int width, int height, int slices, std::vector< int > &histogram)

  Initializes the histogram used in finding the median value and returns the median value.

7.13.1 Detailed Description

Implements median filtering for volumetric data.

MedianFilter3D extends the Filter3D class with methods specifically tailored for applying a median filter to 3D volume data. The class handles the creation and application of a median filter kernel of a given size.

7.13.2 Constructor & Destructor Documentation

7.13.2.1 MedianFilter3D()

```
MedianFilter3D::MedianFilter3D (
            int filterSize )
```

Constructor that initializes a median filter with a specified kernel size.

Parameters

filterSize	The size of the filter kernel to be used.

7.13.3 Member Function Documentation

7.13.3.1 apply()

```
void MedianFilter3D::apply (
            Volume & volume )  [override], [virtual]
```

Applies the median filter to a given volume.

Parameters

volume	Reference to the Volume object to be processed.

Implements Filter3D.

7.13.3.2 info()

```
void MedianFilter3D::info ( )  [override], [virtual]
```

Outputs information about the filter settings to the standard output.

Implements Filter3D.

7.13.3.3 initHistogramAndFindMedian()

```
unsigned char MedianFilter3D::initHistogramAndFindMedian (
            const std::vector< unsigned char * > & data,
            int s,
            int y,
            int x,
            int c,
            int width,
            int height,
            int slices,
            std::vector< int > & histogram )
```

Initializes the histogram used in finding the median value and returns the median value.

Parameters

data	Reference to the volume data.
s	The current slice index.
y	The current row index.
x	The current column index.
c	The current channel index.
width	The width of the volume.
height	The height of the volume.
slices	The number of slices in the volume.
histogram	Reference to the histogram vector to be used for finding the median.

Returns

The median value as an unsigned char.

7.13.3.4 setFilter()

```
void MedianFilter3D::setFilter (
            int filterSize )
```

Sets the filter size for the median filter.

Parameters

filterSize	The new size of the median filter kernel.

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

- include/MedianFilter3D.h

7.14 Projection Class Reference

Class to perform various projection operations on volume data.

```
#include <Projection.h>
```

Public Member Functions

- **Projection** ()
  Default constructor for the Projection class.
- ~**Projection** ()
  Default destructor for the Projection class.
- void applyMIP (const Volume &volume, const std::string &outputPath, int startSlice=-1, int endSlice=-1)
  Applies Maximum Intensity Projection to a volume and saves the result to a specified path.
- void applyMinIP (const Volume &volume, const std::string &outputPath, int startSlice=-1, int endSlice=-1)
  Applies Minimum Intensity Projection to a volume and saves the result to a specified path.
- void applyAIP (const Volume &volume, const std::string &outputPath, int startSlice=-1, int endSlice=-1)
  Applies Average Intensity Projection to a volume and saves the result to a specified path.

7.14.1 Detailed Description

Class to perform various projection operations on volume data.

This class provides different projection operations that can be applied to Volume objects. Projections include Maximum Intensity Projection (MIP), Minimum Intensity Projection (MinIP), and Average Intensity Projection (AIP). These projections help in visualizing volumetric data by projecting the data into a two-dimensional image.

7.14.2 Member Function Documentation

7.14.2.1 applyAIP()

```
void Projection::applyAIP (
            const Volume & volume,
            const std::string & outputPath,
            int startSlice = -1,
            int endSlice = -1 )
```

Applies Average Intensity Projection to a volume and saves the result to a specified path.

Parameters

volume	The volume to apply AIP to.
outputPath	The path where the resulting image will be saved.
startSlice	The starting slice index to consider for the projection.
endSlice	The ending slice index to consider for the projection.

7.14.2.2 applyMinIP()

```
void Projection::applyMinIP (
            const Volume & volume,
            const std::string & outputPath,
            int startSlice = -1,
            int endSlice = -1 )
```

Applies Minimum Intensity Projection to a volume and saves the result to a specified path.

Parameters

volume	The volume to apply MinIP to.
outputPath	The path where the resulting image will be saved.
startSlice	The starting slice index to consider for the projection.
endSlice	The ending slice index to consider for the projection.

7.14.2.3 applyMIP()

```
void Projection::applyMIP (
            const Volume & volume,
            const std::string & outputPath,
            int startSlice = -1,
            int endSlice = -1 )
```

Applies Maximum Intensity Projection to a volume and saves the result to a specified path.

Parameters

volume	The volume to apply MIP to.

Parameters

outputPath	The path where the resulting image will be saved.
startSlice	The starting slice index to consider for the projection.
endSlice	The ending slice index to consider for the projection.

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

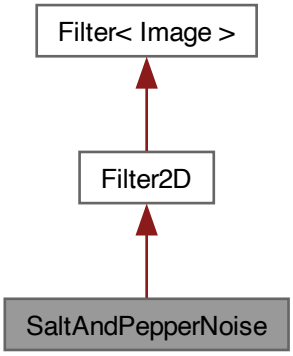
- include/Projection.h

7.15 SaltAndPepperNoise Class Reference

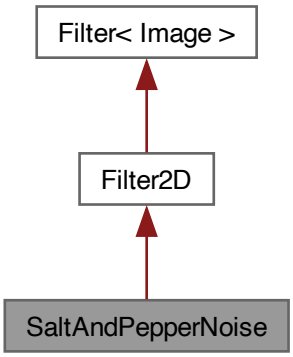
Implements salt-and-pepper noise addition to images.

`#include <SaltAndPepperNoise.h>`

Inheritance diagram for SaltAndPepperNoise:



Collaboration diagram for SaltAndPepperNoise:



Public Member Functions

- SaltAndPepperNoise (double noisePercentage)

  Constructs a new Salt and Pepper Noise object.
- void apply (Image &image)

  Applies salt-and-pepper noise to the given image.

7.15.1 Detailed Description

Implements salt-and-pepper noise addition to images.

The SaltandPepperNoise class, derived from Filter2D, allows for the addition of salt-and-pepper noise to an Image object. The noise is characterized by random black and white pixels scattered across the image, with the degree of noise controlled by a percentage value.

7.15.2 Constructor & Destructor Documentation

7.15.2.1 SaltAndPepperNoise()

```
SaltAndPepperNoise::SaltAndPepperNoise (
            double noisePercentage )
```

Constructs a new Salt and Pepper Noise object.

Parameters

noisePercentage	The percentage of the image to be affected by salt-and-pepper noise, specified as a double. For example, 10 means 10% of the image pixels will be affected.

7.15.3 Member Function Documentation

7.15.3.1 apply()

```
void SaltAndPepperNoise::apply (
            Image & image )  [virtual]
```

Applies salt-and-pepper noise to the given image.

Modifies the given Image object by adding salt-and-pepper noise to it, based on the specified noise percentage at construction. The method randomly selects pixels to change to black or white, simulating the noise.

Parameters

image	Reference to the Image object to be processed.

Implements Filter2D.

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

- include/SaltAndPepperNoise.h

7.16 Slice Class Reference

Class for extracting 2D slices from a 3D volume.

`#include <Slice.h>`

Public Member Functions

- **Slice** ()

  Default constructor for the Slice class.
- ~**Slice** ()

  Default destructor for the Slice class.
- void dosliceYZ (const Volume &volume, int x, const std::string &outputPath)

  Extracts a YZ slice from the volume at a given X coordinate and saves it as an image.
- void dosliceXZ (const Volume &volume, int y, const std::string &outputPath)

  Extracts an XZ slice from the volume at a given Y coordinate and saves it as an image.

Static Public Member Functions

- static void sliceYZ (const Volume &volume, int x, const std::string &outputPath)

  Static method to extract a YZ slice from the volume at a given X coordinate.
- static void sliceXZ (const Volume &volume, int y, const std::string &outputPath)

  Static method to extract an XZ slice from the volume at a given Y coordinate.

7.16.1 Detailed Description

Class for extracting 2D slices from a 3D volume.

Provides functionality to extract specific 2D slices (YZ or XZ plane) from a given 3D volume and save them as 2D images. The class supports extracting slices at a given coordinate along the width or height of the volume.

7.16.2 Member Function Documentation

7.16.2.1 dosliceXZ()

```
void Slice::dosliceXZ (
            const Volume & volume,
            int y,
            const std::string & outputPath )
```

Extracts an XZ slice from the volume at a given Y coordinate and saves it as an image.

Parameters

volume	The volume to extract the slice from.
y	The Y coordinate at which to extract the slice.
outputPath	The file path where the slice image will be saved.

7.16.2.2 dosliceYZ()

```
void Slice::dosliceYZ (
            const Volume & volume,
            int x,
            const std::string & outputPath )
```

Extracts a YZ slice from the volume at a given X coordinate and saves it as an image.

Parameters

volume	The volume to extract the slice from.
x	The X coordinate at which to extract the slice.
outputPath	The file path where the slice image will be saved.

7.16.2.3 sliceXZ()

```
static void Slice::sliceXZ (
            const Volume & volume,
            int y,
            const std::string & outputPath )  [inline], [static]
```

Static method to extract an XZ slice from the volume at a given Y coordinate.

Parameters

volume	The volume to extract the slice from.
y	The Y coordinate at which to extract the slice.
outputPath	The file path where the slice image will be saved.

7.16.2.4 sliceYZ()

```
static void Slice::sliceYZ (
            const Volume & volume,
            int x,
            const std::string & outputPath )  [inline], [static]
```

Static method to extract a YZ slice from the volume at a given X coordinate.

Parameters

volume	The volume to extract the slice from.
x	The X coordinate at which to extract the slice.
outputPath	The file path where the slice image will be saved.

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

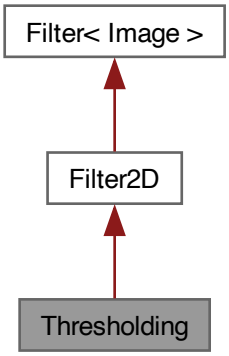
- include/Slice.h

7.17 Thresholding Class Reference

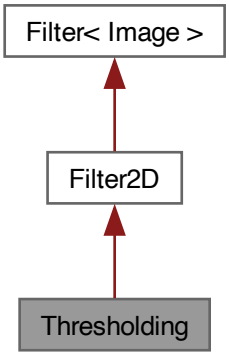
Implements thresholding for image processing.

`#include <Thresholding.h>`

Inheritance diagram for Thresholding:



Collaboration diagram for Thresholding:



Public Member Functions

- Thresholding (double threshold, bool isHSV)

  Constructs a Thresholding filter with a given threshold and color space option.
- void apply (Image &image)

  Applies thresholding to an image.

7.17.1 Detailed Description

Implements thresholding for image processing.

Thresholding converts an image to a binary image where pixels are turned either black or white based on a threshold. For color images, thresholding can be applied to the intensity/lightness value in the HSV/HSL color space.

7.17.2 Constructor & Destructor Documentation

7.17.2.1 Thresholding()

```
Thresholding::Thresholding (
            double threshold,
            bool isHSV )
```

Constructs a Thresholding filter with a given threshold and color space option.

Parameters

threshold	The intensity threshold for the binary conversion. Pixels below this value will be black, above will be white.
isHSV	Specifies whether to perform thresholding in the HSV color space or in the HSV color space for RGB images.

7.17.3 Member Function Documentation

7.17.3.1 apply()

```
void Thresholding::apply (
            Image & image )  [virtual]
```

Applies thresholding to an image.

The method modifies the given Image object, applying thresholding to convert it into a binary image based on the specified threshold value. For color images, the thresholding can be performed on the value/lightness channel in the HSV/HSL color space if specified.

Parameters

image	Reference to the Image object to be processed.

Implements Filter2D.

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

- include/Thresholding.h

7.18 UI Class Reference

Main User Interface class for handling 2D and 3D UI operations.

`#include <UI.h>`

Public Member Functions

- **UI** ()
  Default constructor for the UI class.
- ~**UI** ()
  Default destructor for the UI class.
- void **run** ()
  Runs the UI interaction loop.
- int mainMenu ()
  Displays the main menu and handles user interaction.

7.18.1 Detailed Description

Main User Interface class for handling 2D and 3D UI operations.

This class contains methods to run the main menu of the application and to delegate control to either 2D or 3D user interface classes based on user input. It provides an entry point for user interaction with the system.

7.18.2 Member Function Documentation

7.18.2.1 mainMenu()

`int UI::mainMenu ( )  [inline]`

Displays the main menu and handles user interaction.

Returns

Integer indicating the status upon exit, with 0 for normal termination.

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

- include/UI.h

7.19 UI2D Class Reference

User interface class for 2D visualization and processing options.

`#include <UI2D.h>`

Public Member Functions

- **UI2D** ()=default
  Default constructor for UI2D class.
- int run ()
  Runs the 2D user interface for image processing.

7.19.1 Detailed Description

User interface class for 2D visualization and processing options.

UI2D class provides a menu-driven interface for 2D operations.

7.19.2 Member Function Documentation

7.19.2.1 run()

```
int UI2D::run ( )
```

Runs the 2D user interface for image processing.

Returns

Return 0 if successful.

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

- include/UI2D.h

7.20 UI3D Class Reference

User interface class for 3D visualization and processing options.

`#include <UI3D.h>`

Public Member Functions

- int run ()
  Starts the UI loop for 3D operations.

7.20.1 Detailed Description

User interface class for 3D visualization and processing options.

UI3D class provides a menu-driven interface for 3D operations such as filtering, projection, and slicing. It interacts with the user to obtain input and output paths, filter parameters, and projection or slice specifications.

7.20.2 Member Function Documentation

7.20.2.1 run()

```
int UI3D::run ( )  [inline]
```

Starts the UI loop for 3D operations.

Returns

Returns the user's selection.

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

- include/UI3D.h

7.21 Volume Class Reference

Manages loading, saving, and manipulating 3D volume data.

```
#include <Volume.h>
```

Public Member Functions

- **Volume** ()
  Default constructor.
- ∼**Volume** ()
  Destructor, frees allocated memory.
- const int & **getHeight** () const
- const int & **getWidth** () const
- const int & **getSlices** () const
- const int & **getChannels** () const
- const std::vector< unsigned char * > & **getData** () const
- void setData (const std::vector< unsigned char * > &newData)
  Sets new volume data, replacing the existing data.
- bool loadVolume (const std::string &directoryPath)
  Loads volume data from a given directory.
- void saveVolume (const std::string &directoryPath)
  Saves the current volume to the specified directory.
- void saveSlice (const std::string &outputPath, const unsigned char *sliceData, int sliceWidth, int sliceHeight) const

Saves a single slice of the volume to a file.

- void **reloadVolume** ()

  Reloads the volume from the original data.

- void cloneData (const std::vector< unsigned char * > &source, std::vector< unsigned char * > &destination)

  Clones data from a source to a destination vector.

- void **printVolumeData** () const

  Prints the volume data to the standard output.

- std::vector< unsigned char > getVolumePixelData () const

  Retrieves the pixel data for the entire volume.

- std::vector< unsigned char > getImagePixelData (const std::string &path) const

  Loads and returns pixel data from an image file.

Static Public Member Functions

- static void generateSamples (const std::string &directory, int count, int width, int height, int seed=123)

  Generates sample volume data for testing purposes.

- static void readAndPrintSamples (const std::string &directory)

  Reads and prints sample data from a directory.

7.21.1 Detailed Description

Manages loading, saving, and manipulating 3D volume data.

This class encapsulates a 3D volume, represented as a series of 2D slices, and provides methods for volume data manipulation, including loading and saving the volume from/to disk, and accessing and modifying the volume data.

7.21.2 Member Function Documentation

7.21.2.1 cloneData()

```
void Volume::cloneData (
            const std::vector< unsigned char * > & source,
            std::vector< unsigned char * > & destination )
```

Clones data from a source to a destination vector.

Parameters

source	Source vector from which to clone data.
destination	Destination vector where data will be cloned to.

7.21.2.2 generateSamples()

```
static void Volume::generateSamples (
            const std::string & directory,
            int count,
            int width,
```

```
            int height,
            int seed = 123 )  [static]
```

Generates sample volume data for testing purposes.

Parameters

directory	Directory where sample data will be saved.
count	Number of samples to generate.
width	Width of the sample images.
height	Height of the sample images.
seed	Seed for random number generation.

7.21.2.3 getImagePixelData()

```
std::vector< unsigned char > Volume::getImagePixelData (
            const std::string & path ) const
```

Loads and returns pixel data from an image file.

Parameters

path	Path to the image file.

Returns

A vector containing the pixel data.

7.21.2.4 getVolumePixelData()

```
std::vector< unsigned char > Volume::getVolumePixelData ( ) const
```

Retrieves the pixel data for the entire volume.

Returns

A vector containing the pixel data.

7.21.2.5 loadVolume()

```
bool Volume::loadVolume (
            const std::string & directoryPath )
```

Loads volume data from a given directory.

Parameters

directoryPath	Path to the directory containing the volume data.

Returns

True if loading is successful, false otherwise.

7.21.2.6 readAndPrintSamples()

```
static void Volume::readAndPrintSamples (
            const std::string & directory )  [static]
```

Reads and prints sample data from a directory.

Parameters

directory	Directory containing the sample data.

7.21.2.7 saveSlice()

```
void Volume::saveSlice (
            const std::string & outputPath,
            const unsigned char * sliceData,
            int sliceWidth,
            int sliceHeight ) const
```

Saves a single slice of the volume to a file.

Parameters

outputPath	Path to the file where the slice will be saved.
sliceData	Data of the slice to be saved.
sliceWidth	Width of the slice.
sliceHeight	Height of the slice.

7.21.2.8 saveVolume()

```
void Volume::saveVolume (
            const std::string & directoryPath )
```

Saves the current volume to the specified directory.

Parameters

directoryPath	Path to the directory where the volume data will be saved.

7.21.2.9 setData()

```
void Volume::setData (
            const std::vector< unsigned char * > & newData )
```

Sets new volume data, replacing the existing data.

Parameters

newData	The new data to be set.

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

- include/Volume.h

Chapter 8

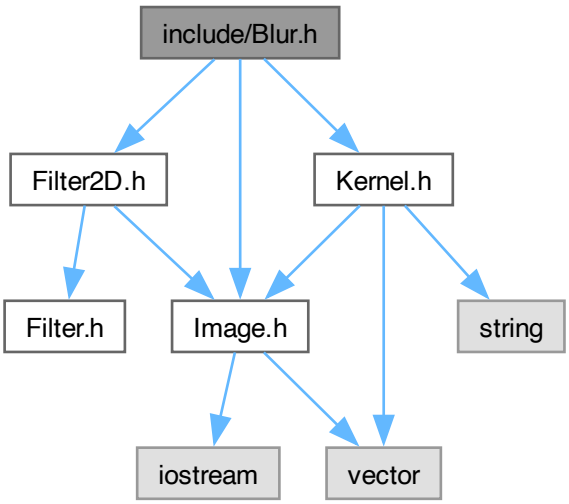
File Documentation

8.1 include/Blur.h File Reference

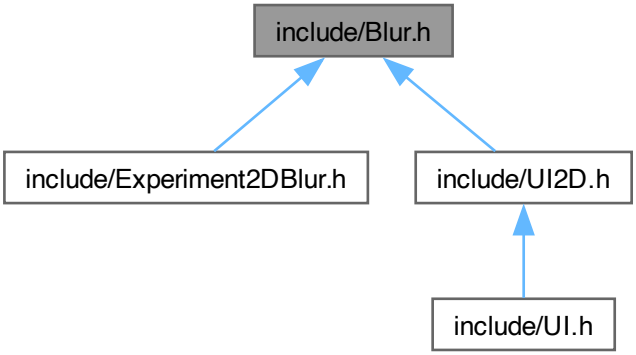
The Blur class is used for applying a 2D blur filter to images using various kernels.

```
#include "Filter2D.h"
#include "Image.h"
#include "Kernel.h"
```

Include dependency graph for Blur.h:



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



Classes

- class Blur

  Class derived from Filter2D to apply blur effects on images.

8.1.1 Detailed Description

The Blur class is used for applying a 2D blur filter to images using various kernels.

Author

Daniel Seal (edsml-ds423)

Copyright

Xavier-Initialization (2024) Daniel Seal (edsml-ds423) Yongwen Chen (acse-yc3321) Zeqi Li (acse-zl123) Jing-Han Huang (edsml-jh123) Wenbo Yu (acse-wy1223) Ning Guan (edsml-ng323)

8.2 Blur.h

Go to the documentation of this file.

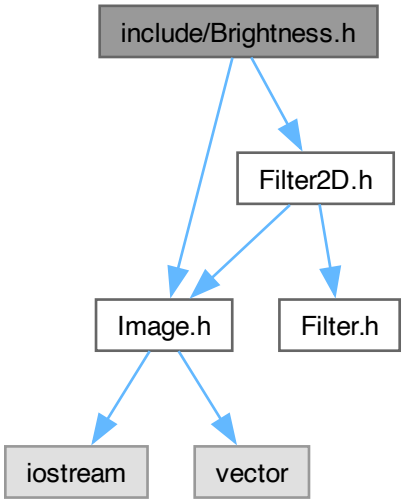
```
00001
00016 #ifndef ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_BLUR_H
00017 #define ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_BLUR_H
00018
00019 #include "Filter2D.h"
00020 #include "Image.h"
00021 #include "Kernel.h"
00022
00029 class Blur : public Filter2D {
00030
00031 public:
00036     explicit Blur(Kernel& kernel);
00037
00043     void apply(Image& image) override;
00044
00049     const Kernel& getKernel() const;
00050
00051 private:
00052     Kernel kernel;
00053 };
00054
00055 #endif //ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_BLUR_H
```

8.3 include/Brightness.h File Reference

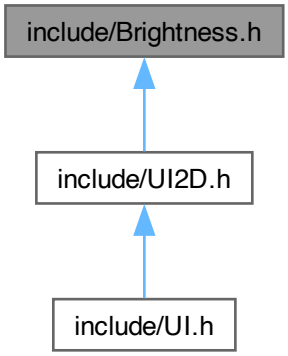
Provides the Brightness class for adjusting the brightness of images.

```
#include "Image.h"
#include "Filter2D.h"
```

Include dependency graph for Brightness.h:



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



Classes

- class Brightness

  Implements brightness adjustment for images.

8.3.1 Detailed Description

Provides the Brightness class for adjusting the brightness of images.

Author

Jing-Han Huang (edsml-jh123)

This file includes the declaration of the Brightness class, which inherits from Filter2D. It is designed to adjust the brightness of an image by modifying the pixel values across all channels, except for the alpha channel in RGBA images.

Copyright

8.4 Brightness.h

Go to the documentation of this file.

```
00001
00019 #ifndef BRIGHTNESS_H
00020 #define BRIGHTNESS_H
00021
00022 #include "Image.h"
00023 #include "Filter2D.h"
00024
00032 class Brightness : Filter2D {
00033 public:
00040     Brightness(int brightnessValue);
00050     void apply(Image& image);
00051
00052 private:
00053     int brightnessValue;
00054 };
00055
00056 #endif // BRIGHTNESS_H
```

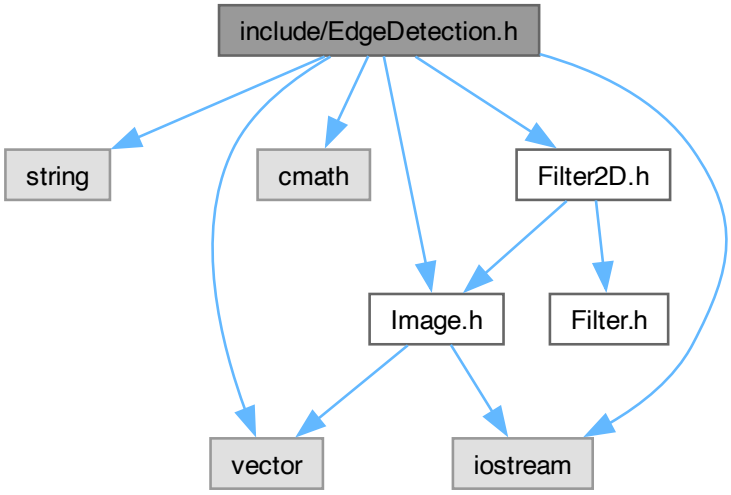
8.5 include/EdgeDetection.h File Reference

Provides the EdgeDetection class for applying edge detection algorithms on images.

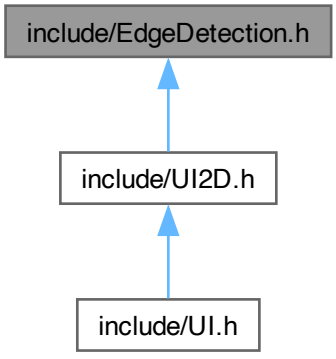
```
#include <string>
#include <vector>
#include <cmath>
#include <iostream>
#include "Image.h"
```

```
#include "Filter2D.h"
```

Include dependency graph for EdgeDetection.h:



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



Classes

- class EdgeDetection

  Implements edge detection algorithms for images.

Enumerations

- enum class EdgeDetectionAlgorithm { **Sobel** , **Prewitt** , **Scharr** , **Roberts** }

  Enumerates the supported edge detection algorithms.

8.5.1 Detailed Description

Provides the EdgeDetection class for applying edge detection algorithms on images.

Author

Yongwen Chen (acse-yc3321), Ning Guan (edsml-ng323)

This file includes the definition of the EdgeDetection class, which inherits from Filter2D. It supports various edge detection algorithms such as Sobel, Prewitt, Scharr, and Roberts. The class allows for the flexible application of these algorithms to an Image object to highlight edges and contours.

8.6 EdgeDetection.h

Go to the documentation of this file.

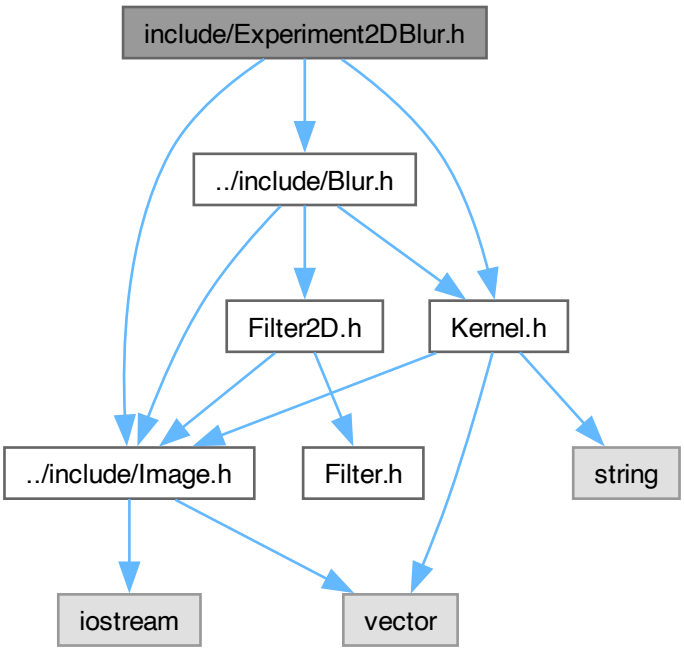
```
00001
00019 #ifndef ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_EdgeDetection_H
00020 #define ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_EdgeDetection_H
00021
00022 #include <string>
00023 #include <vector>
00024 #include <cmath>
00025 #include <iostream>
00026 #include "Image.h"
00027 #include "Filter2D.h"
00028
00033 enum class EdgeDetectionAlgorithm {
00034     Sobel,
00035     Prewitt,
00036     Scharr,
00037     Roberts
00038 };
00039
00047 class EdgeDetection : Filter2D {
00048 public:
00054     EdgeDetection(EdgeDetectionAlgorithm algorithm = EdgeDetectionAlgorithm::Sobel);
00063     void apply(Image& image);
00064
00065 private:
00066     int width;
00067     int height;
00068     std::vector<unsigned char> img_data;
00069     EdgeDetectionAlgorithm algorithm;
00070
00071     // Sobel operators
00072     const int sobel_x[3][3] = {{-1, 0, 1}, {-2, 0, 2}, {-1, 0, 1}};
00073     const int sobel_y[3][3] = {{-1, -2, -1}, {0, 0, 0}, {1, 2, 1}};
00074
00075     // Prewitt operators
00076     const int prewitt_x[3][3] = {{-1, 0, 1}, {-1, 0, 1}, {-1, 0, 1}};
00077     const int prewitt_y[3][3] = {{-1, -1, -1}, {0, 0, 0}, {1, 1, 1}};
00078
00079     // Scharr operators
00080     const int scharr_x[3][3] = {{-3, 0, 3}, {-10, 0, 10}, {-3, 0, 3}};
00081     const int scharr_y[3][3] = {{-3, -10, -3}, {0, 0, 0}, {3, 10, 3}};
00082
00083     // Roberts' Cross operators
00084     const int roberts_x[2][2] = {{1, 0}, {0, -1}};
00085     const int roberts_y[2][2] = {{0, 1}, {-1, 0}};
00086
00098     void applyKernel(Image& image, const int kernel_x[][3], const int kernel_y[][3], int kernelSize);
00099
00100 };
00101
00102 #endif
```

8.7 include/Experiment2DBlur.h File Reference

The Experiment2DBlur class is designed to run a series of experiments applying a 2D blur effect to an image with varying kernel sizes and types.

```
#include "../include/Image.h"
#include "../include/Blur.h"
#include "../include/Kernel.h"
```

Include dependency graph for Experiment2DBlur.h:



Classes

- class Experiment2DBlur

  Class for running multiple blurring experiments on an image.

8.7.1 Detailed Description

The Experiment2DBlur class is designed to run a series of experiments applying a 2D blur effect to an image with varying kernel sizes and types.

Author

Daniel Seal (edsml-ds423)

The purpose of the Experiment2DBlur class is to facilitate running multiple blur filter tests on an image (for optimisation purposes).

Copyright

8.8 Experiment2DBlur.h

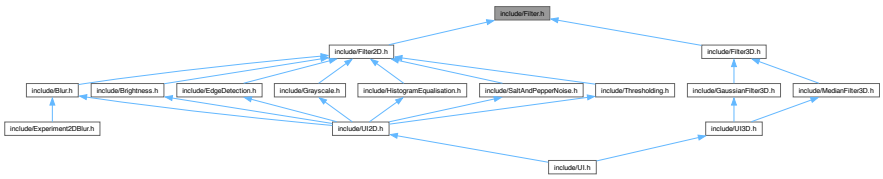
Go to the documentation of this file.

```
00001
00019 #ifndef ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_EXPERIMENT2DBLUR_H
00020 #define ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_EXPERIMENT2DBLUR_H
00021
00022 #include "../include/Image.h"
00023 #include "../include/Blur.h"
00024 #include "../include/Kernel.h"
00025
00033 class Experiment2DBlur {
00034 public:
00044     Experiment2DBlur(int& numExperiments, std::string& imgFilePath,
00045                      int& initKernelSize, std::string& kernelType,
00046                      int& kernelSizeJumps, double& sigma);
00047
00049     void runNExperiments();
00050
00055     const int& getNumExperiments() const;
00056
00057 private:
00058     int initKernelSize, numExperiments, kernelSizeJumps;
00059     std::string imgFilePath, kernelType;
00060     double sigma;
00061
00066     void runExperiment(int& kernelSize_); // keep private as wrapped in runNExperiments(). Allows it
      to also access sourceMesh directly.
00067 };
00068
00069 #endif //ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_EXPERIMENT2DBLUR_H
```

8.9 include/Filter.h File Reference

The Filter class is an abstract base class for image/volume filters.

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



Classes

- class Filter< T >

  Base class for different types of filters.

8.9.1 Detailed Description

The Filter class is an abstract base class for image/volume filters.

Author

Daniel Seal (edsml-ds423)

This file defines the Filter class, which is an abstract base class for different types of filters. It provides a common interface and foundational functionality for various filtering operations. Derived classes must implement the apply() method to define the specific filtering operation for the corresponding data type.

Copyright

8.10 Filter.h

Go to the documentation of this file.

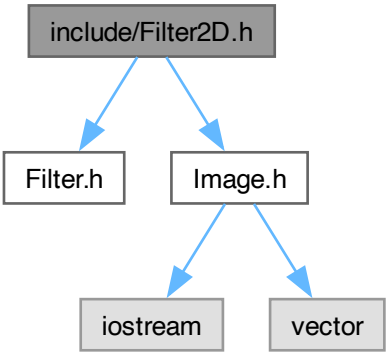
```
00001
00022 #ifndef ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_FILTER_H
00023 #define ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_FILTER_H
00024
00033 template <typename T>
00034 class Filter {
00035 public:
00037     Filter();
00045     virtual void apply(T& data) = 0;
00046
00048     virtual ~Filter() = default;
00049 };
00050
00051 #endif //ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_FILTER_H
```

8.11 include/Filter2D.h File Reference

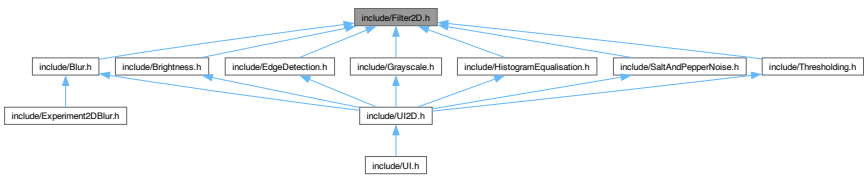
The Filter2D class is an abstract base class for 2D image filters.

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

Include dependency graph for Filter2D.h:



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



Classes

- class Filter2D

  An abstract base class for 2D image filters. Class derived from Filter.

8.11.1 Detailed Description

The Filter2D class is an abstract base class for 2D image filters.

Author

Zeqi Li (acse-zl123)

Copyright

Xavier-Initialization (2024) Daniel Seal (edsml-ds423) Yongwen Chen (acse-yc3321) Zeqi Li (acse-zl123) Jing-Han Huang (edsml-jh123) Wenbo Yu (acse-wy1223) Ning Guan (edsml-ng323)

8.12 Filter2D.h

Go to the documentation of this file.

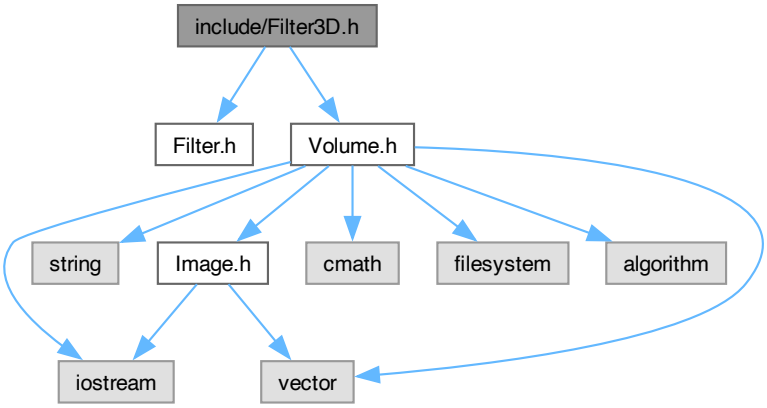
```
00001
00016 #ifndef ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_FILTER2D_H
00017 #define ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_FILTER2D_H
00018
00019 #include "Filter.h"
00020 #include "Image.h"
00021
00029 class Filter2D: Filter<Image> {
00030
00031 public:
00033     Filter2D();
00034     ~Filter2D() override;
00041     void apply(Image& image) override = 0; // abstract class.
00042 };
00043
00044
00045 #endif //ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_FILTER2D_H
```

8.13 include/Filter3D.h File Reference

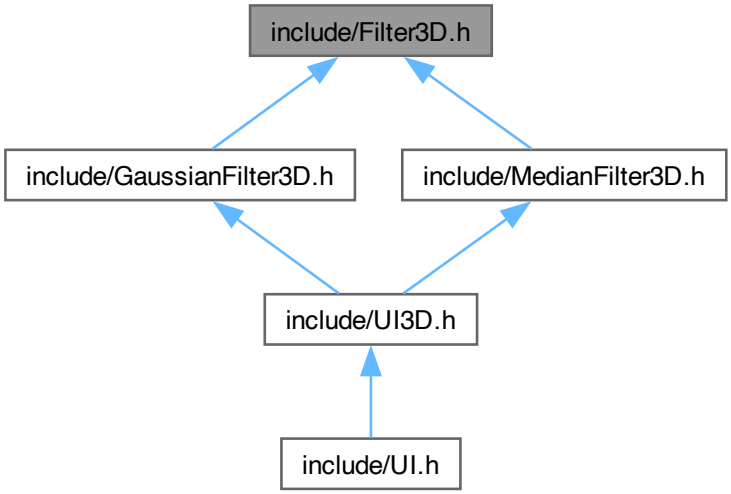
The Filter3D class is an abstract base class for 3D volume filters.

```
#include "Filter.h"
#include "Volume.h"
```

Include dependency graph for Filter3D.h:



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



Classes

- class Filter3D

  An abstract base class for 3D volume filters. Class derived from Filter.

8.13.1 Detailed Description

The Filter3D class is an abstract base class for 3D volume filters.

Author

Wenbo Yu (acse-wy1223)

Copyright

8.14 Filter3D.h

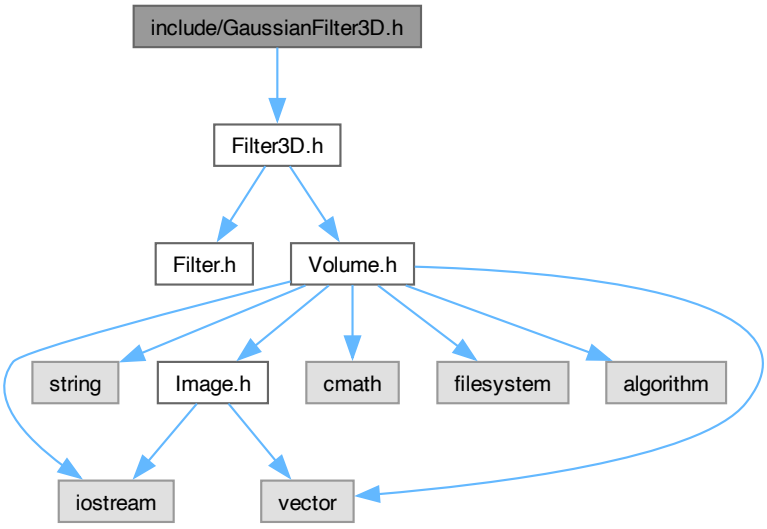
Go to the documentation of this file.

```
00001
00016 #ifndef ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_FILTER3D_H
00017 #define ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_FILTER3D_H
00018
00019 #include "Filter.h"
00020 #include "Volume.h"
00021
00030 class Filter3D: Filter<Volume> {
00031
00032 public:
00034        Filter3D ();
00036        ~Filter3D();
00037
00045         void apply(Volume& volume) override = 0; // apply filter
00046
00053         virtual void info() = 0; // display information
00054 };
00055
00056 #endif //ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_FILTER3D_H
```

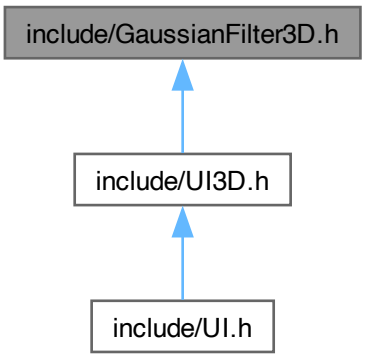
8.15 include/GaussianFilter3D.h File Reference

Defines the GaussianFilter3D class for applying a 3D Gaussian filter on volumetric data.

`#include "Filter3D.h"`
Include dependency graph for GaussianFilter3D.h:



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



Classes

- class GaussianFilter3D

  Implements a 3D Gaussian filter on volumetric data.

8.15.1 Detailed Description

Defines the GaussianFilter3D class for applying a 3D Gaussian filter on volumetric data.

Author

Wenbo Yu (acse-wy1223)

The GaussianFilter3D class extends from Filter3D and implements the application of a Gaussian blur to a 3-dimensional volume. This is commonly used for noise reduction and smoothing of volumetric data.

Copyright

8.16 GaussianFilter3D.h

Go to the documentation of this file.

```
00001
00019 #ifndef GAUSSIANFILTER3D_H
00020 #define GAUSSIANFILTER3D_H
00021 #include "Filter3D.h"
00022
00032 class GaussianFilter3D: Filter3D {
00033
00034 private:
00035     int filterSize;
```

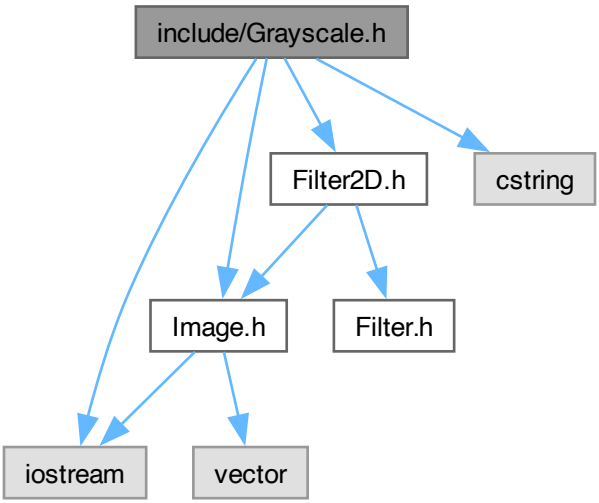
```
00036     double sigma;
00037
00038 public:
00039
00044     GaussianFilter3D();
00050     GaussianFilter3D(int filterSize, double sigma);
00051
00055     ~GaussianFilter3D();
00056
00062     void apply(Volume& volume) override;
00063
00068     void info() override;
00069
00076     void setFilter(int filterSize, double sigma);
00077 };
00078 #endif //GAUSSIANFILTER3D_H
```

8.17 include/Grayscale.h File Reference

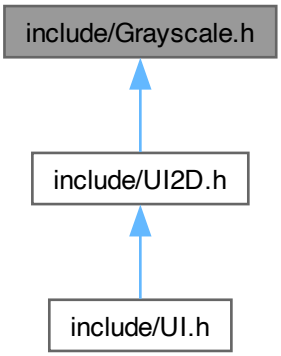
Provides the Grayscale class for converting images to grayscale.

```
#include "Image.h"
#include "Filter2D.h"
#include <cstring>
#include <iostream>
```

Include dependency graph for Grayscale.h:



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



Classes

- class Grayscale

  Implements image grayscaling.

8.17.1 Detailed Description

Provides the Grayscale class for converting images to grayscale.

Author

Yongwen Chen (acse-yc3321)

This file includes the definition of the Grayscale class, which inherits from Filter2D. It is designed to apply a grayscale filter to an image, effectively transforming a color image into grayscale by calculating luminance.

Copyright

8.18 Grayscale.h

Go to the documentation of this file.

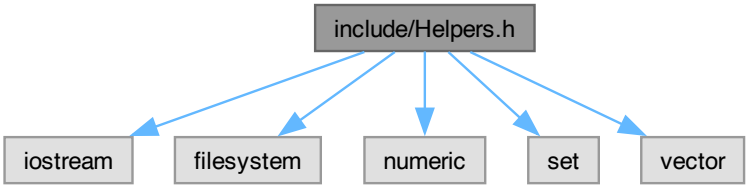
```
00001
00019 #ifndef GRAYSCALE_IMAGE_H
00020 #define GRAYSCALE_IMAGE_H
00021
00022 #include "Image.h"
00023 #include "Filter2D.h"
00024 #include <cstring>
00025 #include <iostream>
00026
00034 class Grayscale : Filter2D{
00035 public:
00041     Grayscale() = default;
00050     void apply(Image& image);
00051 };
00052
00053 #endif
```

8.19 include/Helpers.h File Reference

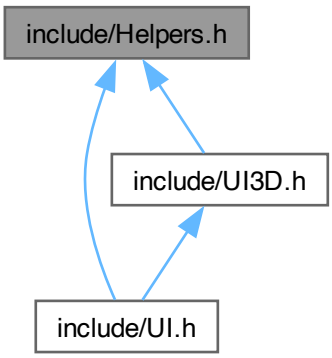
Declaration of utility functions for various simple operations.

```
#include <iostream>
#include <filesystem>
#include <numeric>
#include <set>
#include <vector>
```

Include dependency graph for Helpers.h:



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



Namespaces

- namespace helpers

  Namespace containing utility functions.

Enumerations

- enum class Align { **Left** , **Center** , **Right** }

  Alignment options for text.

Functions

- void helpers::print (const std::string &message)

  Print a message to standard output.
- std::string helpers::centered (const std::string &, int)

  Center-aligns a string within a given width.
- std::string helpers::formatPrice (double)

  Formats a double value as a price string.
- void **helpers::clearScreen** ()

  Clears the console screen.
- void **helpers::pause** ()

  Pauses program execution and waits for the user to press Enter.
- void helpers::printLine (int len=80)

  Prints a horizontal line of a specified length.
- void helpers::printTitle (const std::string &, int, Align, int)

  Prints a title with specified alignment and optional border lines.
- int helpers::getInput (int &)

  Gets integer input from the user between 0 and 100.
- void **helpers::printSigniture** ()

  Prints signature details for the program or script.
- void helpers::ensure_directory_exists (const fs::path &path, bool verbose=false)

  Ensures that a specified directory exists, creating it if necessary.
- std::vector< int > helpers::get_vector (int size)

  Generates and returns a vector filled with a sequence of integers.
- std::set< int > helpers::get_set (int size)

  Generates and returns a set filled with a sequence of integers.

8.19.1 Detailed Description

Declaration of utility functions for various simple operations.

Author

Wenbo Yu (acse-wy1223)

This file declares a series of utility functions in the helpers namespace. These functions include general-purpose operations such as printing messages, formatting prices, clearing the screen, pausing the program, and more advanced operations like ensuring directory existence.

Copyright

8.20 Helpers.h

Go to the documentation of this file.

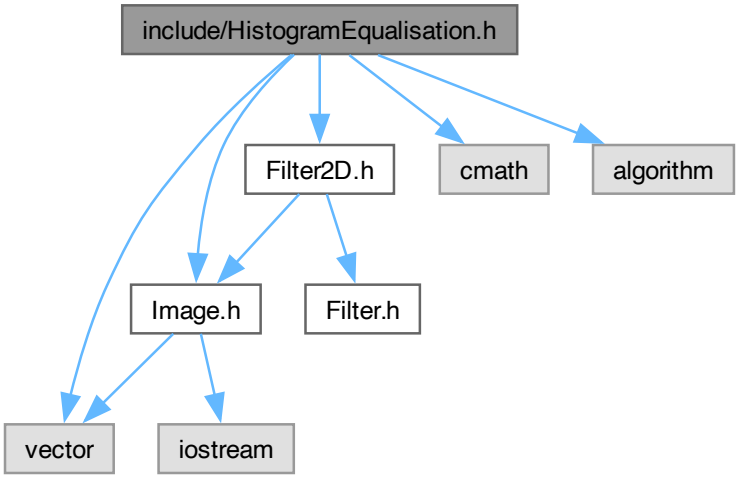
```
00001
00020 #ifndef HELPERS_H
00021 #define HELPERS_H
00022 #include <iostream>
00023 #include <filesystem>
00024 #include <numeric>
00025 #include <set>
00026 #include <vector>
00027 namespace fs = std::filesystem;
00028
00034 enum class Align { Left, Center, Right };
00035
00041 namespace helpers {
00042
00047     void print(const std::string& message);
00048
00055     std::string centered(const std::string&, int);
00056
00062     std::string formatPrice(double);
00063
00067     void clearScreen();
00068
00072     void pause();
00073
00078     void printLine(int len = 80);
00079
00087     void printTitle(const std::string&, int, Align, int);
00088
00094     int getInput(int&);
00095
00099     void printSigniture();
00100
00106     void ensure_directory_exists(const fs::path& path, bool verbose = false);
00107
00113     std::vector<int> get_vector(int size);
00114
00120     std::set<int> get_set(int size);
00121 }
00122
00123 #endif // HELPERS_H
```

8.21 include/HistogramEqualisation.h File Reference

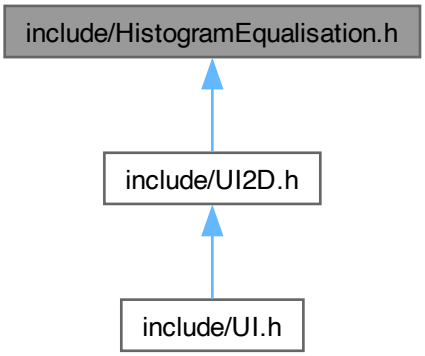
Provides the HistogramEqualisation class for histogram equalization on images.

```
#include "Image.h"
#include "Filter2D.h"
#include <vector>
#include <cmath>
#include <algorithm>
```

Include dependency graph for HistogramEqualisation.h:



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



Classes

- class HistogramEqualisation

  Implements histogram equalization for image contrast enhancement.

8.21.1 Detailed Description

Provides the HistogramEqualisation class for histogram equalization on images.

Author

Zeqi Li (acse-zl123)

This header defines the HistogramEqualisation class which inherits from Filter2D. It implements histogram equalization for enhancing the contrast of images. The class supports both gray and RGB in HSV/HSL color spaces for equalization, allowing for flexible image processing according to the color model specified at instantiation.

Copyright

8.22 HistogramEqualisation.h

Go to the documentation of this file.

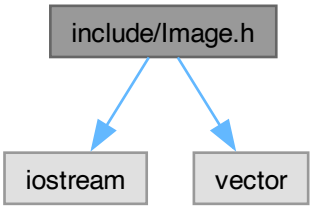
```
00001
00019 #ifndef ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_HISTOGRAM_EQUALISATION_H
00020 #define ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_HISTOGRAM_EQUALISATION_H
00021
00022 #include "Image.h"
00023 #include "Filter2D.h"
00024 #include <vector>
00025 #include <cmath>
00026 #include <algorithm>
00027
00036 class HistogramEqualisation : Filter2D {
00037 public:
00047     HistogramEqualisation(bool isHSV);
00056     void apply(Image& image);
00057 private:
00058     bool isHSV;
00059
00070     void RGBToHSV(double r, double g, double b, double &h, double &s, double &v);
00071
00082     void HSVToRGB(double h, double s, double v, double &r, double &g, double &b);
00083
00094     void RGBToHSL(double r, double g, double b, double &h, double &s, double &l);
00095
00106     void HSLToRGB(double h, double s, double l, double &r, double &g, double &b);
00107
00114     void equalise(unsigned char* data, int imageSize);
00115
00122     void equalise(std::vector<double>& vChannel, int vSize);
00123 };
00124
00125
00126 #endif //ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_HISTOGRAM_EQUALISATION_H
```

8.23 include/Image.h File Reference

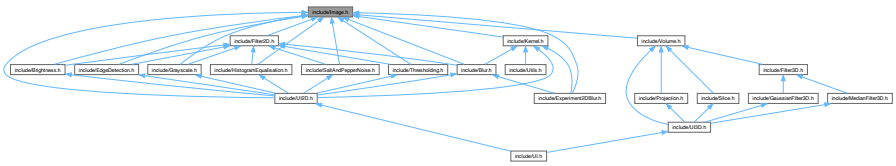
The Image class is used for loading, manipulating, and saving images.

```
#include <iostream>
#include <vector>
```

Include dependency graph for Image.h:



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



Classes

- class Image

  Class for handling image loading, processing, and saving. The Image class used the STB image libraries are used to load, manipulate, and save images.

8.23.1 Detailed Description

The Image class is used for loading, manipulating, and saving images.

Author

Daniel Seal (edsml-ds423)

Copyright

8.24 Image.h

Go to the documentation of this file.

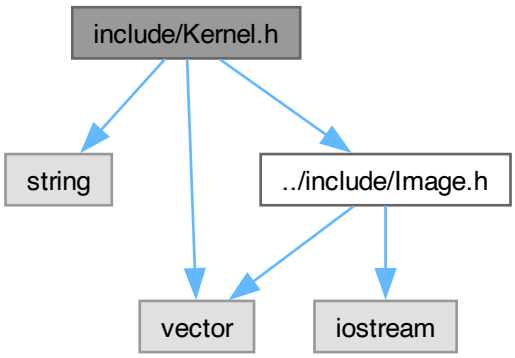
```
00001
00016 #ifndef ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_IMAGE_H
00017 #define ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_IMAGE_H
00018 #include <iostream>
00019 #include <vector>
00020
00026 class Image {
00027 public:
00029     friend std::ostream& operator « (std::ostream& s, Image const& m);
00030
00035     explicit Image(std::string &filepath);
00036
00042     Image(std::string &filepath, int& c);
00043
00051     Image(std::string &filepath, int& h, int& w, int& c);
00053     Image();
00055     Image(const Image& imageCopy);
00057     ~Image();
00058
00066     int& operator()(int x, int y, int z);
00067
00072     void readImage();
00074     void createImageData();
00076     void createPaddedImageData(int& pad);
00081     void outputImage(const std::string& outputFilePath);
00082
00083     // setters.
00084     void setPixel(int &h, int &w, int& c, int& value);
00085     void setHeight(int &h);
00086     void setWidth(int &w);
00087     void setDataPtr(unsigned char*& p);
00088     void setChannels(int& c);
00089     void setDesiredChannels(int& dc);
00090
00091     // getters.
00092     unsigned char& getPixel(int& h, int& w, int& c);
00093     const int& getHeight() const;
00094     const int& getWidth() const;
00095     const int& getRawHeight() const;
00096     const int& getRawWidth() const;
00097     const int& getChannels() const;
00098     const int& getDesiredChannels() const;
00099     unsigned char*& getDataPtr();
00100
00101 private:
00102     std::string filepath;
00103     unsigned char* dataPtr;
00104     std::vector<std::vector<std::vector<int»> data;
00105     int width;
00106     int height;
00107     int channels;
00108     int rawWidth;
00109     int rawHeight;
00110     int desiredChannels;
00111 };
00112
00113 #endif //ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_IMAGE_H
```

8.25 include/Kernel.h File Reference

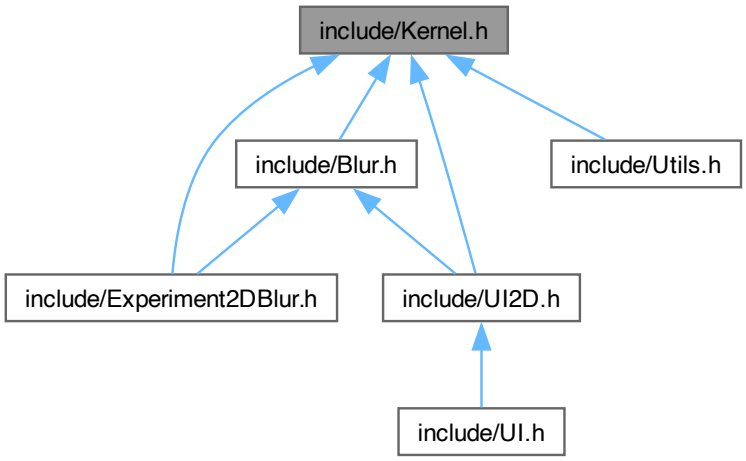
The Kernel class is used for instantiating 2D kernels and applying them on images.

```
#include <string>
#include <vector>
#include "../include/Image.h"
```

Include dependency graph for Kernel.h:



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



Classes

- class Kernel

  Class for instantiating 2D kernels and applying them on images.

Enumerations

- enum class KernelType { **None** , **Gaussian** , **Box** , **Median** }

  Enumerates types of kernels that can be applied to images.

8.25.1 Detailed Description

The Kernel class is used for instantiating 2D kernels and applying them on images.

Author

Daniel Seal (edsml-ds423)

Copyright

8.26 Kernel.h

Go to the documentation of this file.

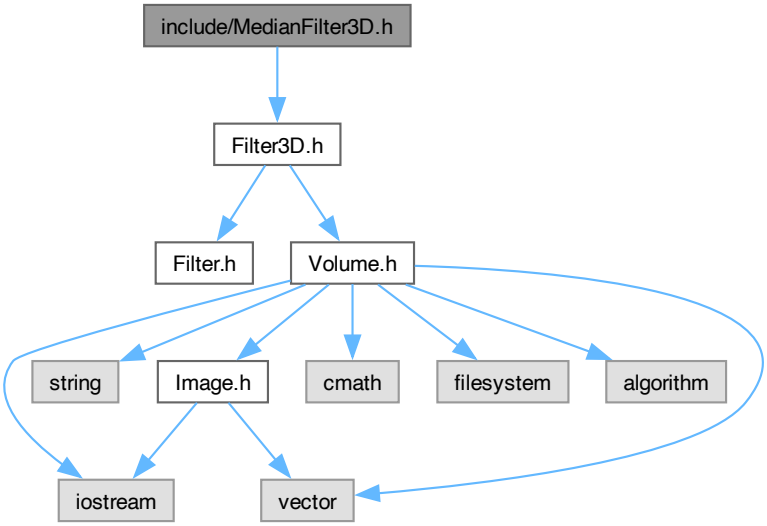
```
00001
00016 #ifndef ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_KERNEL_H
00017 #define ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_KERNEL_H
00018
00019 #include <string>
00020 #include <vector>
00021 #include "../include/Image.h"
00022
00027 enum class KernelType {
00028     None,
00029     Gaussian,
00030     Box,
00031     Median
00032 };
00033
00034
00041 class Kernel {
00042 public:
00044     Kernel();
00045
00051     Kernel(int& kernelSize, std::string& kernelType);
00052
00059     Kernel(int& kernelSize, std::string& kernelType, double& sigma);
00060
00070     static std::vector<double> applyKernel(Image& image, Kernel& k, int& iPoint, int& jPoint, int&
      zPoint);
00071
00077     static KernelType stringToKernelType(std::string& str);
00078
00079     // setters.
00080     void setPadding(int& p);
00081     void setKernel(std::vector<double>& k);
00082
00083     // getters.
00084     const std::vector<double>& getKernel() const;
00085     const int& getKernelSize() const;
00086     const int& getPadding() const;
00087     const double& getSigma() const;
00088     const KernelType& getKernelType() const;
00089
00090 private:
00091     int kernelSize;
00092     KernelType kernelType;
00093     double sigma;
00094     int padding;
00095     std::vector<double> kernel;
00096
00097     // private methods. Only executed on instantiation.
00099     void calculatePadding();
00101     void generateKernel();
00102 };
00103
00104 #endif //ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_KERNEL_H
```

8.27 include/MedianFilter3D.h File Reference

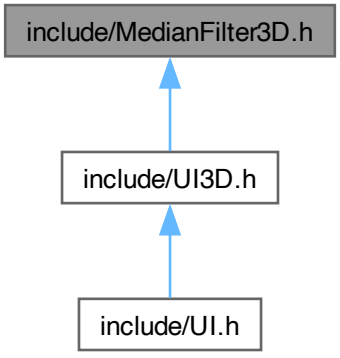
Defines the MedianFilter3D class for applying a median filter to volumetric data.

```
#include "Filter3D.h"
```

Include dependency graph for MedianFilter3D.h:



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



Classes

- class MedianFilter3D

  Implements median filtering for volumetric data.

8.27.1 Detailed Description

Defines the MedianFilter3D class for applying a median filter to volumetric data.

Author

Wenbo Yu (acse-wy1223)

The MedianFilter3D class is derived from Filter3D and is designed to apply a median filtering process to 3-dimensional volumetric data. Median filtering is a non-linear process useful in reducing noise while preserving edges by selecting the median pixel value from the neighborhood of each pixel in the volume.

Copyright

8.28 MedianFilter3D.h

Go to the documentation of this file.

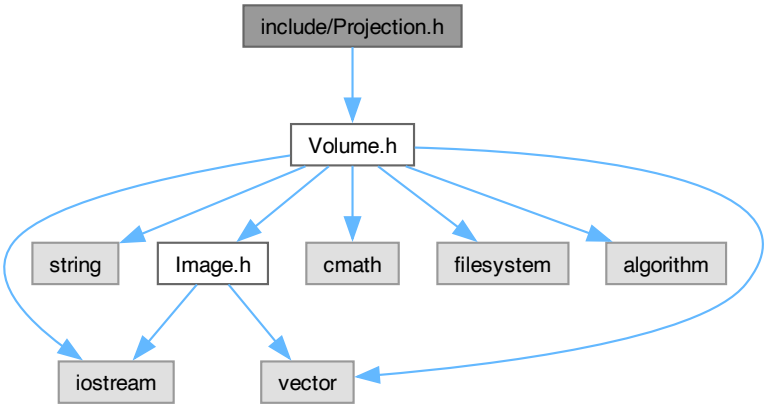
```
00001
00021 #ifndef MEDIANFILTER3D_H
00022 #define MEDIANFILTER3D_H
00023 #include "Filter3D.h"
00024
00033 class MedianFilter3D: Filter3D {
00034
00035 private:
00036     int filterSize;
00037 public:
00038
00042     MedianFilter3D();
00043
00048     MedianFilter3D(int filterSize);
00049
00053     ~MedianFilter3D();
00054
00059     void apply(Volume& volume) override;
00060
00064     void info() override;
00065
00070     void setFilter(int filterSize);
00071
00085     unsigned char initHistogramAndFindMedian(const std::vector<unsigned char*>& data, int s, int y,
      int x, int c, int width, int height, int slices, std::vector<int>& histogram);
00086 };
00087 #endif //MEDIANFILTER3D_H
```

8.29 include/Projection.h File Reference

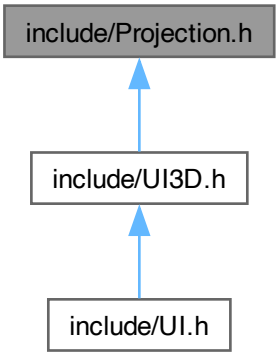
Declaration of the Projection class for computing and applying projection techniques to 3D volume data.

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

Include dependency graph for Projection.h:



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



Classes

- class Projection

  Class to perform various projection operations on volume data.

8.29.1 Detailed Description

Declaration of the Projection class for computing and applying projection techniques to 3D volume data.

Author

Wenbo Yu (acse-wy1223)

The Projection class provides methods to apply different types of projections like Maximum Intensity Projection (MIP), Minimum Intensity Projection (MinIP), and Average Intensity Projection (AIP) to a given Volume. These methods facilitate the visualization of 3D volume data in 2D form.

Copyright

8.30 Projection.h

Go to the documentation of this file.

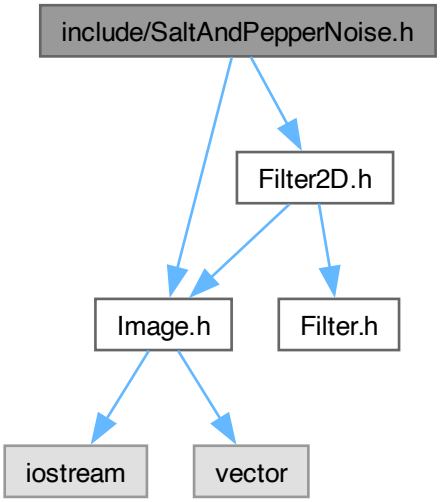
```
00001
00019 #ifndef ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_PROJECTION_H
00020 #define ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_PROJECTION_H
00021
00022 #include "Volume.h"
00023
00033 class Projection {
00034 public:
00035
00039     Projection() {};
00040
00044     ~Projection() {};
00045
00053     void applyMIP(const Volume& volume, const std::string& outputPath, int startSlice = -1, int
      endSlice = -1);
00054
00062     void applyMinIP(const Volume& volume, const std::string& outputPath, int startSlice = -1, int
      endSlice = -1);
00063
00071     void applyAIP(const Volume& volume, const std::string& outputPath, int startSlice = -1, int
      endSlice = -1);
00072 };
00073
00074 #endif //ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_PROJECTION_H
```

8.31 include/SaltAndPepperNoise.h File Reference

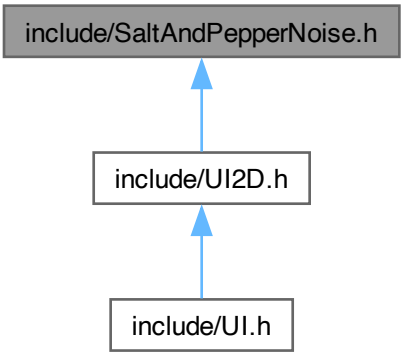
Provides the SaltandPepperNoise class to add salt-and-pepper noise to images.

```
#include "Image.h"
#include "Filter2D.h"
```

Include dependency graph for SaltAndPepperNoise.h:



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



Classes

- class SaltAndPepperNoise

  Implements salt-and-pepper noise addition to images.

8.31.1 Detailed Description

Provides the SaltandPepperNoise class to add salt-and-pepper noise to images.

Author

Jing-Han Huang (edsml-jh123)

This file includes the declaration of the SaltandPepperNoise class, which inherits from Filter2D. It is designed to introduce salt-and-pepper noise (random occurrences of black and white pixels) to an image, based on a specified noise percentage.

Copyright

8.32 SaltAndPepperNoise.h

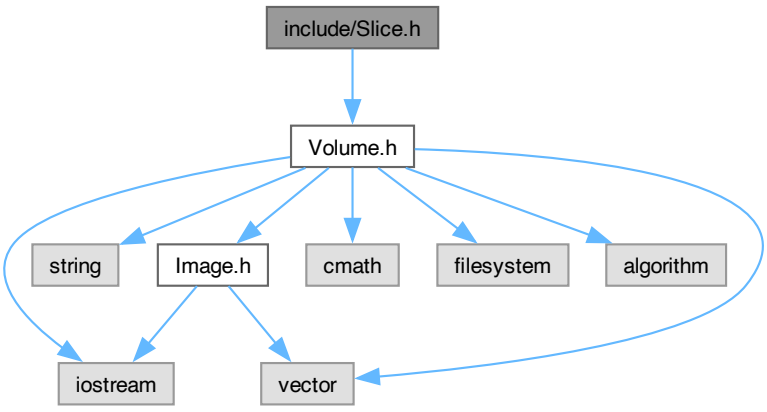
Go to the documentation of this file.

```
00001
00019 #ifndef SALTANDPEPPERNOISE_H
00020 #define SALTANDPEPPERNOISE_H
00021
00022 #include "Image.h"
00023 #include "Filter2D.h"
00024
00033 class SaltAndPepperNoise : Filter2D {
00034 public:
00041     SaltAndPepperNoise(double noisePercentage);
00050     void apply(Image& image);
00051
00052 private:
00053     double noisePercentage;
00054 };
00055
00056 #endif // SALTANDPEPPERNOISE_H
```

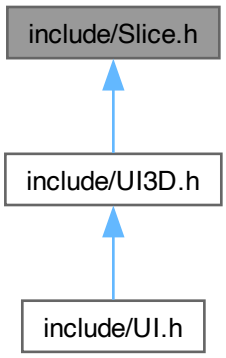
8.33 include/Slice.h File Reference

Declaration of the Slice class for extracting and saving 2D slices from 3D volume data.

`#include "Volume.h"`
Include dependency graph for Slice.h:



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



Classes

- class Slice

  Class for extracting 2D slices from a 3D volume.

8.33.1 Detailed Description

Declaration of the Slice class for extracting and saving 2D slices from 3D volume data.

Author

Wenbo Yu (acse-wy1223)

The Slice class includes methods for extracting YZ and XZ slices from a Volume object. These methods allow the slices to be extracted and saved, facilitating the analysis and visualization of specific cross-sections of the volume data.

Copyright

Xavier-Initialization (2024) Daniel Seal (edsml-ds423) Yongwen Chen (acse-yc3321) Zeqi Li (acse-zl123) Jing-Han Huang (edsml-jh123) Wenbo Yu (acse-wy1223) Ning Guan (edsml-ng323)

8.34 Slice.h

Go to the documentation of this file.

```
00001
00019 #ifndef ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_SLICE_H
00020 #define ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_SLICE_H
00021
00022 #include "Volume.h"
00023
00032 class Slice {
00033 public:
```

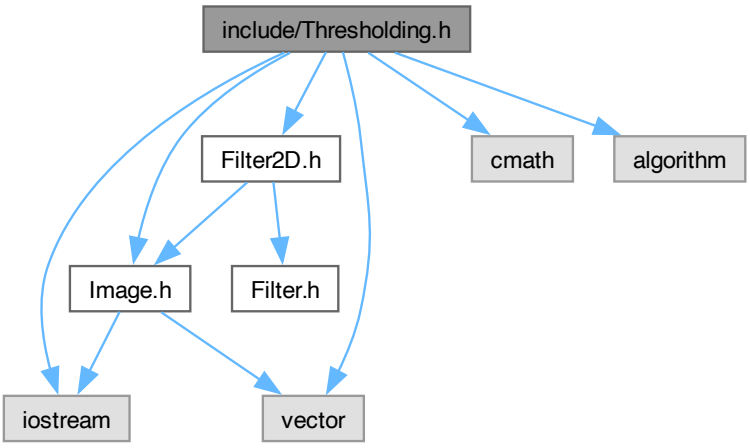
```
00034
00038     Slice() {};
00039
00043     ~Slice() {};
00044
00051     void dosliceYZ(const Volume& volume, int x, const std::string& outputPath);
00052
00059     static void sliceYZ(const Volume& volume, int x, const std::string& outputPath) {
00060
00061         int height = volume.getHeight();
00062         int width = volume.getWidth();
00063         int slices = volume.getSlices();
00064         std::vector<unsigned char*>  data = volume.getData();
00065
00066         if (x < 1 || x > width) {
00067             std::cerr « "Invalid x coordinate for slicing." « std::endl;
00068             return;
00069         }
00070         x -= 1; // Adjust for 0-based indexing
00071
00072         unsigned char* slice = new unsigned char[height * slices * 3];
00073         for (int s = 0; s < slices; ++s) {
00074             for (int y = 0; y < height; ++y) {
00075                 for (int c = 0; c < 3; ++c) {
00076                     slice[(y * slices + s) * 3 + c] = data[s][(y * width + x) * 3 + c];
00077                 }
00078             }
00079         }
00080         //stbi_write_png(outputPath.c_str(), slices, height, 3, slice, slices * 3);
00081         delete[] slice;
00082     }
00083
00090     void dosliceXZ(const Volume& volume, int y, const std::string& outputPath);
00091
00098     static void sliceXZ(const Volume& volume, int y, const std::string& outputPath) {
00099         int height = volume.getHeight();
00100         int width = volume.getWidth();
00101         int slices = volume.getSlices();
00102         std::vector<unsigned char*>  data = volume.getData();
00103
00104         if (y < 1 || y > height) {
00105             std::cerr « "Invalid y coordinate for slicing." « std::endl;
00106             return;
00107         }
00108         y -= 1; // Adjust for 0-based indexing
00109
00110         unsigned char* slice = new unsigned char[width * slices * 3];
00111         for (int s = 0; s < slices; ++s) {
00112             for (int x = 0; x < width; ++x) {
00113                 for (int c = 0; c < 3; ++c) {
00114                     slice[(s * width + x) * 3 + c] = data[s][(y * width + x) * 3 + c];
00115                 }
00116             }
00117         }
00118         //stbi_write_png(outputPath.c_str(), width, slices, 3, slice, width * 3);
00119         delete[] slice;
00120     }
00121 };
00122
00123
00124 #endif //ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_SLICE_H
```

8.35 include/Thresholding.h File Reference

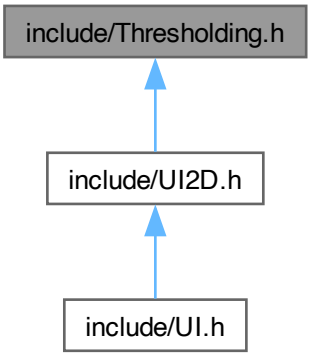
Provides the Thresholding class for applying thresholding techniques on images.

```
#include "Image.h"
#include "Filter2D.h"
#include <vector>
#include <cmath>
#include <algorithm>
#include <iostream>
```

Include dependency graph for Thresholding.h:



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



Classes

- class Thresholding

  Implements thresholding for image processing.

8.35.1 Detailed Description

Provides the Thresholding class for applying thresholding techniques on images.

Author

Zeqi Li (acse-zl123)

This file includes the declaration of the Thresholding class, which inherits from Filter2D. The class is designed to apply thresholding to an image, converting it to a binary image based on a specified threshold value. It supports both grayscale and color images, with an option to perform thresholding in the HSV/HSL color space for color images.

Copyright

8.36 Thresholding.h

Go to the documentation of this file.

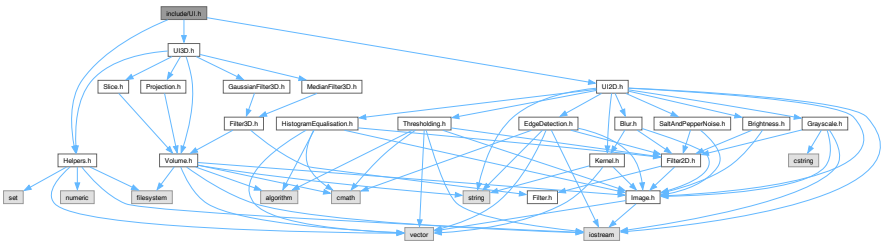
```
00001
00020 #ifndef ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_THRESHOLDING_H
00021 #define ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_THRESHOLDING_H
00022
00023 #include "Image.h"
00024 #include "Filter2D.h"
00025 #include <vector>
00026 #include <cmath>
00027 #include <algorithm>
00028 #include <iostream>
00029
00038 class Thresholding : Filter2D {
00039 public:
00046     Thresholding(double threshold, bool isHSV);
00056     void apply(Image& image);
00057 private:
00058     int threshold;
00059     bool isHSV;
00060     void RGBToHSV(double r, double g, double b, double &h, double &s, double &v);
00061     void RGBToHSL(double r, double g, double b, double &h, double &s, double &l);
00062 };
00063
00064
00065 #endif //ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_THRESHOLDING_H
```

8.37 include/UI.h File Reference

Declaration of the UI class that handles the main user interface for the application.

```
#include "UI2D.h"
#include "UI3D.h"
#include "Helpers.h"
```

Include dependency graph for UI.h:



Classes

- class UI

  Main User Interface class for handling 2D and 3D UI operations.

8.37.1 Detailed Description

Declaration of the UI class that handles the main user interface for the application.

Author

Wenbo Yu (acse-wy1223)

This file contains the declaration of the UI class which orchestrates the user interface and interaction for a console-based application that allows switching between 2D and 3D UI contexts.

Copyright

8.38 UI.h

Go to the documentation of this file.

```
00001
00019 #ifndef ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_UI_H
00020 #define ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_UI_H
00021
00022 #include "UI2D.h"
00023 #include "UI3D.h"
00024 #include "Helpers.h"
00025 using namespace helpers;
00026
00035 class UI
00036 {
00037 public:
00038
00042     UI() {};
00043
00047     ~UI() {};
00048
00052     void run() {
00053         mainMenu();
00054     }
00055
00060     int mainMenu() {
00061         while (true)
00062         {
00063             clearScreen();
00064             print("3D or 2D?");
00065             print("1. 2D");
00066             print("2. 3D");
00067             print("0. Exit");
00068             int option;
00069             std::cin » option;
00070             // Check if the input is valid
00071             if (std::cin.fail()) {
00072                 std::cin.clear();
00073                 std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00074                 print("Please enter a valid number.");
00075                 pause();
00076                 continue;
00077             }
00078             else { std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n'); }
00079             if (option == 1) {
00080                 print("Your choice is 2D");
00081                 UI2D ui2d;
```

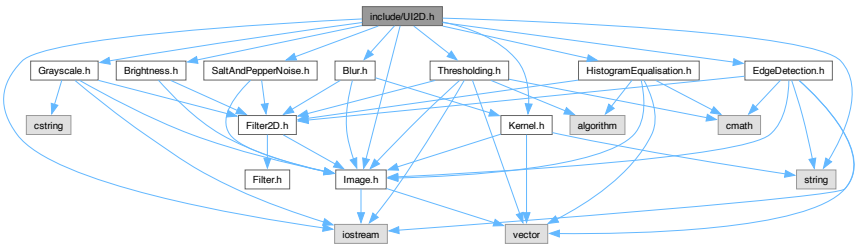
```
00082                 ui2d.run();
00083             }
00084             else if (option == 2) {
00085                 print("Your choice is 3D");
00086                 UI3D ui3d;
00087                 ui3d.run();
00088             }
00089             else if (option == 0) {
00090                 break;
00091             }
00092             else {
00093                 print("Invalid option.");
00094             }
00095         }
00096         return 0;
00097     }
00098 };
00099
00100 #endif //ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_UI_H
```

8.39 include/UI2D.h File Reference

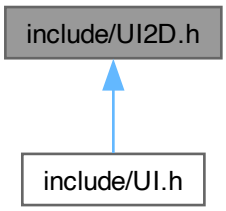
Provides the UI2D class for interacting with users.

```
#include <iostream>
#include <string>
#include "Image.h"
#include "Blur.h"
#include "Brightness.h"
#include "SaltAndPepperNoise.h"
#include "HistogramEqualisation.h"
#include "Grayscale.h"
#include "Thresholding.h"
#include "EdgeDetection.h"
#include "Kernel.h"
```

Include dependency graph for UI2D.h:



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



Classes

- class UI2D

  User interface class for 2D visualization and processing options.

8.39.1 Detailed Description

Provides the UI2D class for interacting with users.

Author

Zeqi Li (acse-zl123)

This header file defines the UI2D class, which allows users to interact with various image processing functionalities through a 2D user interface.

Copyright

Xavier-Initialization (2024) Daniel Seal (edsml-ds423) Yongwen Chen (acse-yc3321) Zeqi Li (acse-zl123) Jing-Han Huang (edsml-jh123) Wenbo Yu (acse-wy1223) Ning Guan (edsml-ng323)

8.40 UI2D.h

Go to the documentation of this file.

```
00001
00018 #ifndef ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_UI2D_H
00019 #define ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_UI2D_H
00020
00021 #include <iostream>
00022 #include <string>
00023 #include "Image.h"
00024 #include "Blur.h"
00025 #include "Brightness.h"
00026 #include "SaltAndPepperNoise.h"
00027 #include "HistogramEqualisation.h"
00028 #include "Grayscale.h"
00029 #include "Thresholding.h"
00030 #include "SaltAndPepperNoise.h"
00031 #include "EdgeDetection.h"
00032 #include "Kernel.h"
```

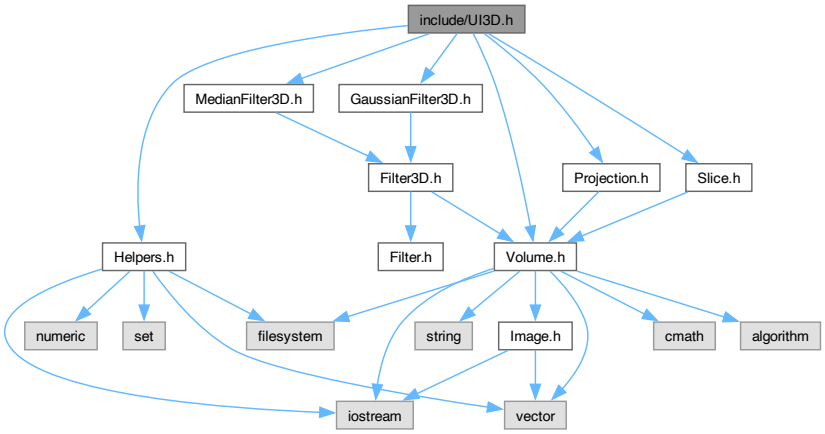
```
00033
00040 class UI2D
00041 {
00042 private:
00047     int mainMenu2D();
00048 public:
00052     UI2D() = default;
00057     int run();
00058 };
00059
00060
00061 #endif //ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_UI2D_H
```

8.41 include/UI3D.h File Reference

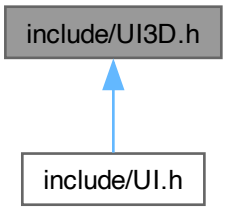
Defines the UI3D class for the 3D user interface of the application.

```
#include "Helpers.h"
#include "Volume.h"
#include "GaussianFilter3D.h"
#include "MedianFilter3D.h"
#include "Projection.h"
#include "Slice.h"
```

Include dependency graph for UI3D.h:



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



Classes

- class UI3D

  User interface class for 3D visualization and processing options.

8.41.1 Detailed Description

Defines the UI3D class for the 3D user interface of the application.

Author

Wenbo Yu (acse-wy1223)

This file contains the definition of the UI3D class, which manages the user interface for 3D visualization and processing options. It allows users to choose different operations like filtering, projection, and slicing on 3D volume data.

Copyright

8.42 UI3D.h

Go to the documentation of this file.

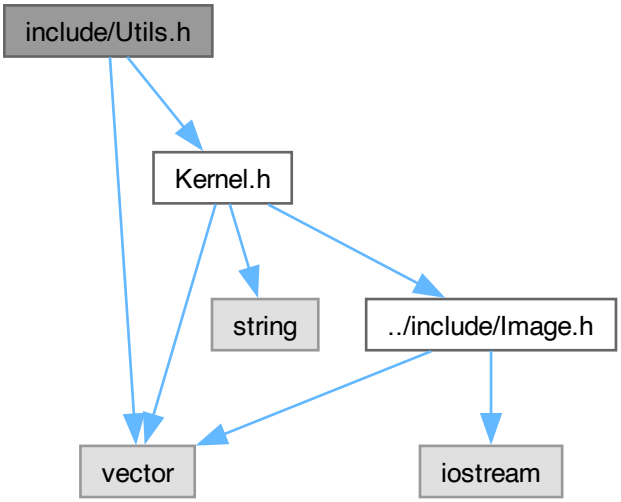
```
00001
00020 #ifndef ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_UI3D_H
00021 #define ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_UI3D_H
00022
00023 #include "Helpers.h"
00024 #include "Volume.h"
00025 #include "GaussianFilter3D.h"
00026 #include "MedianFilter3D.h"
00027 #include "Projection.h"
00028 #include "Slice.h"
00029
00030 using namespace helpers;
00031
00040 class UI3D {
00041 public:
00042
00047     int run() {
00048         int opt = mainMenu();
00049         return opt;
00050     }
00051 private:
00052     std::string inputdir;
00053     std::string outputdir;
00054
00059     int mainMenu();
00060     int enterInputPath(Volume& volume, std::string& path);
00061     std::string enterOutputPath(std::string& path);
00062     int filterMenu3D(Volume& volume, std::string& inputdir, std::string& outputdir);
00063     int gaussianFilterSize(int& filterSize, double& sigma);
00064     int gaussianSigma(int& filterSize, double& sigma);
00065     int medianFilterSize(int& filterSize);
00066     int operationMenu3D(Volume& volume, std::string& inputdir, std::string& outputdir);
00067     int projectionMenu(Volume& volume, std::string& inputdir, std::string& outputdir);
00068     int startSlice(int start, int end, int default_start, int default_end);
00069     int endSlice(int start, int end, int default_start, int default_end);
00070     int sliceMenu(Volume& volume, std::string& inputdir, std::string& outputdir);
00071     int sliceIndex(int default_index);
00072 };
00073
00074 #endif //ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_UI3D_H
```

8.43 include/Utils.h File Reference

This file contains utility functions for various operations.

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

Include dependency graph for Utils.h:



Functions

- int getVectorSum (const std::vector< double > &values)
  Calculates the sum of values in a vector.
- int getVectorMidpoint (const std::vector< double > &values)
  Calculates the midpoint value of a vector.
- int medianOfThree (std::vector< double > &vec, int &low, int &high)
  Finds the median of three values in a vector.
- int partition (std::vector< double > &vec, int &low, int &high)
  Partitions a vector around a pivot value.
- void quickSort (std::vector< double > &vec, int low, int high)
  Performs quicksort on a vector.

8.43.1 Detailed Description

This file contains utility functions for various operations.

Author

Daniel Seal (edsml-ds423)

Copyright

Xavier-Initialization (2024) Daniel Seal (edsml-ds423) Yongwen Chen (acse-yc3321) Zeqi Li (acse-zl123) Jing-Han Huang (edsml-jh123) Wenbo Yu (acse-wy1223) Ning Guan (edsml-ng323)

8.43.2 Function Documentation

8.43.2.1 getVectorMidpoint()

```
int getVectorMidpoint (
            const std::vector< double > & values )
```

Calculates the midpoint value of a vector.

Parameters

values	The input vector.

Returns

The midpoint value of the vector.

8.43.2.2 getVectorSum()

```
int getVectorSum (
            const std::vector< double > & values )
```

Calculates the sum of values in a vector.

Parameters

values	The input vector.

Returns

The sum of values in the vector.

8.43.2.3 medianOfThree()

```
int medianOfThree (
            std::vector< double > & vec,
            int & low,
            int & high )
```

Finds the median of three values in a vector.

Parameters

vec	The input vector.
low	The starting index of the subvector.
high	The ending index of the subvector.

Returns

The index of the median value.

8.43.2.4 partition()

```
int partition (
            std::vector< double > & vec,
            int & low,
            int & high )
```

Partitions a vector around a pivot value.

Parameters

vec	The input vector.
low	The starting index of the subvector.
high	The ending index of the subvector.

Returns

The index of the pivot value after partitioning.

8.43.2.5 quickSort()

```
void quickSort (
            std::vector< double > & vec,
            int low,
            int high )
```

Performs quicksort on a vector.

Parameters

vec	The input vector to be sorted.
low	The starting index of the subvector.
high	The ending index of the subvector.

8.44 Utils.h

Go to the documentation of this file.

```
00001
00016 #ifndef ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_UTILS_H
00017 #define ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_UTILS_H
00018
00019 #include <vector>
00020 #include "Kernel.h"
00021
00027 int getVectorSum(const std::vector<double>& values);
00028
```

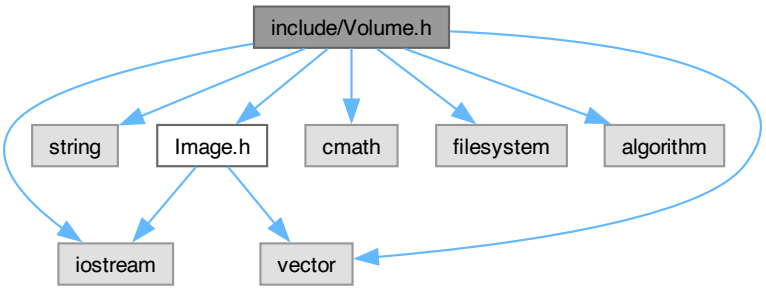
```
00034 int getVectorMidpoint(const std::vector<double>& values);
00035
00043 int medianOfThree(std::vector<double>& vec, int& low, int& high);
00044
00052 int partition(std::vector<double>& vec, int& low, int& high);
00053
00060 void quickSort(std::vector<double>& vec, int low, int high);
00061
00062 #endif //ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_UTILS_H
```

8.45 include/Volume.h File Reference

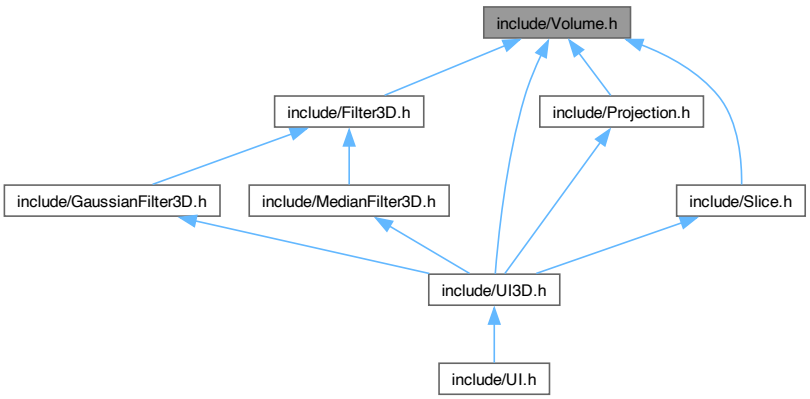
Declaration of the Volume class for managing 3D volume data.

```
#include <iostream>
#include <string>
#include <vector>
#include <cmath>
#include <filesystem>
#include <algorithm>
#include "Image.h"
```

Include dependency graph for Volume.h:



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



Classes

- class Volume

  Manages loading, saving, and manipulating 3D volume data.

8.45.1 Detailed Description

Declaration of the Volume class for managing 3D volume data.

Author

Wenbo Yu (acse-wy1223)

The Volume class provides functionalities for loading, saving, and manipulating 3D volume data from a series of images. It supports operations like reloading the original volume, applying filters, and generating or processing slices of the volume.

Copyright

Xavier-Initialization (2024) Daniel Seal (edsml-ds423) Yongwen Chen (acse-yc3321) Zeqi Li (acse-zl123) Jing-Han Huang (edsml-jh123) Wenbo Yu (acse-wy1223) Ning Guan (edsml-ng323)

8.46 Volume.h

Go to the documentation of this file.

```
00001
00020 #ifndef ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_VOLUME_H
00021 #define ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_VOLUME_H
00022
00023 #include <iostream>
00024 #include <string>
00025 #include <vector>
00026 #include <cmath>
00027 #include <filesystem>
00028 #include <algorithm>
00029 #include "Image.h"
00030
00031 namespace fs = std::filesystem;
00032
00041 class Volume {
00042 private:
00043     int width, height, slices, channels;
00044     std::vector<unsigned char*> data;
00045     std::vector<unsigned char*> originalData;
00046
00047 public:
00051     Volume() : width(0), height(0), slices(0) {}
00052
00056     ~Volume();
00057
00058     const int& getHeight()  const { return height; }
00059     const int& getWidth() const { return width; }
00060     const int& getSlices() const { return slices; }
00061     const int& getChannels() const { return channels; }
00062     const std::vector<unsigned char*>& getData() const { return data; }
00063
00068     void setData(const std::vector<unsigned char*>& newData);
00069
00075     bool loadVolume(const std::string& directoryPath);
00076
00081     void saveVolume(const std::string& directoryPath);
00082
00090     void saveSlice(const std::string& outputPath, const unsigned char* sliceData, int sliceWidth, int
      sliceHeight) const;
00091
00095     void reloadVolume();
00096
```

```
00102     void cloneData(const std::vector<unsigned char*>& source, std::vector<unsigned char*>&
      destination);
00103
00112     static void generateSamples(const std::string& directory, int count, int width, int height, int
      seed=123);
00113
00118     static void readAndPrintSamples(const std::string& directory);
00119
00123     void printVolumeData() const;
00124
00129     std::vector<unsigned char> getVolumePixelData() const;
00130
00136     std::vector<unsigned char> getImagePixelData(const std::string& path) const;
00137
00138 };
00139
00140 #endif //ADVANCED_PROGRAMMING_GROUP_XAVIER_INITIALIZATION_VOLUME_H
```

Index