# Advanced Programming Group Project Fibonacci

Generated by Doxygen 1.10.0

# Chapter 1

# Class Index

## 1.1 Class List

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

# Chapter 2

# File Index

## 2.1 File List

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

# Chapter 3

# Class Documentation

## 3.1 Filter Class Reference

### Static Public Member Functions

- static void grayscale (Image &image)
- static void brightness (Image &image, int brightness, bool autoBrightness)
- static void histogramEqualization (Image &image, bool hsv)
- static void threshold (Image &image, int threshold, bool useHsv)
- static void addSaltPepperNoise (Image &image, float noisePercentage)
- static void applyMedianFilter2D (Image &image, int kernel_size)
- static void applyBoxFilter2D (Image &image, int kernel_size)
- static void applyGaussianFilter2D (Image &image, int kernel_size, double sigma=2.0)
- static void **sobel** (Image &image)
- static void **robert** (Image &image)
- static void **scharr** (Image &image)
- static void **prewitt** (Image &image)
- static void applyGaussianFilter3D (Volume &volume, int kernelSize, double sigma)

  *Apply a Guassian filter to a 3d volume.*
- static void applyMedianFilter3D (Volume &volume, int kernelSize)

  *Apply a median filter to a 3d volume.*

### 3.1.1 Member Function Documentation

#### 3.1.1.1 addSaltPepperNoise()

```
void Filter::addSaltPepperNoise (
            Image & image,
            float noisePercentage )  [static]
```

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

**Parameters**

| | |
|---|---|
| *image* | The image to add noise to. |
| *noisePercentage* | The percentage of pixels to noise. |

The salt-and-pepper noise filter randomly sets a percentage of pixels to either 0 (black) or 255 (white). The noise percentage determines the number of pixels to noise in the image. The filter is applied to the grayscale image or all channels in the RGB image.

#### 3.1.1.2 applyBoxFilter2D()

```
void Filter::applyBoxFilter2D (
             Image & image,
             int kernel_size )  [static]
```

Applies a box blur filter to the given image.

**Parameters**

| | |
|---|---|
| *image* | The image to apply the filter to. |
| *kernel_size* | The size of the kernel used for blurring. |

Applies a simple box blur filter to the image using the specified kernel size. The box blur filter is a simple averaging of the surrounding pixels. The kernel size determines the number of surrounding pixels to consider.

#### 3.1.1.3 applyGaussianFilter2D()

```
void Filter::applyGaussianFilter2D (
             Image & image,
             int kernel_size,
             double sigma = 2.0 )  [static]
```

Applies a Gaussian blur filter to the given image.

**Parameters**

| | |
|---|---|
| *image* | The image to apply the filter to. |
| *kernel_size* | The size of the kernel used for blurring. |
| *sigma* | The standard deviation of the Gaussian distribution. |

Generates a Gaussian kernel using the specified kernel size. Applies the Gaussian kernel to the image to perform the blur.

#### 3.1.1.4 applyGaussianFilter3D()

```
void Filter::applyGaussianFilter3D (
             Volume & volume,
             int kernelSize,
             double sigma )  [static]
```

Apply a Guassian filter to a 3d volume.

**Parameters**

| | |
|---|---|
| *volume* | Reference to the Volume object to be filtered. The volume is updated in-place with the filtered results. |
| *kernelSize* | The size of the Gaussian kernel to be used for filtering, which should be a positive odd integer. |
| *sigma* | The standard deviation of the Gaussian kernel. |

This function applies a Gaussian filter to the given 3D volume using a separable approach. It first generates a 1D Gaussian kernel based on the specified kernel size and sigma (standard deviation). The filtering is performed in three steps, applying the 1D convolution along the X, Y, and Z axes sequentially. This separable approach is more efficient than a direct 3D convolution, especially for larger kernel sizes. The original volume is updated with the filtered results.

#### 3.1.1.5 applyMedianFilter2D()

```
void Filter::applyMedianFilter2D (
            Image & image,
            int kernel_size )  [static]
```

Applies the image median blur filter to the given image.

**Parameters**

| | |
|---|---|
| *image* | The image to apply the filter to. |
| *kernel_size* | The size of the kernel used for blurring. |

The image median blur filter calculates the median value of the RGB channels for each pixel and assigns the median value to the pixel. The filter is applied to the image by looping through each pixel and its surrounding pixels within the specified kernel size. The median value is calculated for the red, green, and blue channels separately, and then assigned to the current pixel. Source reference: https://chat.openai.↩ com/share/bfc5caa6-324f-4f8a-9967-d4a960433151

#### 3.1.1.6 applyMedianFilter3D()

```
void Filter::applyMedianFilter3D (
            Volume & volume,
            int kernelSize )  [static]
```

Apply a median filter to a 3d volume.

**Parameters**

| | |
|---|---|
| *volume* | Reference to the Volume object to be filtered. The volume is updated in-place with the filtered results. |
| *kernelSize* | The size of the cubic neighborhood around each voxel considered for median filtering, which should be a positive odd integer to ensure a single median value can be computed. |

This function filters the given 3D volume using a median filter. The function operates by moving through each voxel in the volume, considering a cubic neighborhood around it defined by the kernel size. It calculates the median value within this neighborhood and sets the voxel's new value to this median.

### 3.1.1.7 brightness()

```
void Filter::brightness (
            Image & image,
            int brightness = 0,
            bool autoBrightness = false )  [static]
```

Applies a brightness filter to the given image.

**Parameters**

| | |
|---|---|
| *image* | The image to apply the filter to. |
| *brightness* | The brightness value to apply to the image. |
| *autoBrightness* | Flag to enable automatic brightness calculation. |

The brightness filter adjusts the brightness of the image by adding a constant value. The brightness value can be manually set or automatically calculated based on the image. The autoBrightness flag enables automatic brightness calculation based on the average pixel value.

### 3.1.1.8 grayscale()

```
void Filter::grayscale (
            Image & image )  [static]
```

Applies a grayscale filter to the given image.

**Parameters**

| | |
|---|---|
| *image* | The image to apply the filter to. |

The grayscale filter converts the RGB image to a single-channel grayscale image. The grayscale value is calculated using the formula: gray = $0.2126 * r + 0.7152 * g + 0.0722 * b$

### 3.1.1.9 histogramEqualization()

```
void Filter::histogramEqualization (
            Image & image,
            bool hsv )  [static]
```

Applies a histogram equalization filter to the given image.

**Parameters**

| | |
|---|---|
| *image* | The image to apply the filter to. |
| *hsv* | Flag to indicate whether to use HSV or HSL color space. |

The histogram equalization filter enhances the contrast of the image by redistributing the pixel intensity values. The filter is applied to the grayscale image or each channel of the RGB image separately.

#### 3.1.1.10 threshold()

```
void Filter::threshold (
            Image & image,
            int threshold,
            bool useHsv = false )  [static]
```

Applies a threshold filter to the given image.

**Parameters**

| | |
|---|---|
| *image* | The image to apply the filter to. |
| *threshold* | The threshold value to use for binarization. |
| *useHsv* | Flag to enable thresholding in HSV color space. |

The threshold filter converts the image to a binary image based on the threshold value. The filter is applied to the grayscale image or the V channel in the HSV color space. Pixels with intensity values below the threshold are set to 0 (black), and above to 255 (white).

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

- temp/Filter.h
- temp/Filter.cpp

## 3.2 Image Class Reference

### Public Member Functions

- int getWidth () const

  *Get the Width object.*
- int getHeight () const

  *Get the Height object.*
- void setPixel (int x, int y, unsigned char value)

  *Set the Pixel object.*
- unsigned char getPixel (int x, int y) const

  *Get the Pixel object.*
- Image clone () const

  *clone the image*
- Image operator- (const Image &other) const
  - *operator overloading*
- std::string getFileName () const

  *Get the File Name object.*
- void save (const std::string &path) const

  *save the image*
- void createEmpty (int width, int height, int channels)

  *Create a Empty object.*
- void load (const std::string &path)

  *load the image*
- bool isSameAs (const Image &other) const

  *check if the image is same as the other image*
- **Image** (int width=0, int height=0, int channels=3)
- unsigned char getPixel (int x, int y, int ch) const

  *Get the Pixel object.*
- void setPixel (int x, int y, int ch, unsigned char value)

  *Set the Pixel object.*

### Static Public Member Functions

- static void rgbToHsv (float r, float g, float b, float &h, float &s, float &v)

  *rgb to hsv conversion*
- static void hsvToRgb (float h, float s, float v, int &r, int &g, int &b)

  *hsv to rgb conversion*
- static void rgbToHsl (float r, float g, float b, float &h, float &s, float &l)

  *rgb to hsl conversion*
- static void hslToRgb (float h, float s, float l, int &r, int &g, int &b)

  *hsl to rgb conversion*
- static void calculateCdf (const std::vector< int > &histogram, std::vector< int > &cdf)

  *calculate cdf*

### Public Attributes

- unsigned char $*$ **data**

  *constructor*
- int **w**
- int **h**
- int **c**

## 3.2.1 Member Function Documentation

### 3.2.1.1 calculateCdf()

```
void Image::calculateCdf (
            const std::vector< int > & histogram,
            std::vector< int > & cdf )  [static]
```

calculate cdf

**Parameters**

| | |
|---|---|
| *histogram* | input histogram |
| *cdf* | output cdf |

Process:

- Calculate the cumulative distribution function (CDF) from the histogram.
- The CDF is calculated by summing the histogram values.
- The CDF values are stored in the cdf vector.
- The histogram values are in the range [0, 255].
- The cdf values are in the range [0, total number of pixels].
- The cdf vector is resized to match the histogram size.

### 3.2.1.2 clone()

```
Image Image::clone ( ) const
```

clone the image

**Returns**

Image reference to the cloned image

Process:

- Clone the image.
- Return the reference to the cloned image.

### 3.2.1.3 createEmpty()

```
void Image::createEmpty (
              int width,
              int height,
              int channels )
```

Create a Empty object.

**Parameters**

| | |
|---|---|
| *width* | input width |
| *height* | input height |
| *channels* | input channels |

Process:

- Create an empty image with the specified width, height, and number of channels.

### 3.2.1.4 getFileName()

```
std::string Image::getFileName ( ) const
```

Get the File Name object.

**Returns**

std::string name of the file

Process:

- Get the name of the file.
- Return the name of the file.

#### 3.2.1.5 getHeight()

```
int Image::getHeight ( ) const
```

Get the Height object.

**Returns**

int height of the image

Process:

- Get the height of the image.

#### 3.2.1.6 getPixel() [1/2]

```
unsigned char Image::getPixel (
            int x,
            int y ) const
```

Get the Pixel object.

**Parameters**

| | |
|---|---|
| *x* | input x coordinate |
| *y* | input y coordinate |

**Returns**

unsigned char

Process:

- Get the pixel value at the given coordinates.
- If the coordinates are out of bounds, the function returns 0.

#### 3.2.1.7 getPixel() [2/2]

```
unsigned char Image::getPixel (
            int x,
            int y,
            int ch ) const
```

Get the Pixel object.

**Parameters**

| | |
|---|---|
| *x* | input x coordinate |
| *y* | input y coordinate |
| *ch* | input channel |

**Returns**

unsigned char pixel value

Process:

- Get the pixel value at the given coordinates and channel.
- If the coordinates or channel are out of bounds, the function throws an exception.
- Return the pixel value.

### 3.2.1.8 getWidth()

```
int Image::getWidth ( ) const
```

Get the Width object.

**Returns**

int width of the image

Process:

- Get the width of the image.

### 3.2.1.9 hslToRgb()

```
void Image::hslToRgb (
             float h,
             float s,
             float l,
             int & r,
             int & g,
             int & b )  [static]
```

hsl to rgb conversion

**Parameters**

| | |
|---|---|
| *h* | input hue value |
| *s* | input saturation value |
| *l* | input lightness value |
| *r* | output red value |
| *g* | output green value |
| *b* | output blue value |

Process:

- Convert the HSL color to RGB color.
- Return the RGB color.
- The HSL color values are in the range [0, 360] for hue and [0, 1] for saturation and lightness.
- The RGB color values are in the range [0, 255].

### 3.2.1.10 hsvToRgb()

```
void Image::hsvToRgb (
             float h,
             float s,
             float v,
             int & r,
             int & g,
             int & b )  [static]
```

hsv to rgb conversion

**Parameters**

| | |
|---|---|
| *h* | input hue value |
| *s* | input saturation value |
| *v* | input value |
| *r* | output red value |
| *g* | output green value |
| *b* | output blue value |

Process:

- Convert the HSV color to RGB color.
- Return the RGB color.
- The HSV color values are in the range [0, 360] for hue and [0, 1] for saturation and value.
- The RGB color values are in the range [0, 255].
- The hue value is in the range [0, 360].

### 3.2.1.11 isSameAs()

```
bool Image::isSameAs (
             const Image & other ) const
```

check if the image is same as the other image

**Parameters**

|  |  |
|---|---|
| *other* | input image |

**Returns**

true if the image is same

false if the image is not same

Process:

- Check if the image is the same as the other image.
- Return true if the images are the same.
- Return false if the images are not the same.

#### 3.2.1.12 load()

```
void Image::load (
            const std::string & path )
```

load the image

**Parameters**

|  |  |
|---|---|
| *path* | input path |

Process:

- Load the image from the specified path.
- If the image is not loaded successfully, an exception is thrown.

#### 3.2.1.13 operator-()

```
Image Image::operator- (
            const Image & other ) const
```

- operator overloading

**Parameters**

|  |  |
|---|---|
| *other* | input image |

**Returns**

Image resultant image

Process:

- Subtract the pixel values of the two images.
- Return the resultant image.
- If the images have different sizes, the resultant image will have the size of the larger image.
- If the images have different number of channels, the resultant image will have the number of channels of the larger image.
- If the pixel value is negative, it is set to 0.

#### 3.2.1.14 rgbToHsl()

```
void Image::rgbToHsl (
            float r,
            float g,
            float b,
            float & h,
            float & s,
            float & l )  [static]
```

rgb to hsl conversion

**Parameters**

| | |
|---|---|
| *r* | input red value |
| *g* | input green value |
| *b* | input blue value |
| *h* | output hue value |
| *s* | output saturation value |
| *l* | output lightness value |

Process:

- Convert the RGB color to HSL color.
- Return the HSL color.
- The RGB color values are in the range [0, 255].
- The HSL color values are in the range [0, 360] for hue and [0, 1] for saturation and lightness.
- The hue value is in the range [0, 360].

### 3.2.1.15 rgbToHsv()

```
void Image::rgbToHsv (
             float r,
             float g,
             float b,
             float & h,
             float & s,
             float & v )  [static]
```

rgb to hsv conversion

**Parameters**

| | |
|---|---|
| *r* | input red value |
| *g* | input green value |
| *b* | input blue value |
| *h* | output hue value |
| *s* | output saturation value |
| *v* | output value |

Process:

- Convert the RGB color to HSV color.
- Return the HSV color.
- The RGB color values are in the range [0, 255].
- The HSV color values are in the range [0, 360] for hue and [0, 1] for saturation and value.
- The hue value is in the range [0, 360].

### 3.2.1.16 save()

```
void Image::save (
             const std::string & path ) const
```

save the image

**Parameters**

| | |
|---|---|
| *path* | input path |

Process:

- Save the image to the specified path.
- If the image is not saved successfully, an exception is thrown.

### 3.2.1.17 setPixel() [1/2]

```
void Image::setPixel (
            int x,
            int y,
            int ch,
            unsigned char value )
```

Set the Pixel object.

**Parameters**

| | |
|---|---|
| *x* | input x coordinate |
| *y* | input y coordinate |
| *ch* | input channel |
| *value* | input value |

Process:

- Set the pixel value at the given coordinates and channel.

If the coordinates or channel are out of bounds, the function throws an exception.

- Set the pixel value.

### 3.2.1.18 setPixel() [2/2]

```
void Image::setPixel (
            int x,
            int y,
            unsigned char value )
```

Set the Pixel object.

**Parameters**

| | |
|---|---|
| *x* | input x coordinate |
| *y* | input y coordinate |
| *value* | input value |

Process:

- Set the pixel value at the given coordinates.
- If the coordinates are out of bounds, the function does nothing.

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

- temp/Image.h
- temp/Image.cpp

## 3.3 Projection Class Reference

### Static Public Member Functions

- static Image MIP (const Volume &volume, int minIndex, int maxIndex)

  *Apply max intensity projection to 3d volume.*
- static Image MinIP (const Volume &volume, int minIndex, int maxIndex)

  *Apply min intensity projection to 3d volume.*
- static Image AIP (const Volume &volume, int minIndex, int maxIndex)

  *Apply average intensity projection to 3d volume.*
- static Image AIP_Median (const Volume &volume, int minIndex, int maxIndex)

  *Apply median intensity projection to 3d volume.*

### 3.3.1 Member Function Documentation

#### 3.3.1.1 AIP()

```
Image Projection::AIP (
             const Volume & volume,
             int minIndex = 1,
             int maxIndex = -1 )   [static]
```

Apply average intensity projection to 3d volume.

**Parameters**

| | |
|---|---|
| *volume* | input volume |
| *minIndex* | input min index |
| *maxIndex* | input max index |

**Returns**

Image resultant image

Process:

- For each pixel in the output image, find the average intensity value along the z-axis.
- The output image will have the same width and height as the input volume.
- The intensity value at each pixel is the average intensity value found along the z-axis. Reference: https://chat.openai.com/share/53c71398-4690-4906-a012-da51c683b7e1

#### 3.3.1.2 AIP_Median()

```
Image Projection::AIP_Median (
             const Volume & volume,
             int minIndex = 1,
             int maxIndex = -1 )   [static]
```

Apply median intensity projection to 3d volume.

**Parameters**

| | |
|---|---|
| *volume* | input volume |
| *minIndex* | input min index |
| *maxIndex* | input max index |

**Returns**

Image resultant image

Process:

- For each pixel in the output image, find the median intensity value along the z-axis.
- The output image will have the same width and height as the input volume.
- The intensity value at each pixel is the median intensity value found along the z-axis.
- The median intensity value is calculated using the quick select algorithm.

Reference: https://chat.openai.com/share/53c71398-4690-4906-a012-da51c683b7e1

### 3.3.1.3 MinIP()

```
Image Projection::MinIP (
            const Volume & volume,
            int minIndex = 1,
            int maxIndex = -1 )  [static]
```

Apply min intensity projection to 3d volume.

**Parameters**

| | |
|---|---|
| *volume* | input volume |
| *minIndex* | input min index |
| *maxIndex* | input max index |

**Returns**

Image resultant image

Process:

- For each pixel in the output image, find the minimum intensity value along the z-axis.
- The output image will have the same width and height as the input volume.
- The intensity value at each pixel is the minimum intensity value found along the z-axis. Reference: https://chat.openai.com/share/53c71398-4690-4906-a012-da51c683b7e1

#### 3.3.1.4 MIP()

```
Image Projection::MIP (
              const Volume & volume,
              int minIndex = 1,
              int maxIndex = -1 )  [static]
```

Apply max intensity projection to 3d volume.

**Parameters**

| | |
|---|---|
| *volume* | input volume |
| *minIndex* | input min index |
| *maxIndex* | input max index |

**Returns**

Image resultant image

Process:

- For each pixel in the output image, find the maximum intensity value along the z-axis.
- The output image will have the same width and height as the input volume.
- The intensity value at each pixel is the maximum intensity value found along the z-axis. Reference: `https://chat.openai.com/share/53c71398-4690-4906-a012-da51c683b7e1`

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

- temp/Projection.h
- temp/Projection.cpp

## 3.4 Slicer Class Reference

### Static Public Member Functions

- static Image sliceXY (const Volume &volume, int z)

  *Slice the volume in XY plane at z.*
- static Image sliceYZ (const Volume &volume, int x)

  *Slice the volume in YZ plane at x.*
- static Image sliceXZ (const Volume &volume, int y)

  *Slice the volume in XZ plane at y.*
- static void generateSlices (const Volume &volume, const std::string &sliceFolder, int xCoordinate, int y↩ Coordinate)

  *Generate slices at specific coordinates.*

## 3.4.1 Member Function Documentation

### 3.4.1.1 generateSlices()

```
void Slicer::generateSlices (
             const Volume & volume,
             const std::string & sliceFolder,
             int xCoordinate,
             int yCoordinate )  [static]
```

Generate slices at specific coordinates.

**Parameters**

| | |
|---|---|
| *volume* | input volume |
| *sliceFolder* | input slice folder |
| *xCoordinate* | input x coordinate |
| *yCoordinate* | input y coordinate |

Process:

- Ensure the slice folder exists.
- Generate and save YZ slice at xCoordinate.
- Generate and save XZ slice at yCoordinate.

#### 3.4.1.2 sliceXY()

```
Image Slicer::sliceXY (
                const Volume & volume,
                int z )   [static]
```

Slice the volume in XY plane at z.

**Parameters**

| | |
|---|---|
| *volume* | input volume |
| *z* | input z coordinate |

**Returns**

Image resultant image

Process:

- Generate a slice in the XY plane at a given z-depth.
- If the z-coordinate is out of range, print an error message and return an empty image.

#### 3.4.1.3 sliceXZ()

```
Image Slicer::sliceXZ (
                const Volume & volume,
                int y )   [static]
```

Slice the volume in XZ plane at y.

**Parameters**

| | |
|---|---|
| *volume* | input volume |
| *y* | input y coordinate |

**Returns**

Image resultant image

Process:

- Generate a slice in the XZ plane at a given y-coordinate.
- If the y-coordinate is out of range, print an error message and return an empty image.

#### 3.4.1.4 sliceYZ()

```
Image Slicer::sliceYZ (
            const Volume & volume,
            int x )  [static]
```

Slice the volume in YZ plane at x.

**Parameters**

| | |
|---|---|
| *volume* | input volume |
| *x* | input x coordinate |

**Returns**

Image resultant image Process:

- Generate a slice in the YZ plane at a given x-coordinate.
- If the x-coordinate is out of range, print an error message and return an empty image.

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

- temp/Slice.h
- temp/Slice.cpp

## 3.5 Volume Class Reference

### Public Member Functions

- **Volume** ()

  *Construct a new Volume object.*
- Volume (const std::string &directoryPath)

  *Construct a new Volume object from the directory path.*
- Image getSlice (int index) const

  *Get the Slice object.*
- void loadVolume (const std::string &directoryPath)

  *load the volume from the directory path*
- void createEmpty (int width, int height, int depth)

*Create a Empty object.*

- int getWidth () const

  *Get the Width object.*

- int getHeight () const

  *Get the Height object.*

- int getDepth () const

  *Get the Depth object.*

- unsigned char getVoxel (int x, int y, int z) const

  *Get the Voxel object.*

- void setVoxel (unsigned char v, int x, int y, int z)

  *Set the Voxel object.*

- void updateSlice (int z, Image newData)

  *Update the Slice object.*

- void saveVolume (const std::string &outputDirectory) const

  *save the volume to the output directory*

- Image sliceXY (int zCoordinate) const

  *slice the volume in the XY plane*

- Image sliceXZ (int yCoordinate) const

  *slice the volume in the XZ plane*

- Image sliceYZ (int xCoordinate) const

  *slice the volume in the YZ plane*

- const std::vector< Image > & getSlices () const

  *Get the Slices object.*

## 3.5.1 Constructor & Destructor Documentation

### 3.5.1.1 Volume()

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

Construct a new Volume object from the directory path.

**Parameters**

| | |
|---|---|
| *directoryPath* | input directory path |

## 3.5.2 Member Function Documentation

### 3.5.2.1 createEmpty()

```
void Volume::createEmpty (
            int width,
            int height,
            int depth )
```

Create a Empty object.

**Parameters**

| | |
|---|---|
| *width* | input width |
| *height* | input height |
| *depth* | input depth |

#### 3.5.2.2 getDepth()

```
int Volume::getDepth ( ) const
```

Get the Depth object.

**Returns**

int resultant depth

#### 3.5.2.3 getHeight()

```
int Volume::getHeight ( ) const
```

Get the Height object.

**Returns**

int resultant height

#### 3.5.2.4 getSlice()

```
Image Volume::getSlice (
              int index ) const
```

Get the Slice object.

**Parameters**

| | |
|---|---|
| *index* | input index |

**Returns**

Image resultant image

#### 3.5.2.5 getSlices()

```
const std::vector< Image > & Volume::getSlices ( ) const  [inline]
```

Get the Slices object.

**Returns**

const std::vector<Image>& resultant vector of images

### 3.5.2.6 getVoxel()

```
unsigned char Volume::getVoxel (
             int x,
             int y,
             int z ) const
```

Get the Voxel object.

**Parameters**

| | |
|---|---|
| *x* | input x |
| *y* | input y |
| *z* | input z |

**Returns**

unsigned char resultant unsigned char

### 3.5.2.7 getWidth()

```
int Volume::getWidth ( ) const
```

Get the Width object.

**Returns**

int resultant width

### 3.5.2.8 loadVolume()

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

load the volume from the directory path

**Parameters**

| | |
|---|---|
| *directoryPath* | input directory path |

#### 3.5.2.9 saveVolume()

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

save the volume to the output directory

**Parameters**

| | |
|---|---|
| *outputDirectory* | input output directory |

#### 3.5.2.10 setVoxel()

```
void Volume::setVoxel (
             unsigned char v,
             int x,
             int y,
             int z )
```

Set the Voxel object.

**Parameters**

| | |
|---|---|
| *v* | input v |
| *x* | input x |
| *y* | input y |
| *z* | input z |

#### 3.5.2.11 sliceXY()

```
Image Volume::sliceXY (
             int zCoordinate ) const
```

slice the volume in the XY plane

**Parameters**

| | |
|---|---|
| *zCoordinate* | input z coordinate |

**Returns**

Image resultant image

#### 3.5.2.12 sliceXZ()

```
Image Volume::sliceXZ (
             int yCoordinate ) const
```

slice the volume in the XZ plane

**Parameters**

| | |
|---|---|
| *yCoordinate* | input y coordinate |

**Returns**

Image resultant image

#### 3.5.2.13 sliceYZ()

```
Image Volume::sliceYZ (
             int xCoordinate ) const
```

slice the volume in the YZ plane

**Parameters**

| | |
|---|---|
| *xCoordinate* | input x coordinate |

**Returns**

Image resultant image

#### 3.5.2.14 updateSlice()

```
void Volume::updateSlice (
             int z,
             Image newData )
```

Update the Slice object.

**Parameters**

| | |
|---|---|
| *z* | input z |
| *newData* | input newData |

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

- temp/Volume.h
- temp/Volume.cpp

# Chapter 4

# File Documentation

## 4.1 temp/Filter.cpp File Reference

```
#include <iostream>
#include <vector>
#include <cassert>
#include <algorithm>
#include <cmath>
#include <cstdlib>
#include <tuple>
#include <ctime>
#include "utils.h"
#include "Filter.h"
#include "Image.h"
#include <chrono>
```

### 4.1.1 Detailed Description

Group Name: Fibonacci Name: Yibin Gao Github_name: edsml-kg23 Name: Ruihan He Github_name: edsml-rh323 Name: Yu Yin Github_name: acse-yy923 Name: Sara Lakatos Github_name: acse-sl4623 Name: Manawi Kahie Github_name: acse-mk1923 Name: Wenyi Yang Github_name: acse-wy1023

## 4.2 temp/Filter.h File Reference

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

**Classes**

- class Filter

### Enumerations

- enum class **Axis** { **X** , **Y** , **Z** }

### 4.2.1 Detailed Description

Group Name: Fibonacci Name: Yibin Gao Github_name: edsml-kg23 Name: Ruihan He Github_name: edsml-rh323 Name: Yu Yin Github_name: acse-yy923 Name: Sara Lakatos Github_name: acse-sl4623 Name: Manawi Kahie Github_name: acse-mk1923 Name: Wenyi Yang Github_name: acse-wy1023

## 4.3 Filter.h

Go to the documentation of this file.

```

#ifndef FILTER_H
#define FILTER_H

#include "Image.h"
#include "Volume.h"
#include <string>
#include <vector>


enum class Axis { X, Y, Z };


class Filter {
private:
    enum class PaddingMethod {
        Zero, // Zero padding
        Reflect, // Reflect padding
        Copy // Copy edge padding
    };

/*
 * Applies a generic edge detection filter to an image using specified gradient operators.
 *
 * @param image Reference to an Image object containing the source image data.
 *              The function directly modifies the image object to store the result.
 * @param gx    The horizontal gradient operator (kernel) for edge detection.
 * @param gy    The vertical gradient operator (kernel) for edge detection.
 * @param padding Enum value specifying the padding method to use (Zero, Reflect, Copy).
 *
 * Process:
 * - Computes horizontal and vertical gradients at each pixel using gx and gy kernels.
 * - Calculates the gradient magnitude as the Euclidean norm of the horizontal and vertical gradients.
 * - Normalizes the gradient magnitude to fit within the range of 0 to 255.
 * - Supports different padding methods to handle edge pixels.
 * - reference: https://chat.openai.com/share/6133745c-f97e-465e-a3a8-a39cfe72eeaa
 */
    static void applyEdgeDetection(Image &image, const int gx[3][3], const int gy[3][3],
                                   PaddingMethod padding = PaddingMethod::Reflect);

public:

    static void grayscale(Image &image);

    static void brightness(Image &image, int brightness, bool autoBrightness);

    static void histogramEqualization(Image &image, bool hsv);

    static void threshold(Image &image, int threshold, bool useHsv);

    static void addSaltPepperNoise(Image &image, float noisePercentage);

    static void applyMedianFilter2D(Image& image, int kernel_size);

    static void applyBoxFilter2D(Image& image, int kernel_size);

    static void applyGaussianFilter2D(Image& image, int kernel_size, double sigma = 2.0);

/*
 * Applies the Sobel edge detection filter to an image.
 *
 * @param image Reference to an Image object that will be processed.
 *              The result is stored directly in this object.
 *
 * Process:
 * - Utilizes predefined Sobel operators for horizontal and vertical gradient calculation.
 * - Invokes applyEdgeDetection with the Sobel operators to detect edges in the image.
 * - reference: https://en.wikipedia.org/wiki/Sobel_operator
 */
    static void sobel(Image &image);

/*
 * Applies the Robert Cross edge detection filter to an image.
 *
 * @param image Reference to an Image object that will be processed.
 *              The result is stored directly in this object.
 *
 * Process:
 * - Uses predefined Robert Cross operators for horizontal and vertical gradient calculation.
 * - Invokes applyEdgeDetection with the Robert Cross operators to perform the edge detection.
 * - reference: https://en.wikipedia.org/wiki/Roberts_Cross
 */
    static void robert(Image &image);

/*
 * Applies the Scharr edge detection filter to an image.
 *
 * @param image Reference to an Image object that will be processed.
 *              The result is stored directly in this object.
 *
 * Process:
 * - Uses predefined Scharr operators for more accurate horizontal and vertical gradient calculation.
 * - Invokes applyEdgeDetection with the Scharr operators to perform the edge detection.
 * - reference: https://theailearner.com/tag/scharr-operator/
 */
    static void scharr(Image &image);

/*
 * Applies the Prewitt edge detection filter to an image.
 *
 * @param image Reference to an Image object that will be processed.
 *              The result is stored directly in this object.
 *
 * Process:
 * - Uses predefined Prewitt operators for horizontal and vertical gradient calculation.
 * - Invokes applyEdgeDetection with the Prewitt operators to perform the edge detection.
 * - reference: https://en.wikipedia.org/wiki/Prewitt_operator
 */
    static void prewitt(Image &image);

    static void applyGaussianFilter3D(Volume& volume, int kernelSize, double sigma);

    static void applyMedianFilter3D(Volume& volume, int kernelSize);

};
#endif

```

## 4.4 temp/Image.cpp File Reference

```
#include "Image.h"
#include "stb_image.h"
#include "stb_image_write.h"
#include <iostream>
#include <cmath>
#include <algorithm>
#include <cstring>
```

### 4.4.1 Detailed Description

Group Name: Fibonacci Name: Yibin Gao Github_name: edsml-kg23 Name: Ruihan He Github_name: edsml-rh323 Name: Yu Yin Github_name: acse-yy923 Name: Sara Lakatos Github_name: acse-sl4623 Name: Manawi Kahie Github_name: acse-mk1923 Name: Wenyi Yang Github_name: acse-wy1023

## 4.5 temp/Image.h File Reference

```
#include <string>
#include <vector>
```

### Classes

- class Image

### 4.5.1 Detailed Description

Group Name: Fibonacci Name: Yibin Gao Github_name: edsml-kg23 Name: Ruihan He Github_name: edsml-rh323 Name: Yu Yin Github_name: acse-yy923 Name: Sara Lakatos Github_name: acse-sl4623 Name: Manawi Kahie Github_name: acse-mk1923 Name: Wenyi Yang Github_name: acse-wy1023

## 4.6 Image.h

Go to the documentation of this file.

```

// Image.h
#ifndef IMAGE_H
#define IMAGE_H

#include <string>
#include <vector>

class Image {
public:
    //3d slicing

    int getWidth() const;

    int getHeight() const;

    void setPixel(int x, int y, unsigned char value);

    unsigned char getPixel(int x, int y) const;

    Image clone() const;

    Image operator-(const Image& other) const;

    std::string getFileName() const;

    void save(const std::string& path) const;

    void createEmpty(int width, int height, int channels);

    void load(const std::string& path);

    bool isSameAs(const Image& other) const;

    unsigned char* data;
    int w, h, c;

    // Constructor
    Image(int width = 0, int height = 0, int channels = 3);

    // getter functions
    unsigned char getPixel(int x, int y, int ch) const;

    void setPixel(int x, int y, int ch, unsigned char value);

    // Utility functions declarations

    static void rgbToHsv(float r, float g, float b, float &h, float &s, float &v);

    static void hsvToRgb(float h, float s, float v, int &r, int &g, int &b);

    static void rgbToHsl(float r, float g, float b, float &h, float &s, float &l);

    static void hslToRgb(float h, float s, float l, int &r, int &g, int &b);

    static void calculateCdf(const std::vector<int>& histogram, std::vector<int>& cdf);
};

#endif
```

## 4.7 temp/main.cpp File Reference

```
#include <iostream>
#include <string>
#include <sstream>
#include <vector>
#include <limits>
#include "Filter.h"
#include "Slice.h"
#include "Projection.h"
#include "Volume.h"
#include "utils.h"
#include "algorithm"
```

#### Functions

- int **main** ()

### 4.7.1 Detailed Description

Group Name: Fibonacci Name: Yibin Gao Github_name: edsml-kg23 Name: Ruihan He Github_name: edsml-rh323 Name: Yu Yin Github_name: acse-yy923 Name: Sara Lakatos Github_name: acse-sl4623 Name: Manawi Kahie Github_name: acse-mk1923 Name: Wenyi Yang Github_name: acse-wy1023

## 4.8 temp/Projection.cpp File Reference

```
#include "Projection.h"
#include <limits>
#include <iostream>
#include <vector>
#include "utils.h"
```

#### Functions

- int **adjustIndex** (int userIndex, int sliceCount)

### 4.8.1 Detailed Description

Group Name: Fibonacci Name: Yibin Gao Github_name: edsml-kg23 Name: Ruihan He Github_name: edsml-rh323 Name: Yu Yin Github_name: acse-yy923 Name: Sara Lakatos Github_name: acse-sl4623 Name: Manawi Kahie Github_name: acse-mk1923 Name: Wenyi Yang Github_name: acse-wy1023

## 4.9 temp/Projection.h File Reference

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

**Classes**

- class Projection

### 4.9.1 Detailed Description

Group Name: Fibonacci Name: Yibin Gao Github_name: edsml-kg23 Name: Ruihan He Github_name: edsml-rh323 Name: Yu Yin Github_name: acse-yy923 Name: Sara Lakatos Github_name: acse-sl4623 Name: Manawi Kahie Github_name: acse-mk1923 Name: Wenyi Yang Github_name: acse-wy1023

## 4.10 Projection.h

Go to the documentation of this file.

```

// Projection.h
#ifndef PROJECTION_H
#define PROJECTION_H
#include "Image.h"
#include "Volume.h"

class Projection {
public:
    static Image MIP(const Volume& volume, int minIndex, int maxIndex);

    static Image MinIP(const Volume& volume, int minIndex, int maxIndex);

    static Image AIP(const Volume& volume, int minIndex, int maxIndex);

    static Image AIP_Median(const Volume& volume, int minIndex, int maxIndex);
};

#endif // PROJECTION_H
```

## 4.11 temp/Slice.cpp File Reference

```
#include "Slice.h"
#include <iostream>
#include "Volume.h"
#include <filesystem>
```

### 4.11.1 Detailed Description

Group Name: Fibonacci Name: Yibin Gao Github_name: edsml-kg23 Name: Ruihan He Github_name: edsml-rh323 Name: Yu Yin Github_name: acse-yy923 Name: Sara Lakatos Github_name: acse-sl4623 Name: Manawi Kahie Github_name: acse-mk1923 Name: Wenyi Yang Github_name: acse-wy1023

## 4.12 temp/Slice.h File Reference

```
#include "Volume.h"
#include "Image.h"
#include <string>
```

**Classes**

- class Slicer

### 4.12.1 Detailed Description

Group Name: Fibonacci Name: Yibin Gao Github_name: edsml-kg23 Name: Ruihan He Github_name: edsml-rh323 Name: Yu Yin Github_name: acse-yy923 Name: Sara Lakatos Github_name: acse-sl4623 Name: Manawi Kahie Github_name: acse-mk1923 Name: Wenyi Yang Github_name: acse-wy1023

## 4.13 Slice.h

Go to the documentation of this file.

```

//Slice.h
#ifndef SLICE_H
#define SLICE_H

#include "Volume.h"
#include "Image.h"
#include <string>

class Slicer {
public:


    static Image sliceXY(const Volume& volume, int z);

    static Image sliceYZ(const Volume& volume, int x);

    static Image sliceXZ(const Volume& volume, int y);


    // Declare the new method for generating slices at specific coordinates

    static void generateSlices(const Volume& volume, const std::string& sliceFolder, int xCoordinate, int yCoordinate);
};

#endif // SLICE_H
```

## 4.14 temp/utils.cpp File Reference

```
#include <cmath>
#include <iostream>
#include "Filter.h"
#include "utils.h"
```

#### Functions

- int partition (std::vector< unsigned char > &arr, int left, int right)

  *Partitions the array for the quicksort algorithm.*
- unsigned char quickSelect (std::vector< unsigned char > &arr, int left, int right, int k)

  *Apply quick select algorithm to find the k-th smallest element.*
- std::vector< double > generate1DGaussianKernel (int kernelSize, double sigma)

  *Generate 1D Gaussian kernel.*
- void apply1DConvolution2D (Image &inputImage, Image &outputImage, const std::vector< double > &kernel, Axis axis)

  *Applies a 1D convolution to a 2D image along a specified axis.*
- void apply1DConvolution3D (Volume &inputVolume, Volume &outputVolume, const std::vector< double > &kernel, Axis axis)

  *Apply 1D convolution to a volume.*
- void merge (std::vector< std::string > &filePaths, int left, int middle, int right)

  *Merge two sorted arrays.*
- void mergeSort (std::vector< std::string > &filePaths, int left, int right)

  *Merge sort the array.*
- std::string extractFilename (const std::string &path)

  *Extract the filename from the path.*
- std::string extractFilenameWithoutExtension (const std::string &filename)

  *Extract the filename without extension.*
- std::string convertNumberToString (int number)

  *Convert number to string.*
- void createOutputDirectory (const std::string &path)

  *Create output directory.*

### 4.14.1 Detailed Description

Group Name: Fibonacci Name: Yibin Gao Github_name: edsml-kg23 Name: Ruihan He Github_name: edsml-rh323 Name: Yu Yin Github_name: acse-yy923 Name: Sara Lakatos Github_name: acse-sl4623 Name: Manawi Kahie Github_name: acse-mk1923 Name: Wenyi Yang Github_name: acse-wy1023

### 4.14.2 Function Documentation

#### 4.14.2.1 apply1DConvolution2D()

```
void apply1DConvolution2D (
            Image & inputImage,
            Image & outputImage,
            const std::vector< double > & kernel,
            Axis axis )
```

Applies a 1D convolution to a 2D image along a specified axis.

**Parameters**

| | |
|---|---|
| *inputImage* | The input image to be convolved. This image remains unchanged. |
| *outputImage* | The output image where the result of the convolution is stored. |
| *kernel* | The convolution kernel as a vector of doubles. The kernel should be normalized. |
| *axis* | The axis along which to apply the convolution, specified as an `Axis` type. This can be `Axis::X` for horizontal or `Axis::Y` for vertical convolution. |

This function applies a 1D convolution to the input image using the specified kernel. The convolution can be performed either horizontally (along the X axis) or vertically (along the Y axis) as specified by the `axis` parameter. The function handles edge pixels by replicating the nearest edge value, ensuring that the output image has the same dimensions as the input. The output image is then filled with the results of the convolution.

#### 4.14.2.2 apply1DConvolution3D()

```
void apply1DConvolution3D (
             Volume & inputVolume,
             Volume & outputVolume,
             const std::vector< double > & kernel,
             Axis axis )
```

Apply 1D convolution to a volume.

**Parameters**

| | |
|---|---|
| *inputVolume* | Input volume for convolution. |
| *outputVolume* | Volume to store the convolved data. |
| *kernel* | Convolution kernel, normalized to ensure intensity preservation. |
| *axis* | Axis along which the convolution is applied (X, Y, or Z). |

Performs 1D convolution across one axis of a 3D volume with a specified kernel. It's used to apply filters like Gaussian blur in a separable manner. The function ensures edges are handled by replicating edge values, maintaining the volume's dimensions.

#### 4.14.2.3 convertNumberToString()

```
std::string convertNumberToString (
             int number )
```

Convert number to string.

**Parameters**

| | |
|---|---|
| *number* | input number |

**Returns**

std::string resultant string

#### 4.14.2.4 createOutputDirectory()

```
void createOutputDirectory (
             const std::string & path )
```

Create output directory.

**Parameters**

| | |
|---|---|
| *path* | input path |

#### 4.14.2.5 extractFilename()

```
std::string extractFilename (
             const std::string & path )
```

Extract the filename from the path.

**Parameters**

| | |
|---|---|
| *path* | input path |

**Returns**

std::string resultant filename

#### 4.14.2.6 extractFilenameWithoutExtension()

```
std::string extractFilenameWithoutExtension (
             const std::string & filename )
```

Extract the filename without extension.

**Parameters**

| | |
|---|---|
| *filename* | input filename |

**Returns**

std::string resultant filename

#### 4.14.2.7 generate1DGaussianKernel()

```
std::vector< double > generate1DGaussianKernel (
             int kernelSize,
             double sigma )
```

Generate 1D Gaussian kernel.

**Parameters**

| | |
|---|---|
| *kernelSize* | The size of the kernel. It determines the length of the generated Gaussian kernel vector. The kernel size should be a positive odd integer to ensure symmetry around the center. |
| *sigma* | The standard deviation of the Gaussian distribution. |

**Returns**

> std::vector<double> A vector of doubles representing the normalized 1D Gaussian kernel.

This function generates a 1D Gaussian kernel with a specified size and standard deviation (sigma). The generated kernel is normalized so that the sum of its elements equals 1, ensuring that the application of the kernel does not change the overall brightness of the image.

#### 4.14.2.8 merge()

```
void merge (
             std::vector< std::string > & filePaths,
             int left,
             int middle,
             int right )
```

Merge two sorted arrays.

**Parameters**

| | |
|---|---|
| *arr* | input array |
| *left* | input left index |
| *middle* | input middle index |
| *right* | input right index |

#### 4.14.2.9 mergeSort()

```
void mergeSort (
             std::vector< std::string > & filePaths,
             int left,
             int right )
```

Merge sort the array.

**Parameters**

| | |
|---|---|
| *arr* | input array |
| *left* | input left index |
| *right* | input right index |

#### 4.14.2.10 partition()

```
int partition (
            std::vector< unsigned char > & arr,
            int left,
            int right )
```

Partitions the array for the quicksort algorithm.

**Parameters**

| | |
|---|---|
| *arr* | Reference to the vector of unsigned char elements to be partitioned. |
| *left* | The starting index of the segment of the array to be partitioned. |
| *right* | The ending index of the segment of the array to be partitioned. |

**Returns**

int The index position of the pivot element after partitioning.

This function takes the last element as pivot, places the pivot element at its correct position in the sorted array, and places all smaller (smaller than pivot) to the left of the pivot and all greater elements to the right of the pivot. This is a helper function used by the quicksort algorithm for sorting elements within an array. Reference: Quickselect algorithm: https://chat.openai.com/share/4db4258a-0cc1-4d63-b2fd-8191371133e1

#### 4.14.2.11 quickSelect()

```
unsigned char quickSelect (
            std::vector< unsigned char > & arr,
            int left,
            int right,
            int k )
```

Apply quick select algorithm to find the k-th smallest element.

**Parameters**

| | |
|---|---|
| *arr* | Reference to the vector of unsigned char elements from which to select. |
| *left* | The starting index of the segment within the array to perform the selection. |
| *right* | The ending index of the segment within the array to perform the selection. |
| *k* | The index (0-based) of the smallest element to find. For example, k = 0 will find the smallest element, and k = length of array - 1 will find the largest element. |

**Returns**

unsigned char The k-th smallest element in the specified segment of the array.

Quickselect is a selection algorithm to find the k-th smallest element in an unordered list.Quickselect uses the same overall approach as quicksort, choosing one element as a pivot and partitioning the data in two based on the pivot, accordingly as less than or greater than the pivot.

## 4.15 temp/utils.h File Reference

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

### Functions

- int partition (std::vector< unsigned char > &arr, int left, int right)

    *Partitions the array for the quicksort algorithm.*
- unsigned char quickSelect (std::vector< unsigned char > &arr, int left, int right, int k)

    *Apply quick select algorithm to find the k-th smallest element.*
- std::vector< double > generate1DGaussianKernel (int kernelSize, double sigma)

    *Generate 1D Gaussian kernel.*
- void apply1DConvolution2D (Image &inputImage, Image &outputImage, const std::vector< double > &kernel, Axis axis)

    *Applies a 1D convolution to a 2D image along a specified axis.*
- void apply1DConvolution3D (Volume &inputVolume, Volume &outputVolume, const std::vector< double > &kernel, Axis axis)

    *Apply 1D convolution to a volume.*
- void merge (std::vector< std::string > &filePaths, int left, int middle, int right)

    *Merge two sorted arrays.*
- void mergeSort (std::vector< std::string > &filePaths, int left, int right)

    *Merge sort the array.*
- std::string extractFilenameWithoutExtension (const std::string &filename)

    *Extract the filename without extension.*
- std::string convertNumberToString (int number)

    *Convert number to string.*
- void createOutputDirectory (const std::string &path)

    *Create output directory.*
- std::string extractFilename (const std::string &path)

    *Extract the filename from the path.*

### 4.15.1 Detailed Description

Group Name: Fibonacci Name: Yibin Gao Github_name: edsml-kg23 Name: Ruihan He Github_name: edsml-rh323 Name: Yu Yin Github_name: acse-yy923 Name: Sara Lakatos Github_name: acse-sl4623 Name: Manawi Kahie Github_name: acse-mk1923 Name: Wenyi Yang Github_name: acse-wy1023

### 4.15.2 Function Documentation

#### 4.15.2.1 apply1DConvolution2D()

```
void apply1DConvolution2D (
            Image & inputImage,
            Image & outputImage,
            const std::vector< double > & kernel,
            Axis axis )
```

Applies a 1D convolution to a 2D image along a specified axis.

**Parameters**

| | |
|---|---|
| *inputImage* | The input image to be convolved. This image remains unchanged. |
| *outputImage* | The output image where the result of the convolution is stored. |
| *kernel* | The convolution kernel as a vector of doubles. The kernel should be normalized. |
| *axis* | The axis along which to apply the convolution, specified as an `Axis` type. This can be `Axis::X` for horizontal or `Axis::Y` for vertical convolution. |

This function applies a 1D convolution to the input image using the specified kernel. The convolution can be performed either horizontally (along the X axis) or vertically (along the Y axis) as specified by the `axis` parameter. The function handles edge pixels by replicating the nearest edge value, ensuring that the output image has the same dimensions as the input. The output image is then filled with the results of the convolution.

#### 4.15.2.2 apply1DConvolution3D()

```
void apply1DConvolution3D (
            Volume & inputVolume,
            Volume & outputVolume,
            const std::vector< double > & kernel,
            Axis axis )
```

Apply 1D convolution to a volume.

**Parameters**

| | |
|---|---|
| *inputVolume* | Input volume for convolution. |
| *outputVolume* | Volume to store the convolved data. |
| *kernel* | Convolution kernel, normalized to ensure intensity preservation. |
| *axis* | Axis along which the convolution is applied (X, Y, or Z). |

Performs 1D convolution across one axis of a 3D volume with a specified kernel. It's used to apply filters like Gaussian blur in a separable manner. The function ensures edges are handled by replicating edge values, maintaining the volume's dimensions.

#### 4.15.2.3 convertNumberToString()

```
std::string convertNumberToString (
            int number )
```

Convert number to string.

**Parameters**

| | |
|---|---|
| *number* | input number |

**Returns**

std::string resultant string

#### 4.15.2.4 createOutputDirectory()

```
void createOutputDirectory (
             const std::string & path )
```

Create output directory.

**Parameters**

| | |
|---|---|
| *path* | input path |

#### 4.15.2.5 extractFilename()

```
std::string extractFilename (
             const std::string & path )
```

Extract the filename from the path.

**Parameters**

| | |
|---|---|
| *path* | input path |

**Returns**

std::string resultant filename

#### 4.15.2.6 extractFilenameWithoutExtension()

```
std::string extractFilenameWithoutExtension (
             const std::string & filename )
```

Extract the filename without extension.

**Parameters**

| | |
|---|---|
| *filename* | input filename |

**Returns**

std::string resultant filename

#### 4.15.2.7 generate1DGaussianKernel()

```
std::vector< double > generate1DGaussianKernel (
             int kernelSize,
             double sigma )
```

Generate 1D Gaussian kernel.

**Parameters**

| | |
|---|---|
| *kernelSize* | The size of the kernel. It determines the length of the generated Gaussian kernel vector. The kernel size should be a positive odd integer to ensure symmetry around the center. |
| *sigma* | The standard deviation of the Gaussian distribution. |

**Returns**

> std::vector<double> A vector of doubles representing the normalized 1D Gaussian kernel.

This function generates a 1D Gaussian kernel with a specified size and standard deviation (sigma). The generated kernel is normalized so that the sum of its elements equals 1, ensuring that the application of the kernel does not change the overall brightness of the image.

#### 4.15.2.8 merge()

```
void merge (
             std::vector< std::string > & filePaths,
             int left,
             int middle,
             int right )
```

Merge two sorted arrays.

**Parameters**

| | |
|---|---|
| *arr* | input array |
| *left* | input left index |
| *middle* | input middle index |
| *right* | input right index |

#### 4.15.2.9 mergeSort()

```
void mergeSort (
             std::vector< std::string > & filePaths,
             int left,
             int right )
```

Merge sort the array.

**Parameters**

| | |
|---|---|
| *arr* | input array |
| *left* | input left index |
| *right* | input right index |

#### 4.15.2.10 partition()

```
int partition (
            std::vector< unsigned char > & arr,
            int left,
            int right )
```

Partitions the array for the quicksort algorithm.

**Parameters**

| | |
|---|---|
| *arr* | Reference to the vector of unsigned char elements to be partitioned. |
| *left* | The starting index of the segment of the array to be partitioned. |
| *right* | The ending index of the segment of the array to be partitioned. |

**Returns**

int The index position of the pivot element after partitioning.

This function takes the last element as pivot, places the pivot element at its correct position in the sorted array, and places all smaller (smaller than pivot) to the left of the pivot and all greater elements to the right of the pivot. This is a helper function used by the quicksort algorithm for sorting elements within an array. Reference: Quickselect algorithm: https://chat.openai.com/share/4db4258a-0cc1-4d63-b2fd-8191371133e1

#### 4.15.2.11 quickSelect()

```
unsigned char quickSelect (
            std::vector< unsigned char > & arr,
            int left,
            int right,
            int k )
```

Apply quick select algorithm to find the k-th smallest element.

**Parameters**

| | |
|---|---|
| *arr* | Reference to the vector of unsigned char elements from which to select. |
| *left* | The starting index of the segment within the array to perform the selection. |
| *right* | The ending index of the segment within the array to perform the selection. |
| *k* | The index (0-based) of the smallest element to find. For example, k = 0 will find the smallest element, and k = length of array - 1 will find the largest element. |

**Returns**

unsigned char The k-th smallest element in the specified segment of the array.

Quickselect is a selection algorithm to find the k-th smallest element in an unordered list.Quickselect uses the same overall approach as quicksort, choosing one element as a pivot and partitioning the data in two based on the pivot, accordingly as less than or greater than the pivot.

## 4.16 utils.h

Go to the documentation of this file.

```

#ifndef UTILS_H
#define UTILS_H

#include <iostream>
#include <vector>
#include <cmath>
#include <vector>
#include "Image.h" // Make sure this is correctly included for Image class definition
#include "Volume.h" // Include the header file that defines the "Volume" class
#include "Filter.h" // Include if necessary for Axis enum

int partition(std::vector<unsigned char>& arr, int left, int right);

unsigned char quickSelect(std::vector<unsigned char>& arr, int left, int right, int k);

std::vector<double> generate1DGaussianKernel(int kernelSize, double sigma);

void apply1DConvolution2D(Image& inputImage, Image& outputImage, const std::vector<double>& kernel, Axis axis);

void apply1DConvolution3D(Volume& inputVolume, Volume& outputVolume, const std::vector<double>& kernel, Axis axis);

void merge(std::vector<std::string>& filePaths, int left, int middle, int right);

void mergeSort(std::vector<std::string>& filePaths, int left, int right);

std::string extractFilenameWithoutExtension(const std::string &filename);

std::string convertNumberToString(int number);

void createOutputDirectory(const std::string &path);

std::string extractFilename(const std::string &path);

#endif // UTILS_H

```

## 4.17 temp/Volume.cpp File Reference

```
#include "Volume.h"
#include <iostream>
#include <filesystem>
#include <string>
#include <sstream>
#include <algorithm>
#include "utils.h"
```

### 4.17.1 Detailed Description

Group Name: Fibonacci Name: Yibin Gao Github_name: edsml-kg23 Name: Ruihan He Github_name: edsml-rh323 Name: Yu Yin Github_name: acse-yy923 Name: Sara Lakatos Github_name: acse-sl4623 Name: Manawi Kahie Github_name: acse-mk1923 Name: Wenyi Yang Github_name: acse-wy1023

## 4.18 temp/Volume.h File Reference

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

### Classes

- class Volume

### 4.18.1 Detailed Description

Group Name: Fibonacci Name: Yibin Gao Github_name: edsml-kg23 Name: Ruihan He Github_name: edsml-rh323 Name: Yu Yin Github_name: acse-yy923 Name: Sara Lakatos Github_name: acse-sl4623 Name: Manawi Kahie Github_name: acse-mk1923 Name: Wenyi Yang Github_name: acse-wy1023

## 4.19 Volume.h

Go to the documentation of this file.

```

// Volume.h
#ifndef VOLUME_H
#define VOLUME_H

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

class Volume {
public:
    Volume(); // use the directory path to load the volume

    Volume(const std::string& directoryPath);

    Image getSlice(int index) const; // get the image slice from the volume index

    void loadVolume(const std::string& directoryPath); // load the volume from the directory path

    void createEmpty(int width, int height, int depth);

    int getWidth() const;

    int getHeight() const;

    int getDepth() const;

    unsigned char getVoxel(int x, int y, int z) const;

    void setVoxel(unsigned char v, int x, int y, int z);

    void updateSlice(int z, Image newData);

    void saveVolume(const std::string& outputDirectory) const;

    // Added methods
    Image sliceXY(int zCoordinate) const;

    Image sliceXZ(int yCoordinate) const;

    Image sliceYZ(int xCoordinate) const;

    // Inline definition of getSlices
    const std::vector<Image>& getSlices() const { return slices; }

private:
    int width, height, depth;
    std::vector<Image> slices; // store the slices of the volume
};

#endif
```

# Index