

# ParallelPic

1.0

Generated by Doxygen 1.8.4

Mon Dec 9 2013 01:36:14



# Contents

<b>1</b>	<b>Class Index</b>	<b>1</b>
1.1	Class List	1
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List	3
<b>3</b>	<b>Class Documentation</b>	<b>5</b>
3.1	Image Class Reference	5
3.1.1	Detailed Description	7
3.1.2	Constructor & Destructor Documentation	8
3.1.2.1	Image	8
3.1.2.2	Image	8
3.1.2.3	Image	8
3.1.2.4	~Image	9
3.1.3	Member Function Documentation	9
3.1.3.1	autocovariance	9
3.1.3.2	average_omp	10
3.1.3.3	binarize_img	10
3.1.3.4	color_slicing	10
3.1.3.5	coorrelogram	11
3.1.3.6	coorrelogram_par	12
3.1.3.7	coorrelogram_ZC	12
3.1.3.8	display	13
3.1.3.9	filter_average	13
3.1.3.10	filter_dynamic_range_dilatation	14
3.1.3.11	filter_edge_enhancement_displacement	14
3.1.3.12	filter_freeman_0	15
3.1.3.13	filter_freeman_1	15
3.1.3.14	filter_freeman_2	16

3.1.3.15	<a href="#">filter_freeman_3</a>	16
3.1.3.16	<a href="#">filter_freeman_4</a>	17
3.1.3.17	<a href="#">filter_freeman_5</a>	18
3.1.3.18	<a href="#">filter_freeman_6</a>	18
3.1.3.19	<a href="#">filter_freeman_7</a>	19
3.1.3.20	<a href="#">filter_gaussian</a>	19
3.1.3.21	<a href="#">filter_Gradient_horizontal</a>	20
3.1.3.22	<a href="#">filter_Gradient_vertical</a>	21
3.1.3.23	<a href="#">filter_horizontal_borders</a>	21
3.1.3.24	<a href="#">filter_kirsch_0</a>	22
3.1.3.25	<a href="#">filter_kirsch_135</a>	22
3.1.3.26	<a href="#">filter_kirsch_180</a>	23
3.1.3.27	<a href="#">filter_kirsch_225</a>	24
3.1.3.28	<a href="#">filter_kirsch_270</a>	24
3.1.3.29	<a href="#">filter_kirsch_315</a>	25
3.1.3.30	<a href="#">filter_kirsch_45</a>	25
3.1.3.31	<a href="#">filter_kirsch_90</a>	26
3.1.3.32	<a href="#">filter_Laplacian</a>	26
3.1.3.33	<a href="#">filter_Laplacian_no_diagonal</a>	27
3.1.3.34	<a href="#">filter_maximum</a>	28
3.1.3.35	<a href="#">filter_median</a>	28
3.1.3.36	<a href="#">filter_minimum</a>	29
3.1.3.37	<a href="#">filter_modal</a>	30
3.1.3.38	<a href="#">filter_order_statistics</a>	31
3.1.3.39	<a href="#">filter_Prewitt_E_W</a>	32
3.1.3.40	<a href="#">filter_Prewitt_N_S</a>	33
3.1.3.41	<a href="#">filter_Prewitt_NE_SW</a>	33
3.1.3.42	<a href="#">filter_Prewitt_NW_SE</a>	34
3.1.3.43	<a href="#">filter_vertical_borders</a>	34
3.1.3.44	<a href="#">gaussian_noise</a>	35
3.1.3.45	<a href="#">get_histogram</a>	35
3.1.3.46	<a href="#">gray_scale</a>	36
3.1.3.47	<a href="#">histogram_equalization</a>	36
3.1.3.48	<a href="#">interpolation</a>	36
3.1.3.49	<a href="#">inverse</a>	37
3.1.3.50	<a href="#">log_transformation</a>	37
3.1.3.51	<a href="#">median_omp</a>	38

3.1.3.52	<a href="#">multiply_img</a>	38
3.1.3.53	<a href="#">plot_histogram</a>	39
3.1.3.54	<a href="#">plot_histogram_equalization</a>	40
3.1.3.55	<a href="#">power_law_transformation</a>	40
3.1.3.56	<a href="#">rgb_hsv</a>	41
3.1.3.57	<a href="#">salt_pepper</a>	41
3.1.3.58	<a href="#">set_pixel_value</a>	41
3.1.3.59	<a href="#">subtract_img</a>	41
3.1.3.60	<a href="#">sum_img</a>	42
3.1.3.61	<a href="#">variance</a>	43
<b>4</b>	<b>File Documentation</b>	<b>45</b>
4.1	<a href="#">/home/fish/Documents/ParallelPic/Proyecto/include/ParallelPic.hh File Reference</a>	45
4.2	<a href="#">ParallelPic.hh</a>	45
4.3	<a href="#">/home/fish/Documents/ParallelPic/Proyecto/src/image.cpp File Reference</a>	48
4.4	<a href="#">image.cpp</a>	48
4.5	<a href="#">/home/fish/Documents/ParallelPic/Proyecto/src/ParallelPic.cpp File Reference</a>	73
4.6	<a href="#">ParallelPic.cpp</a>	73
	<b>Index</b>	<b>104</b>



# Chapter 1

## Class Index

### 1.1 Class List

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

<a href="#">Image</a>	Abstraction of the image, contains a CImg object that defines the handling of the image . . . . .	<a href="#">5</a>
-----------------------	---	-------------------





## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

/home/fish/Documents/ParallelPic/Proyecto/include/ <a href="#">ParallelPic.hh</a> . . . . .	45
/home/fish/Documents/ParallelPic/Proyecto/src/ <a href="#">image.cpp</a> . . . . .	48
/home/fish/Documents/ParallelPic/Proyecto/src/ <a href="#">ParallelPic.cpp</a> . . . . .	73



## Chapter 3

# Class Documentation

### 3.1 Image Class Reference

The `Image` class is the abstraction of the image, contains a `CImg` object that defines the handling of the image.

```
#include <ParallelPic.hh>
```

#### Public Member Functions

- `Image ()`  
*Constructor This constructor initializes the four dimension params at 0; The `Img` calls the constructor of `CImg` to create an empty image.*
- `Image (const char *const filename)`  
*`Image()` is the constructor of the image used when the image doesn't been be created.*
- `Image (const unsigned int width, const unsigned int height, const unsigned int depth, const unsigned int spectrum, int value)`  
*`Image (const char *const filename)` this constructor is used when the image already exist's and is stored in.*
- `~Image (void)`
- void `display (const char *message)`
- void `set_pixel_value (int x, int y, int z, int c, unsigned char)`
- `Image subtract_img (Image, int)`  
*This function subtracts the pixel values of two images, that can be used to see the differences between them.*
- `Image sum_img (Image, int)`
- `Image multiply_img (double, int)`  
*This function multiplies the pixel values by a factor. If the pixel value is higher than 255, adjust the pixel value to 255.*
- `Image binarize_img (unsigned int, int)`
- `Image filter_Laplacian (int)`  
*Returns an image after applying the Laplacian filter to the image. Considers the diagonal values This function applies a convolution with this kernel:  $((1, 1, 1), (1, -8, 1), (1, 1, 1))$ .*
- `Image filter_Laplacian_no_diagonal (int)`
- `Image filter_Gradient_vertical (int)`
- `Image filter_Gradient_horizontal (int)`
- `Image filter_Prewitt_N_S (int)`
- `Image filter_Prewitt_NE_SW (int)`
- `Image filter_Prewitt_E_W (int)`

- [Image filter\\_Prewitt\\_NW\\_SE](#) (int)
- [Image filter\\_edge\\_enhancement\\_displacement](#) (unsigned int, unsigned int, int)
- [Image filter\\_horizontal\\_borders](#) (int)
- [Image filter\\_vertical\\_borders](#) (int)
- [Image filter\\_median](#) (int, int)
 

*This function calculates the median of the range of pixels into the kernel and sets this value in the central pixel of the kernel.*
- [Image filter\\_average](#) (int, int)
 

*This function calculates the average of the range of pixels into the kernel and sets this value in the central pixel of the kernel.*
- [Image average\\_omp](#) (int, int)
- [Image filter\\_gaussian](#) (int, int, int)
 

*This function applies a gaussian kernel trough the hole image.*
- [Image filter\\_modal](#) (int, int)
 

*This function calculates the modal of the range of pixels into the kernel and sets this value in the central pixel of the kernel.*
- [Image median\\_omp](#) (int, int)
- [Image filter\\_dynamic\\_range\\_dilatation](#) (unsigned char, unsigned char, double, double, double, int)
 

*All the pixel values are divided in 3 ranges, and each range suffer a diferent transformation. This function is used to transform the range of lower pixel values in medium values and the higher too, to smooth the image.*
- [Image inverse](#) (int)
- [Image log\\_transformation](#) (int)
- [Image power\\_law\\_transformation](#) (double exponent, int)
 

*Executes this transformation:  $v(x, y, z, c) = c \log(u(x, y, z, c) + 1)$ .*
- [Image color\\_slicing](#) (unsigned char[], unsigned char[], unsigned char[], int)
 

*Highlights the desired colors, between the two colors given as parameters.*
- int \* [get\\_histogram](#) (unsigned int c, unsigned int z)
- void [plot\\_histogram](#) (int, const char \*title)
 

*This function plot the histogram, using the CImg histogram function.*
- int \* [histogram\\_equalization](#) (int \*, const char \*title)
- CImg< float > [autocovariance](#) (int, int, int)
 

*Calculates the autocovariance matrix for the image. The covariance, calculates the covariance matrix of an image. This function calculates something similar to the function below:  $g(x, y) = \sum_{n=0}^N \sum_{m=1}^M \left( f(x, y) - \overline{f(x, y)} \right) \left( f(x + \Delta x, y + \Delta y) - \overline{f(x + \Delta x, y + \Delta y)} \right)$*

*Where it calculates the variation between two series, one is the normal one, and the other is displaced by two parameters \$x\$ & \$y\$. For an image it its calculated for a neighborhood around each pixel.*
- void [plot\\_histogram\\_equalization](#) (int, const char \*title)
- [Image filter\\_order\\_statistics](#) (int dim, int order, int)
- [Image variance](#) (int, int)
 

*This function compute the variance of an image. The variance is given by the summation of the average multiplied by the subtraction of the average with the pixel value, squared.*
- [Image filter\\_kirsch\\_0](#) (int)
 

*Applies the kirsch mask at 0°: (-3, -3, 5)(-3, 0, 5)(-3, -3, 5).*
- [Image filter\\_kirsch\\_45](#) (int)
 

*Applies the kirsch mask at 45°: (-3, 5, 5)(-3, 0, 5)(-3, -3, -3).*
- [Image filter\\_kirsch\\_90](#) (int)
 

*Applies the kirsch mask at 90°: (5, 5, 5)(-3, 0 - 3)(-3, -3, -3).*
- [Image filter\\_kirsch\\_135](#) (int)
 

*Applies the kirsch mask at 135°: (5, 5, -3)(5, 0, -3)(-3, -3, -3).*
- [Image filter\\_kirsch\\_180](#) (int)
 

*Applies the kirsch mask at 180°: (5, -3, -3)(5, 0, -3)(5, -3, -3).*

- [Image filter\\_kirsch\\_225](#) (int)  
*Applies the kirsch mask at 225°:  $(-3, -3, -3)(5, 0, -3)(5, 5, -3)$ .*
- [Image filter\\_kirsch\\_270](#) (int)  
*Applies the kirsch mask at 270°:  $(-3, -3, -3)(-3, 0, -3)(5, 5, 5)$ .*
- [Image filter\\_kirsch\\_315](#) (int)  
*Applies the kirsch mask at 315°:  $(-3, -3, -3)(-3, 0, 5)(-3, 5, 5)$ .*
- [Image filter\\_freeman\\_0](#) (int)  
*Applies the freeman mask  $(1, 1, 1)(1, -2, 1)(1, -1, -1)$ .*
- [Image filter\\_freeman\\_1](#) (int)  
*Applies the freeman mask  $(1, 1, 1)(-1, -2, 1)(1, -1, 1)$ .*
- [Image filter\\_freeman\\_2](#) (int)  
*Applies the freeman mask  $(-1, 1, 1)(-1, -2, 1)(1, 1, 1)$ .*
- [Image filter\\_freeman\\_3](#) (int)  
*Applies the freeman mask  $(-1, -1, 1)(-1, -2, 1)(1, 1, 1)$ .*
- [Image filter\\_freeman\\_4](#) (int)  
*Applies the freeman mask  $(-1, -1, -1)(1, -2, 1)(1, 1, 1)$ .*
- [Image filter\\_freeman\\_5](#) (int)  
*Applies the freeman mask  $(1, -1, -1)(1, -2, -1)(1, 1, 1)$ .*
- [Image filter\\_freeman\\_6](#) (int)  
*Applies the freeman mask  $(1, 1, -1)(1, -2, -1)(1, 1, -1)$ .*
- [Image filter\\_freeman\\_7](#) (int)  
*Applies the freeman mask  $(1, 1, 1)(1, -2, -1)(1, -1, -1)$ .*
- [Image filter\\_maximum](#) (int)  
*Assigns the highest value in the neighborhood. Assigns the highest value in the neighborhood around the desired pixel.*
- [Image filter\\_minimum](#) (int)  
*Assigns the lowest value in the neighborhood.*
- [Image gray\\_scale](#) (int)  
*Assigns the lowest value in the neighborhood.*
- void [gaussian\\_noise](#) (double, int)  
*Converts an RGB image to gray scale.*
- void [salt\\_pepper](#) (double, int)  
*Put pepper (black pixels) and salt(white pixels)*
- [Image interpolation](#) (int)  
*This function doubles the size of the image and use the closer neighborhood interpolation.*
- [Image coorrelogram](#) (unsigned int, unsigned int, int)  
*This function compute the coorrelogram of an image.*
- [Image coorrelogram\\_ZC](#) (unsigned int, unsigned int, unsigned int, unsigned int, int)
- [Image coorrelogram\\_par](#) (unsigned int, unsigned int, unsigned int, unsigned int)
- [Image rgb\\_hsv](#) ()

### 3.1.1 Detailed Description

The [Image](#) class is the abstraction of the image, contains a CImg object that defines the handling of the image.

This class have the attributes of the four dimensions of the image: height, width, depth and spectrum

Definition at line 21 of file [ParallelPic.hh](#).

### 3.1.2 Constructor & Destructor Documentation

#### 3.1.2.1 Image::Image ( )

Constructor This constructor initializes the four dimension params at 0; The `Img` calls the constructor of `CImg` to create an empty image.

Definition at line 17 of file [image.cpp](#).

```
00018 {
00019     this->Img = new CImg<unsigned char>();
00020     this->width = 0;
00021     this->height = 0;
00022     this->depth = 0;
00023     this->spectrum = 0;
00024
00025 }
```

#### 3.1.2.2 Image::Image ( const char \*const filename )

[Image\(\)](#) is the constructor of the image used when the image doesn't been be created.

Constructor.

This constructor is used when the image already exist's and is stored in the

Parameters

<i>filename</i>	path.
-----------------	-------

Parameters

<i>&lt;img&gt;</i>	is a var of type <code>Cimg</code> that is treated like an unsigned char.
<i>&lt;width&gt;</i>	refers to the number of columns of pixels in the image.
<i>&lt;height&gt;</i>	refers to the number of rows of pixels in the image.
<i>&lt;depth&gt;</i>	is the amount of layers of depth the image has, usually is one, except for 3D images.
<i>&lt;spectrum&gt;</i>	is the number of channels in the image, RGB has a spectrum of 3, a monochromatic image has a spectrum of 1.

Definition at line 32 of file [image.cpp](#).

```
00033 {
00034     this->Img = new CImg<unsigned char>(filename);
00036     this->width = this->Img->width();
00038     this->height = this->Img->height();
00040     this->depth = this->Img->depth();
00042     this->spectrum = this->Img->spectrum();
00044 }
```

#### 3.1.2.3 Image::Image ( const unsigned int width, const unsigned int height, const unsigned int depth, const unsigned int spectrum, int value )

[Image](#) (const char \*const filename) this constructor is used when the image already exist's and is stored in.

This constructor is used when we need to create an image, and gives the dimensions of the image, and the value of a color that fills all the pixels.

## Parameters

<i>filename.</i>	
------------------	--

Definition at line 49 of file [image.cpp](#).

```

00050 {
00051     this->Img = new CImg<unsigned char>(width, height, depth, spectrum, value);
00052     this->width = width;
00053     this->height = height;
00054     this->depth = depth;
00055     this->spectrum = spectrum;
00056 }
```

## 3.1.2.4 Image::~Image ( void )

[Image](#) (const unsigned int width, const unsigned int height, const unsigned int depth, const unsigned int spectrum, int value) This constructor is used when we try to create an image with the specified dimensions and the parameter

## Parameters

<i>value</i>	is the value of the color of all the pixels of the image.
--------------	---

Definition at line 60 of file [image.cpp](#).

```

00061 {
00062
00063 }
```

## 3.1.3 Member Function Documentation

## 3.1.3.1 CImg&lt; float &gt; Image::autocovariance ( int hor\_dis, int ver\_dis, int num\_threads )

Calculates the autocovariance matrix for the image. The covariance, calculates the covariance matrix of an image. This function calculates something similar to the function below:  $g(x,y) = \sum_{n=0}^N \sum_{m=1}^M \left( f(x,y) - \overline{f(x,y)} \right) \left( f(x+\Delta x, y+\Delta y) - \overline{f(x+\Delta x, y+\Delta y)} \right)$  Where it calculates the variation between two series, one is the normal one, and the other is displaced by two parameters \$x\$ & \$y\$. For an image it its calculated for a neighborhood around each pixel.

## Returns

A CImg object, because it must contain float values.

Definition at line 2047 of file [image.cpp](#).

```

02048 {
02049     CImg<float> autocovariance (this->get_width() , this->get_height() , this->get_depth() ,
                                this->get_spectrum(), 0);
02050
02051     Image average = this->filter_average(1);
02052
02053     for(unsigned int c = 0; c < this->get_spectrum(); c++)
02054     {
02055         for(unsigned int z = 0; z < this->get_depth(); z++)
02056         {
02057             for(unsigned int x = 3+hor_dis; x < this->get_width()-(3+hor_dis); x++)
02058             {
02059                 for(unsigned int y = 3+ver_dis; y < this->get_height()-(3+ver_dis); y++)
02060                 {
02061                     int sum = 0;
02062                     for(unsigned int i = x-3; i<x+4; i++)
02063                     {
```

```

02064         for(unsigned int j= y-3; j<y+4; j++)
02065         {
02066             sum += ( (this->get_pixel_value(i,j,z,c)) - average.get_pixel_value(i,j,z,c))
* ( (this->get_pixel_value(i+hor_dis,j+ver_dis,z,c)) - average.get_pixel_value(i+hor_dis,j+ver_dis,z,c))
;
02067         }
02068     }
02069
02070     autocovariance(x,y,z,c) = sum/49;
02071 }
02072 }
02073 }
02074 }
02075 return autocovariance;
02076 }

```

### 3.1.3.2 Image Image::average\_omp ( int , int )

### 3.1.3.3 Image Image::binarize\_img ( unsigned int *cutoff\_value*, int *num\_threads* )

Definition at line 301 of file [image.cpp](#).

```

00302 {
00303     Image result (this->get_width() , this->get_height() , this->get_depth() , this->get_spectrum(), 0);
00304     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00305     {
00306         for(unsigned int z = 0; z < this->get_depth(); z++)
00307         {
00308             for(unsigned int x = 0; x < this->get_width(); x++)
00309             {
00310                 for(unsigned int y = 0; y < this->get_height(); y++)
00311                 {
00312                     unsigned char pixel= static_cast<unsigned int>(this->get_pixel_value(x,y,z,c));
00313                     if(pixel >= cutoff_value)
00314                         pixel=255;
00315                     else
00316                         pixel=0;
00317                     result.set_pixel_value(x,y,z,c,pixel);
00318                 }
00319             }
00320         }
00321     }
00322 }
00323
00324
00325 return result;
00326 }

```

### 3.1.3.4 Image Image::color\_slicing ( unsigned char *color1*[], unsigned char *color2*[], unsigned char *neutral*[], int *num\_threads* )

Highlights the desired colors, between the two colors given as parameters.

#### Attention

{The colors must be unsigned char, and must be the size of the spectrum of the image (Number of channels), 3 in case o RGB images}

#### Parameters

<i>unsigned</i>	char color1[]: The start color of the color slicing.
-----------------	--



<i>unsigned</i>	char color2[]: The end color of the color slicing.
<i>unsigned</i>	char neutral: The intensity every other pixels that are not between the given colors will be set to.

Definition at line 1124 of file [image.cpp](#).

```

01125 {
01126     Image filtered (this->get_width() , this->get_height() , this->get_depth() , this->get_spectrum() , 0
01127 );
01128     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01129     {
01130         for(unsigned int z = 0; z < this->get_depth(); z++)
01131         {
01132             for(unsigned int x = 0; x < this->get_width(); x++)
01133             {
01134                 for(unsigned int y = 0; y < this->get_height(); y++)
01135                 {
01136                     unsigned char pixel = this->get_pixel_value(x,y,z,c);
01137                     if(pixel > color1[c] && pixel < color2[c] )
01138                     {
01139                         filtered.set_pixel_value(x, y, z, c, pixel);
01140                     }
01141                     else
01142                     {
01143                         filtered.set_pixel_value(x,y,z,c, neutral[c]);
01144                     }
01145                 }
01146             }
01147         }
01148     }
01149 }
01150 }
01151
01152 return filtered;
01153
01154 }
```

### 3.1.3.5 Image Image::coorrelogram ( unsigned int ver, unsigned int hor, int num\_threads )

This function compute the coorrelogram of an image.

This function compute the coorrelogram of a specified depth and spectrum of an image.

#### Parameters

<i>unsigned</i>	int ver is the vertical distance of the original pixel that we use to compute the coorrelogram.
<i>unsigned</i>	int hor is the horizontal distance of the original pixel that we use to compute the coorrelogram.

#### Returns

This function returns the coorrelogram image.

#### Parameters

<i>unsigned</i>	int ver is the vertical distance of the original pixel that we use to compute the coorrelogram.
<i>unsigned</i>	int hor is the horizontal distance of the original pixel that we use to compute the coorrelogram.
<i>unsigned</i>	int z is the specified depth of the image that will be obtained the coorrelogram.
<i>unsigned</i>	int c is the specified spectrum of the image that will be obtained the coorrelogram.

#### Returns

This function returns the coorrelogram image.

Definition at line 2164 of file [image.cpp](#).

```

02165 {
02166     Image result (256,256, 1, 1, 0);
02167     for(unsigned int i = 0; i < 256; i++)
02168     {
02169         for(unsigned int j=0; j< 256; j++)
02170         {
02171             unsigned int pixel = 0;
02172             for(unsigned int x=0; x< (this->get_width()-hor);++x)
02173             {
02174                 for(unsigned int y=0; y< (this->get_height()-ver);++y)
02175                 {
02176                     unsigned char first = (this->get_pixel_value(x,y,0,0));
02177                     unsigned char secnd = (this->get_pixel_value(x+hor, y+ver, 0, 0));
02178                     if(first == i && secnd == j)
02179                     {
02180                         pixel ++;
02181                     }
02182                 }
02183             }
02184             if(pixel>255)
02185             {
02186                 pixel=255;
02187             }
02188             result.set_pixel_value(i, j, 0, 0, pixel);
02189         }
02190     }
02191     cout<<"\n"<<i<<"\n"<<endl;
02192 }
02193 return result;
02194 }
02195 }
02196 }
02197 }
02198 }
02199 }
02200 }
02201 }
02202 }

```

**3.1.3.6** Image Image::coorrelogram\_par ( unsigned int, unsigned int, unsigned int, unsigned int )

**3.1.3.7** Image Image::coorrelogram\_ZC ( unsigned int ver, unsigned int hor, unsigned int z, unsigned int c, int num\_threads )

Definition at line 2212 of file [image.cpp](#).

```

02213 {
02214     Image result (256,256, 1, 1, 0);
02215     for(unsigned int i = 0; i < 256; i++)
02216     {
02217         for(unsigned int j=0; j< 256; j++)
02218         {
02219             unsigned int pixel = 0;
02220             for(unsigned int x=0; x< (this->get_width()-hor);++x)
02221             {
02222                 for(unsigned int y=0; y< (this->get_height()-ver);++y)
02223                 {
02224                     unsigned char first = (this->get_pixel_value(x,y,z,c));
02225                     unsigned char secnd = (this->get_pixel_value(x+hor, y+ver, z, c));
02226                     if(first == i && secnd == j)
02227                     {
02228                         pixel ++;
02229                     }
02230                 }
02231             }
02232             if(pixel>255)
02233             {
02234                 pixel=255;
02235             }
02236             result.set_pixel_value(i, j, 0, 0, pixel);
02237         }
02238     }
02239 }
02240 }
02241 }
02242 }

```

```

02243     }
02244
02245     }
02246     return result;
02247
02248 }

```

### 3.1.3.8 void Image::display ( const char \* message )

Definition at line 77 of file [image.cpp](#).

```

00078 {
00079     CImgDisplay main (*(this->Img), message);
00080     while (!main.is_closed())
00081     {
00082         main.wait();
00083     }
00084 }

```

### 3.1.3.9 Image Image::filter\_average ( int dim, int num\_threads )

This function calculates the average of the range of pixels into the kernel and sets this value in the central pixel of the kernel.

#### Parameters

<i>Only</i>	receives the dimension of the kernel (dim), wich only can be an impair number.
-------------	--

#### Returns

[Image](#) filtered which is the image with the average filter applied.

Definition at line 803 of file [image.cpp](#).

Referenced by [autocovariance\(\)](#).

```

00804 {
00805     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00806 );
00807     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00808     {
00809         for(unsigned int z = 0; z < this->get_depth(); z++)
00810         {
00811             for(unsigned int x = dim; x < this->get_width()-dim; x++)
00812             {
00813                 for(unsigned int y = dim; y < this->get_height()-dim; y++)
00814                 {
00815                     int sum = 0;
00816                     for(unsigned int i = x-dim; i<= x+dim; i++)
00817                     {
00818                         for(unsigned int j = y-dim; j<= y+dim; j++)
00819                         {
00820                             sum += this->get_pixel_value(i, j, z, c);
00821                         }
00822                     }
00823                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (sum/((dim*2+1)*(dim*2+
00824 1)));
00825                     filtered.set_pixel_value(x, y, z, c, pixel);
00826                 }
00827             }
00828         }
00829     }
00830 }
00831 }
00832 return filtered;
00833 }

```

### 3.1.3.10 Image Image::filter\_dynamic\_range\_dilatation ( unsigned char *a*, unsigned char *b*, double *alpha*, double *beta*, double *gamma*, int *num\_threads* )

All the pixel values are divided in 3 ranges, and each range suffer a diferent transformation. This function is used to transform the range of lower pixel values in medium values and the higher too, to smooth the image.

#### Parameters

<i>unsigned</i>	char a is the first cutoff pixel value.
<i>unsigned</i>	char b is the second cutoff pixel value.
<i>double</i>	alpha is the first multiplier.
<i>double</i>	beta is the second multiplier.
<i>double</i>	gamma is the third multiplier.

#### Returns

An image object that contains the dilatated image.

Definition at line 1052 of file [image.cpp](#).

```

01053 {
01054     Image filtered (this->get_width() , this->get_height() , this->get_depth() , this->get_spectrum() , 0
    );
01055
01056     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01057     {
01058         for(unsigned int z = 0; z < this->get_depth(); z++)
01059         {
01060             for(unsigned int x = 0; x < this->get_width(); x++)
01061             {
01062                 for(unsigned int y = 0; y < this->get_height(); y++)
01063                 {
01064                     unsigned char pixel=0;
01065
01066                     if(this->get_pixel_value(x,y,z,c)<a)
01067                         pixel =abs(alpha*this->get_pixel_value(x,y,z,c));
01068
01069                     else if(this->get_pixel_value(x,y,z,c)>=a && this->get_pixel_value(x,y,z,c)<b)
01070                         pixel=abs(beta*(this->get_pixel_value(x,y,z,c)-a)+alpha*a);
01071
01072                     else if(this->get_pixel_value(x,y,z,c)<=b)
01073
01074                         pixel=abs(gamma*(this->get_pixel_value(x,y,z,c)-b)+((beta*(b-a))+alpha*a));
01075                     filtered.set_pixel_value(x,y,z,c,static_cast<unsigned int>(pixel));
01076                 }
01077             }
01078         }
01079     }
01080     return filtered;
01081 }
```

### 3.1.3.11 Image Image::filter\_edge\_enhacement\_displacement ( unsigned int *horizontal\_dis*, unsigned int *vertical\_dis*, int *num\_threads* )

Definition at line 662 of file [image.cpp](#).

```

00663 {
00664     Image result (this->get_width() , this->get_height() , this->get_depth() , this->get_spectrum() , 0);
00665
00666     if((horizontal_dis < this->get_width()) && (vertical_dis < this->get_height()))
00667     {
00668         for(unsigned int c = 0; c < this->get_spectrum(); c++)
00669         {
00670             for(unsigned int z = 0; z < this->get_depth(); z++)
00671             {
00672                 for(unsigned int x = horizontal_dis; x < this->get_width(); x++)
```

```

00672         {
00673             for(unsigned int y = vertical_dis; y < this->get_height(); y++)
00674             {
00675                 unsigned char value = static_cast<unsigned char>(abs(this->get_pixel_value(x,y,z,c)
- this->get_pixel_value(x-horizontal_dis, y-vertical_dis, z, c)));
00676
00677                 result.set_pixel_value(x,y,z,c, value);
00678             }
00679         }
00680     }
00681 }
00682 }
00683 return result;
00684 }

```

### 3.1.3.12 Image Image::filter\_freeman\_0 ( int num\_threads )

Applies the freeman mask  $(1, 1, 1)(1, -2, 1)(1, -1, -1)$ .

Definition at line 1510 of file [image.cpp](#).

```

01511 {
01512     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
01513
01514     int m = 1;
01515
01516     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01517     {
01518         for(unsigned int z = 0; z < this->get_depth(); z++)
01519         {
01520             for(unsigned int x = m; x < this->get_width()-m; x++)
01521             {
01522                 for(unsigned int y = m; y < this->get_height()-m; y++)
01523                 {
01524                     int sum = (get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x-1, y, z, c)+
get_pixel_value(x-1, y+1, z, c)+get_pixel_value(x, y-1, z, c)+get_pixel_value(x+1, y-1, z, c)+get_pixel_value(x+1,y,z,c))-
1*(get_pixel_value(x+1, y+1, z, c)+get_pixel_value(x, y+1, z, c))+2*get_pixel_value(x, y, z, c);
01525                     if (sum > 255 || sum < -255)
01526                     {
01527                         sum = 255;
01528                     }
01529                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01530                     filtered.set_pixel_value(x, y, z, c, pixel);
01531                 }
01532             }
01533         }
01534     }
01535 }
01536 }
01537
01538 return filtered;
01539
01540 }

```

### 3.1.3.13 Image Image::filter\_freeman\_1 ( int num\_threads )

Applies the freeman mask  $(1, 1, 1)(-1, -2, 1)(1, -1, 1)$ .

Definition at line 1545 of file [image.cpp](#).

```

01546 {
01547     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
01548
01549     int m = 1;
01550
01551     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01552     {
01553         for(unsigned int z = 0; z < this->get_depth(); z++)

```

```

01554     {
01555         for(unsigned int x = m; x < this->get_width()-m; x++)
01556         {
01557             for(unsigned int y = m; y < this->get_height()-m; y++)
01558             {
01559                 int sum = (get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x+1, y+1, z, c)+
get_pixel_value(x-1, y+1, z, c)+get_pixel_value(x, y-1, z, c)+get_pixel_value(x+1, y-1, z, c)+get_pixel_value(x+1,y,z,c)
)-1*(get_pixel_value(x-1, y, z, c)+get_pixel_value(x, y+1, z, c))+2*get_pixel_value(x, y, z, c);
01560                 if (sum > 255 || sum < -255)
01561                 {
01562                     sum = 255;
01563                 }
01564                 unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01565                 filtered.set_pixel_value(x, y, z, c, pixel);
01566             }
01567         }
01568     }
01569 }
01570 }
01571 }
01572
01573 return filtered;
01574
01575 }

```

### 3.1.3.14 Image Image::filter\_freeman\_2 ( int num\_threads )

Applies the freeman mask  $\begin{pmatrix} -1, 1, 1 \\ -1, -2, 1 \\ 1, 1, 1 \end{pmatrix}$ .

Definition at line 1580 of file [image.cpp](#).

```

01581 {
01582     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
01583
01584     int m = 1;
01585
01586     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01587     {
01588         for(unsigned int z = 0; z < this->get_depth(); z++)
01589         {
01590             for(unsigned int x = m; x < this->get_width()-m; x++)
01591             {
01592                 for(unsigned int y = m; y < this->get_height()-m; y++)
01593                 {
01594                     int sum = (get_pixel_value(x, y-1, z, c)+get_pixel_value(x+1, y-1, z, c)+
get_pixel_value(x+1, y, z, c)+get_pixel_value(x+1, y+1, z, c)+get_pixel_value(x, y+1, z, c)+get_pixel_value(x-1,y+1,z,c))-
1*(get_pixel_value(x-1, y, z, c)+get_pixel_value(x-1, y-1, z, c))+2*get_pixel_value(x, y, z, c);
01595                     if (sum > 255 || sum < -255)
01596                     {
01597                         sum = 255;
01598                     }
01599                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01600                     filtered.set_pixel_value(x, y, z, c, pixel);
01601                 }
01602             }
01603         }
01604     }
01605 }
01606
01607 return filtered;
01608
01609 }
01610 }

```

### 3.1.3.15 Image Image::filter\_freeman\_3 ( int num\_threads )

Applies the freeman mask  $\begin{pmatrix} -1, -1, 1 \\ -1, -2, 1 \\ 1, 1, 1 \end{pmatrix}$ .

Definition at line 1615 of file [image.cpp](#).

```

01616 {
01617     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01618 );
01619     int m = 1;
01620     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01621     {
01622         for(unsigned int z = 0; z < this->get_depth(); z++)
01623         {
01624             for(unsigned int x = m; x < this->get_width()-m; x++)
01625             {
01626                 for(unsigned int y = m; y < this->get_height()-m; y++)
01627                 {
01628                     int sum = (get_pixel_value(x+1, y-1, z, c)+get_pixel_value(x+1, y, z, c)+
01629 get_pixel_value(x+1, y+1, z, c)+get_pixel_value(x, y+1, z, c)+get_pixel_value(x-1,y+1,z,c))-1*(get_pixel_value(x-1, y, z,
01630 c)+get_pixel_value(x, y-1, z, c)+get_pixel_value(x-1,y-1,z,c))+2*get_pixel_value(x, y, z, c);
01631                     if (sum > 255 || sum < -255)
01632                     {
01633                         sum = 255;
01634                     }
01635                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01636                     filtered.set_pixel_value(x, y, z, c, pixel);
01637                 }
01638             }
01639         }
01640     }
01641 }
01642
01643 return filtered;
01644
01645 }

```

### 3.1.3.16 Image Image::filter\_freeman\_4 ( int num\_threads )

Applies the freeman mask  $(-1, -1, -1)(1, -2, 1)(1, 1, 1)$ .

Definition at line 1650 of file [image.cpp](#).

```

01651 {
01652     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01653 );
01654     int m = 1;
01655     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01656     {
01657         for(unsigned int z = 0; z < this->get_depth(); z++)
01658         {
01659             for(unsigned int x = m; x < this->get_width()-m; x++)
01660             {
01661                 for(unsigned int y = m; y < this->get_height()-m; y++)
01662                 {
01663                     int sum = (get_pixel_value(x-1, y, z, c)+get_pixel_value(x-1, y+1, z, c)+
01664 get_pixel_value(x, y+1, z, c)+get_pixel_value(x+1, y+1, z, c)+get_pixel_value(x+1, y, z, c))-1*(get_pixel_value(x-1, y-1,
01665 z, c)+get_pixel_value(x, y-1, z, c)+get_pixel_value(x+1,y-1,z,c))+2*get_pixel_value(x, y, z, c);
01666                     if (sum > 255 || sum < -255)
01667                     {
01668                         sum = 255;
01669                     }
01670                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01671                     filtered.set_pixel_value(x, y, z, c, pixel);
01672                 }
01673             }
01674         }
01675     }
01676 }
01677
01678 return filtered;
01679
01680 }

```

### 3.1.3.17 Image Image::filter\_freeman\_5 ( int num\_threads )

Applies the freeman mask  $(1, -1, -1)(1, -2, -1)(1, 1, 1)$ .

Definition at line 1686 of file [image.cpp](#).

```

01687 {
01688     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01689 );
01690     int m = 1;
01691     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01692     {
01693         for(unsigned int z = 0; z < this->get_depth(); z++)
01694         {
01695             for(unsigned int x = m; x < this->get_width()-m; x++)
01696             {
01697                 for(unsigned int y = m; y < this->get_height()-m; y++)
01698                 {
01699                     int sum = (get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x-1, y, z, c)+
01700 get_pixel_value(x-1, y+1, z, c)+get_pixel_value(x, y+1, z, c)+get_pixel_value(x+1, y+1, z, c))-1*(get_pixel_value(x+1, y,
01701 z, c)+get_pixel_value(x, y-1, z, c)+get_pixel_value(x+1,y-1,z,c))+-2*get_pixel_value(x, y, z, c);
01702                     if (sum > 255 || sum < -255)
01703                     {
01704                         sum = 255;
01705                     }
01706                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01707                     filtered.set_pixel_value(x, y, z, c, pixel);
01708                 }
01709             }
01710         }
01711     }
01712 }
01713
01714 return filtered;
01715
01716 }
```

### 3.1.3.18 Image Image::filter\_freeman\_6 ( int num\_threads )

Applies the freeman mask  $(1, 1, -1)(1, -2, -1)(1, 1, -1)$ .

Definition at line 1722 of file [image.cpp](#).

```

01723 {
01724     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01725 );
01726     int m = 1;
01727     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01728     {
01729         for(unsigned int z = 0; z < this->get_depth(); z++)
01730         {
01731             for(unsigned int x = m; x < this->get_width()-m; x++)
01732             {
01733                 for(unsigned int y = m; y < this->get_height()-m; y++)
01734                 {
01735                     int sum = (get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x-1, y, z, c)+
01736 get_pixel_value(x-1, y+1, z, c)+get_pixel_value(x, y+1, z, c)+get_pixel_value(x, y-1, z, c))-1*(get_pixel_value(x+1, y-1,
01737 z, c)+get_pixel_value(x+1, y, z, c)+get_pixel_value(x+1,y+1,z,c))+-2*get_pixel_value(x, y, z, c);
01738                     if (sum > 255 || sum < -255)
01739                     {
01740                         sum = 255;
01741                     }
01742                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01743                     filtered.set_pixel_value(x, y, z, c, pixel);
01744                 }
01745             }
01746         }
01747     }
01748 }
```



```

01747     }
01748 }
01749
01750     return filtered;
01751
01752 }

```

### 3.1.3.19 Image Image::filter\_freeman\_7 ( int num\_threads )

Applies the freeman mask  $(1, 1, 1)(1, -2, -1)(1, -1, -1)$ .

Definition at line 1757 of file [image.cpp](#).

```

01758 {
01759     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01760 );
01761     int m = 1;
01762
01763     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01764     {
01765         for(unsigned int z = 0; z < this->get_depth(); z++)
01766         {
01767             for(unsigned int x = m; x < this->get_width()-m; x++)
01768             {
01769                 for(unsigned int y = m; y < this->get_height()-m; y++)
01770                 {
01771                     int sum = (get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x-1, y, z, c)+
get_pixel_value(x-1, y+1, z, c)+get_pixel_value(x+1, y-1, z, c)+get_pixel_value(x, y-1, z, c))-1*(get_pixel_value(x, y+1,
z, c)+get_pixel_value(x+1, y, z, c)+get_pixel_value(x+1,y+1,z,c))+-2*get_pixel_value(x, y, z, c);
01772                     if (sum > 255 || sum < -255)
01773                     {
01774                         sum = 255;
01775                     }
01776                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01777                     filtered.set_pixel_value(x, y, z, c, pixel);
01778                 }
01779             }
01780         }
01781     }
01782 }
01783
01784     return filtered;
01785
01786 }
01787 }

```

### 3.1.3.20 Image Image::filter\_gaussian ( int o, int dim\_kernel, int num\_threads )

This function applies a gaussian kernel trough the hole image.

#### Parameters

<i>Receives</i>	the dimension of the kernel (dim_kernel) and a paremeter o wich stablish the values on the gaussian kernel.
-----------------	---

#### Returns

[Image](#) filtered which is the image with the gaussian filter applied.

Definition at line 841 of file [image.cpp](#).

```

00842 {
00843     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00844 );

```

```

00845     double kernel[dim_kernel*dim_kernel];
00846
00847     int m = (dim_kernel-1)/2;
00848
00849     double gaussian =1/pow(3.1415*2*o*o,0.5);
00850
00851     for(int i =-m; i <=m; i++)
00852     {
00853         for(int j =-m; j<=+m; j++)
00854         {
00855             double exp= -(i*i+j*j)*0.5/(o*o);
00856             kernel[(i+m)*dim_kernel + (j+m)]=gaussian*pow(2.7,exp);
00857         }
00858     }
00859
00860
00861
00862
00863     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00864     {
00865         for(unsigned int z = 0; z < this->get_depth(); z++)
00866         {
00867             for(unsigned int x = m; x < this->get_width(); x++)
00868             {
00869                 for(unsigned int y = m; y < this->get_height(); y++)
00870                 {
00871                     int cont=0;
00872                     unsigned char pixel=0;
00873
00874                     for(unsigned int i = x-m; i < x+m; i++)
00875                     {
00876                         for(unsigned int j = y-m; j< y+m; j++)
00877                         {
00878                             pixel+= this->get_pixel_value(i, j, z, c)*(kernel[cont]);
00879                             cont++;
00880                         }
00881                     }
00882                     filtered.set_pixel_value(x, y, z, c, (pixel/2));
00883                 }
00884             }
00885         }
00886     }
00887 }
00888
00889     return filtered;
00890 }

```

### 3.1.3.21 Image Image::filter\_Gradient\_horizontal ( int num\_threads )

This filter is used as as Sharpening Spatial Filter, used to identify borders and noise in the image. Can be used to identify horizontal borders or discrepation Applies the following filter:  $((1,2,1),(0,0,0),(-1,-2,-1))$

#### Returns

An image object that contains the original image after receiving a gradient filter in the horizontal direction. Could be used to identify horizontal borders.

Definition at line 436 of file [image.cpp](#).

```

00437 {
00438     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00439 );
00440
00441     int m = 1;
00442
00443     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00444     {
00445         for(unsigned int z = 0; z < this->get_depth(); z++)
00446         {
00447             for(unsigned int x = m; x < this->get_width()-m; x++)
00448             {
00449                 for(unsigned int y = m; y < this->get_height()-m; y++)

```

```

00449         {
00450             int sum = this->get_pixel_value(x-1, y-1, z, c) + 2*(this->get_pixel_value(x, y-1, z, c
)) + this->get_pixel_value(x+1, y-1, z, c) - (this->get_pixel_value(x-1, y+1, z, c) + 2*(this->
get_pixel_value(x, y+1, z, c)) + this->get_pixel_value(x+1, y+1, z, c));
00451             if (sum > 255 || sum < -255)
00452             {
00453                 sum = 255;
00454             }
00455             unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
00456             filtered.set_pixel_value(x, y, z, c, pixel);
00457         }
00458     }
00459 }
00460
00461     }
00462 }
00463     return filtered;
00464 }

```

### 3.1.3.22 Image Image::filter\_Gradient\_vertical ( int num\_threads )

This filter is used as as Sharpening Spatial Filter, used to identify borders and noise in the image. Can be used to identify vertical borders or discrepancies. Applies the following filter:  $((1,0,-1),(2,0,-2),(1,0,-1))$

#### Returns

An image object that contains the original image after receiving a gradient filter in the vertical direction. Could be used to identify vertical borders.

Definition at line 473 of file [image.cpp](#).

```

00474 {
00475     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
00476
00477     int m = 1;
00478
00479     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00480     {
00481         for(unsigned int z = 0; z < this->get_depth(); z++)
00482         {
00483             for(unsigned int x = m; x < this->get_width()-m; x++)
00484             {
00485                 for(unsigned int y = m; y < this->get_height()-m; y++)
00486                 {
00487                     int sum = get_pixel_value(x-1, y-1, z, c) + 2*get_pixel_value(x-1, y, z, c) +
get_pixel_value(x-1, y+1, z, c) - (get_pixel_value(x+1, y-1, z, c) + 2*get_pixel_value(x+1, y, z, c) +
get_pixel_value(x+1, y+1, z, c));
00488                     if (sum > 255 || sum < -255)
00489                     {
00490                         sum = 255;
00491                     }
00492                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
00493                     filtered.set_pixel_value(x, y, z, c, pixel);
00494                 }
00495             }
00496         }
00497     }
00498 }
00499     return filtered;
00500 }
00501 }

```

### 3.1.3.23 Image Image::filter\_horizontal\_borders ( int num\_threads )

Definition at line 711 of file [image.cpp](#).

```

00712 {
00713     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00714 );
00715     int m = 1;
00716     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00717     {
00718         for(unsigned int z = 0; z < this->get_depth(); z++)
00719         {
00720             for(unsigned int x = 0; x < this->get_width()-m; x++)
00721             {
00722                 for(unsigned int y = m; y < this->get_height()-m; y++)
00723                 {
00724                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(this->
00725 get_pixel_value(x, y-1, z, c) - get_pixel_value(x, y+1, z, c)));
00726                     filtered.set_pixel_value(x, y, z, c, pixel);
00727                 }
00728             }
00729         }
00730     }
00731 }
00732 }
00733 return filtered;
00734 }

```

### 3.1.3.24 Image Image::filter\_kirsch\_0 ( int num\_threads )

Applies the kirsch mask at 0°.  $(-3, -3, 5)(-3, 0, 5)(-3, -3, 5)$ .

Definition at line 1228 of file [image.cpp](#).

```

01229 {
01230     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01231 );
01232     int m = 1;
01233     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01234     {
01235         for(unsigned int z = 0; z < this->get_depth(); z++)
01236         {
01237             for(unsigned int x = m; x < this->get_width()-m; x++)
01238             {
01239                 for(unsigned int y = m; y < this->get_height()-m; y++)
01240                 {
01241                     int sum = -3*(get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x-1, y, z, c)+
01242 get_pixel_value(x-1, y+1, z, c)+get_pixel_value(x, y-1, z, c)+get_pixel_value(x, y+1, z, c))+5*(get_pixel_value(x+1, y-
01243 1, z, c)+get_pixel_value(x+1, y, z, c)+get_pixel_value(x+1, y+1, z, c));
01244                     if (sum > 255 || sum < -255)
01245                     {
01246                         sum = 255;
01247                     }
01248                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01249                     filtered.set_pixel_value(x, y, z, c, pixel);
01250                 }
01251             }
01252         }
01253     }
01254 }
01255 return filtered;
01256 }
01257 }
01258 }

```

### 3.1.3.25 Image Image::filter\_kirsch\_135 ( int num\_threads )

Applies the kirsch mask at 135°.  $(5, 5, -3)(5, 0, -3)(-3, -3, -3)$ .

Definition at line 1335 of file [image.cpp](#).

```

01336 {
01337     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01338 );
01339     int m = 1;
01340     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01341     {
01342         for(unsigned int z = 0; z < this->get_depth(); z++)
01343         {
01344             for(unsigned int x = m; x < this->get_width()-m; x++)
01345             {
01346                 for(unsigned int y = m; y < this->get_height()-m; y++)
01347                 {
01348                     int sum = -3*(get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x+1, y, z, c)+
get_pixel_value(x+1, y+1, z, c)+get_pixel_value(x, y+1, z, c)+get_pixel_value(x-1, y+1, z, c))+5*(get_pixel_value(x-1,
y, z, c)+get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x, y-1, z, c));
01349                     if (sum > 255 || sum < -255)
01350                     {
01351                         sum = 255;
01352                     }
01353                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01354                     filtered.set_pixel_value(x, y, z, c, pixel);
01355                 }
01356             }
01357         }
01358     }
01359 }
01360 }
01361
01362 return filtered;
01363
01364 }

```

### 3.1.3.26 Image Image::filter\_kirsch\_180( int num\_threads )

Applies the kirsch mask at 180°. (5,-3,-3)(5,0,-3)(5,-3,-3).

Definition at line 1370 of file [image.cpp](#).

```

01371 {
01372     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01373 );
01374     int m = 1;
01375     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01376     {
01377         for(unsigned int z = 0; z < this->get_depth(); z++)
01378         {
01379             for(unsigned int x = m; x < this->get_width()-m; x++)
01380             {
01381                 for(unsigned int y = m; y < this->get_height()-m; y++)
01382                 {
01383                     int sum = -3*(get_pixel_value(x, y-1, z, c)+get_pixel_value(x+1, y-1, z, c)+
get_pixel_value(x+1, y, z, c)+get_pixel_value(x+1, y+1, z, c)+get_pixel_value(x, y+1, z, c))+5*(get_pixel_value(x-1, y,
z, c)+get_pixel_value(x-1, y+1, z, c)+get_pixel_value(x-1, y-1, z, c));
01384                     if (sum > 255 || sum < -255)
01385                     {
01386                         sum = 255;
01387                     }
01388                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01389                     filtered.set_pixel_value(x, y, z, c, pixel);
01390                 }
01391             }
01392         }
01393     }
01394 }
01395 }
01396
01397 return filtered;
01398
01399 }

```

### 3.1.3.27 Image Image::filter\_kirsch\_225 ( int num\_threads )

Applies the kirsch mask at 225°.  $(-3, -3, -3)(5, 0, -3)(5, 5, -3)$ .

Definition at line 1405 of file [image.cpp](#).

```

01406 {
01407     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01408 );
01409     int m = 1;
01410     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01411     {
01412         for(unsigned int z = 0; z < this->get_depth(); z++)
01413         {
01414             for(unsigned int x = m; x < this->get_width()-m; x++)
01415             {
01416                 for(unsigned int y = m; y < this->get_height()-m; y++)
01417                 {
01418                     int sum = -3*(get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x, y-1, z, c)+
get_pixel_value(x+1, y-1, z, c)+get_pixel_value(x+1, y, z, c)+get_pixel_value(x+1, y+1, z, c))+5*(get_pixel_value(x-1,
y, z, c)+get_pixel_value(x-1, y+1, z, c)+get_pixel_value(x, y+1, z, c));
01419                     if (sum > 255 || sum < -255)
01420                     {
01421                         sum = 255;
01422                     }
01423                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01424                     filtered.set_pixel_value(x, y, z, c, pixel);
01425                 }
01426             }
01427         }
01428     }
01429 }
01430 }
01431
01432 return filtered;
01433
01434 }
```

### 3.1.3.28 Image Image::filter\_kirsch\_270 ( int num\_threads )

Applies the kirsch mask at 270°.  $(-3, -3, -3)(-3, 0, -3)(5, 5, 5)$ .

Definition at line 1440 of file [image.cpp](#).

```

01441 {
01442     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01443 );
01444     int m = 1;
01445     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01446     {
01447         for(unsigned int z = 0; z < this->get_depth(); z++)
01448         {
01449             for(unsigned int x = m; x < this->get_width()-m; x++)
01450             {
01451                 for(unsigned int y = m; y < this->get_height()-m; y++)
01452                 {
01453                     int sum = -3*(get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x-1, y, z, c)+
get_pixel_value(x, y-1, z, c)+get_pixel_value(x+1, y-1, z, c)+get_pixel_value(x+1, y, z, c))+5*(get_pixel_value(x, y+1,
z, c)+get_pixel_value(x+1, y+1, z, c)+get_pixel_value(x-1, y+1, z, c));
01454                     if (sum > 255 || sum < -255)
01455                     {
01456                         sum = 255;
01457                     }
01458                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01459                     filtered.set_pixel_value(x, y, z, c, pixel);
01460                 }
01461             }
01462         }
01463     }
01464 }
01465 }
```

```

01466
01467     return filtered;
01468
01469 }

```

### 3.1.3.29 Image Image::filter\_kirsch\_315 ( int num\_threads )

Applies the kirsch mask at 315°.  $(-3, -3, -3)(-3, 0, 5)(-3, 5, 5)$ .

Definition at line 1475 of file [image.cpp](#).

```

01476 {
01477     Image filtered (this->get_width() , this->get_height() , this->get_depth() , this->get_spectrum(), 0
01478 );
01479     int m = 1;
01480     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01481     {
01482         for(unsigned int z = 0; z < this->get_depth(); z++)
01483         {
01484             for(unsigned int x = m; x < this->get_width()-m; x++)
01485             {
01486                 for(unsigned int y = m; y < this->get_height()-m; y++)
01487                 {
01488                     int sum = -3*(get_pixel_value(x-1, y+1, z, c)+get_pixel_value(x-1, y, z, c)+
get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x, y-1, z, c)+get_pixel_value(x+1, y-1, z, c))+5*(get_pixel_value(x, y+
1, z, c)+get_pixel_value(x+1, y+1, z, c)+get_pixel_value(x+1, y, z, c));
01489                     if (sum > 255 || sum < -255)
01490                     {
01491                         sum = 255;
01492                     }
01493                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01494                     filtered.set_pixel_value(x, y, z, c, pixel);
01495                 }
01496             }
01497         }
01498     }
01499 }
01500 }
01501
01502     return filtered;
01503
01504 }

```

### 3.1.3.30 Image Image::filter\_kirsch\_45 ( int num\_threads )

Applies the kirsch mask at 45°.  $(-3, 5, 5)(-3, 0, 5)(-3, -3, -3)$ .

Definition at line 1264 of file [image.cpp](#).

```

01265 {
01266     Image filtered (this->get_width() , this->get_height() , this->get_depth() , this->get_spectrum(), 0
01267 );
01268     int m = 1;
01269     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01270     {
01271         for(unsigned int z = 0; z < this->get_depth(); z++)
01272         {
01273             for(unsigned int x = m; x < this->get_width()-m; x++)
01274             {
01275                 for(unsigned int y = m; y < this->get_height()-m; y++)
01276                 {
01277                     int sum = -3*(get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x-1, y, z, c)+
get_pixel_value(x-1, y+1, z, c)+get_pixel_value(x+1, y+1, z, c)+get_pixel_value(x, y+1, z, c))+5*(get_pixel_value(x+1,
y-1, z, c)+get_pixel_value(x+1, y, z, c)+get_pixel_value(x, y-1, z, c));
01278                     if (sum > 255 || sum < -255)
01279                     {
01280                         sum = 255;
01281                     }

```

```

01282             unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01283             filtered.set_pixel_value(x, y, z, c, pixel);
01284         }
01285     }
01286 }
01287
01288     }
01289 }
01290
01291     return filtered;
01292
01293
01294 }
```

### 3.1.3.31 Image Image::filter\_kirsch\_90 ( int num\_threads )

Applies the kirsch mask at 90°.  $(5,5,5)(-3,0-3)(-3,-3,-3)$ .

Definition at line 1300 of file [image.cpp](#).

```

01301 {
01302     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01303 );
01304     int m = 1;
01305     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01306     {
01307         for(unsigned int z = 0; z < this->get_depth(); z++)
01308         {
01309             for(unsigned int x = m; x < this->get_width()-m; x++)
01310             {
01311                 for(unsigned int y = m; y < this->get_height()-m; y++)
01312                 {
01313                     int sum = -3*(get_pixel_value(x-1, y, z, c)+get_pixel_value(x-1, y+1, z, c)+
01314 get_pixel_value(x, y+1, z, c)+get_pixel_value(x+1, y+1, z, c)+get_pixel_value(x+1, y, z, c))+5*(get_pixel_value(x+1, y-
01315 1, z, c)+get_pixel_value(x, y-1, z, c)+get_pixel_value(x-1, y-1, z, c));
01316                     if (sum > 255 || sum < -255)
01317                     {
01318                         sum = 255;
01319                     }
01320                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01321                     filtered.set_pixel_value(x, y, z, c, pixel);
01322                 }
01323             }
01324         }
01325     }
01326     return filtered;
01327
01328
01329 }
```

### 3.1.3.32 Image Image::filter\_Laplacian ( int num\_threads )

Returns an image after applying the Laplacian filter to the image. Considers the diagonal values This function applies a convolution with this kernel:  $((1,1,1),(1,-8,1),(1,1,1))$ .

#### Returns

A Filtered image with the Laplacian filter applied.

Definition at line 347 of file [image.cpp](#).

```

00348 {
00349     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00350 );
00351 }
```



```

00350
00351     int m = 1;
00352
00353     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00354     {
00355         for(unsigned int z = 0; z < this->get_depth(); z++)
00356         {
00357             for(unsigned int x = m; x < this->get_width()-m; x++)
00358             {
00359                 for(unsigned int y = m; y < this->get_height()-m; y++)
00360                 {
00361                     int sum = 0;
00362
00363                     for (unsigned int i = 0 ; i < 3; i++)
00364                     {
00365                         for(unsigned int j = 0 ; j < 3; j++)
00366                         {
00367                             sum += -this->get_pixel_value(x+i-1, y+i-1, z, c);
00368                         }
00369                     }
00370
00371                     sum += 9*(this->get_pixel_value(x,y,z,c));
00372
00373                     if (sum > 255 || sum < -255)
00374                     {
00375                         sum = 255;
00376                     }
00377                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
00378                     filtered.set_pixel_value(x, y, z, c, pixel);
00379                 }
00380             }
00381         }
00382     }
00383 }
00384 }
00385 return filtered;
00386 }

```

### 3.1.3.33 Image Image::filter\_Laplacian\_no\_diagonal ( int num\_threads )

Works as a derivative function, reacts to high change on the pixels value, especially to noise, and borders. Applies the following filter:  $((0, -1, 0), (-1, 4, -1), (0, -1, 0))$

#### Returns

A filtered [Image](#) with the laplacian filter.

Definition at line 393 of file [image.cpp](#).

```

00394 {
00395     //int kernel[9] = {0, -1, 0, -1, 4, -1, 0, -1, 0};
00396
00397     //return (this->filter(kernel, 3, 1));
00398
00399     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00400 );
00401
00402     int m = 1;
00403
00404     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00405     {
00406         for(unsigned int z = 0; z < this->get_depth(); z++)
00407         {
00408             for(unsigned int x = m; x < this->get_width()-m; x++)
00409             {
00410                 for(unsigned int y = m; y < this->get_height()-m; y++)
00411                 {
00412                     int sum = 4*(this->get_pixel_value(x,y,z,c)) - (this->get_pixel_value(x-1,y,z,c) +
00413 this->get_pixel_value(x+1,y,z,c) + this->get_pixel_value(x,y-1,z,c) +this->get_pixel_value(x,y+1,z,c));
00414
00415                     if (sum > 255 || sum < -255)
00416                     {
00417                         sum = 255;
00418                     }
00419                 }
00420             }
00421         }
00422     }
00423 }
00424 }
00425 return filtered;
00426 }

```

```

00416         }
00417         unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
00418         filtered.set_pixel_value(x, y, z, c, pixel);
00419     }
00420 }
00421 }
00422 }
00423 }
00424 }
00425 return filtered;
00426 }
00427 }

```

### 3.1.3.34 Image Image::filter\_maximum ( int num\_threads )

Assigns the highest value in the neighborhood. Assigns the highest value in the neighborhood around the desired pixel.

Definition at line 1795 of file [image.cpp](#).

```

01796 {
01797     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01798 );
01799     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01800     {
01801         for(unsigned int z = 0; z < this->get_depth(); z++)
01802         {
01803             for(unsigned int x = 1; x < this->get_width()-1; x++)
01804             {
01805                 for(unsigned int y = 1; y < this->get_height()-1; y++)
01806                 {
01807                     unsigned char max = 0;
01808
01809                     for (unsigned int i = x-1; i< x+2; i++)
01810                     {
01811                         for (unsigned int j = y-1; j< y+2; j++)
01812                         {
01813                             unsigned char pixel = (this->get_pixel_value(i, j, z, c));
01814
01815                             if (pixel > max)
01816                             {
01817                                 max = this->get_pixel_value(i, j, z, c);
01818                             }
01819                         }
01820                     }
01821
01822                     filtered.set_pixel_value(x, y, z, c, max);
01823                 }
01824             }
01825         }
01826     }
01827 }
01828 }
01829 return filtered;
01830 }

```

### 3.1.3.35 Image Image::filter\_median ( int dim, int num\_threads )

This function calculates the median of the range of pixels into the kernel and sets this value in the central pixel of the kernel.

#### Parameters

<i>Only</i>	receives the dimension of the kernel (dim), wich only can be an impair number.
-------------	--

**Returns**

[Image](#) filtered which is the image with the median filter applied.

Definition at line 747 of file [image.cpp](#).

```

00748 {
00749     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00750 );
00751     //int kernel [dim*dim];
00752
00753     int m = (dim-1)/2;
00754     unsigned char pixel_values [dim*dim];
00755     unsigned char temp;
00756
00757     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00758     {
00759         for(unsigned int z = 0; z < this->get_depth(); z++)
00760         {
00761             for(unsigned int x = m; x < this->get_width(); x++)
00762             {
00763                 for(unsigned int y = m; y < this->get_height(); y++)
00764                 {
00765                     for(unsigned int i = x-m; i < x+m; i++)
00766                     {
00767                         for(unsigned int j = y-m; j < y+m; j++)
00768                         {
00769                             pixel_values [(i-x+m)*dim + (j-y+m)] = this->get_pixel_value(i, j, z, c);
00770                         }
00771                     }
00772                 }
00773                 for(int k=0; k<dim*dim ; k++)
00774                 {
00775                     for(int p=k+1 ; p<dim*dim ; p++)
00776                     {
00777                         if(pixel_values[p] < pixel_values[k])
00778                         {
00779                             // Intercambiar los valores
00780                             temp = pixel_values[k];
00781                             pixel_values[k] = pixel_values[p];
00782                             pixel_values[p] = temp;
00783                         }
00784                     }
00785                 }
00786                 unsigned char pixel = pixel_values[((dim*dim-1)/2)-1];
00787                 filtered.set_pixel_value(x, y, z, c, pixel);
00788             }
00789         }
00790     }
00791 }
00792 }
00793 }
00794     return filtered;
00795 }
```

**3.1.3.36 Image Image::filter\_minimum ( int num\_threads )**

Assigns the highest value in the neighborhood.

Definition at line 1838 of file [image.cpp](#).

```

01839 {
01840     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01841 );
01842     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01843     {
01844         for(unsigned int z = 0; z < this->get_depth(); z++)
01845         {
01846             for(unsigned int x = 1; x < this->get_width()-1; x++)
01847             {
01848                 for(unsigned int y = 1; y < this->get_height()-1; y++)
01849                 {
```

```

01850         unsigned char minimum = 255;
01851
01852         for (unsigned int i = x-1; i < x+2; i++)
01853         {
01854             for (unsigned int j = y-1; j < y+2; j++)
01855             {
01856                 if ((this->get_pixel_value(i, j, z, c)) < minimum)
01857                 {
01858                     minimum = this->get_pixel_value(i, j, z, c);
01859                 }
01860             }
01861         }
01862         filtered.set_pixel_value(x, y, z, c, minimum);
01863     }
01864 }
01865 }
01866 }
01867 }
01868 }
01869 }
01870     return filtered;
01871 }

```

### 3.1.3.37 Image Image::filter\_modal ( int dim, int num\_threads )

This function calculates the modal of the range of pixels into the kernel and sets this value in the central pixel of the kernel.

#### Parameters

<i>Only</i>	receives the dimension of the kernel (dim), wich only can be an impair number.
-------------	--

#### Returns

[Image](#) filtered which is the image with the modal filter applied.

Definition at line 898 of file [image.cpp](#).

```

00899 {
00900     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00901 );
00902     unsigned char pixel_values[dim*dim];
00903     unsigned char moda;
00904     unsigned char average=0;
00905     int m=(dim-1)/2;
00906     unsigned char copy_pixels[dim*dim];
00907
00908     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00909     {
00910         for(unsigned int z = 0; z < this->get_depth(); z++)
00911         {
00912             for(unsigned int x = m; x < this->get_width(); x++)
00913             {
00914                 for(unsigned int y = m; y < this->get_height(); y++)
00915                 {
00916                     for(unsigned int i = x-m; i < x+m; i++)
00917                     {
00918                         for(unsigned int j = y-m; j < y+m; j++)
00919                         {
00920                             pixel_values [(i-x+m)*dim + (j-y+m)] = this->get_pixel_value(i, j, z, c);
00921
00922                             int frequency[dim*dim];
00923                             moda=0;
00924
00925                             for(int k=0;k<dim*dim;k++)
00926                             {
00927                                 copy_pixels[k] = pixel_values[k];
00928                                 frequency[k]=0;
00929                             }
00930
00931                             for(int p=0;p<dim*dim;p++)
00932                             {

```

```

00932         for(int q=p+1;q<dim*dim;q++)
00933         {
00934             if(copy_pixels[p]==pixel_values[q]){
00935                 frequency[p]++;
00936             }
00937         }
00938     }
00939 }
00940
00941 }
00942
00943
00944
00945     for(int s=0; s<dim*dim ; s++)
00946     {
00947         for(int e=s+1 ; e<dim*dim ; e++)
00948         {
00949             if(frequency[e] < frequency[s])
00950             {
00951                 moda = copy_pixels[s];
00952                 average=copy_pixels[s];
00953             }
00954         }
00955     }
00956
00957
00958     if(mod==0)
00959     {
00960         for(int k=0;k<dim*dim;k++)
00961         {
00962             moda += pixel_values[k];
00963         }
00964         average=(moda/dim);
00965     }
00966
00967 }
00968
00969 }
00970
00971     filtered.set_pixel_value(x, y, z, c, average);
00972 }
00973
00974 }
00975
00976 }
00977 }
00978
00979 return filtered;
00980
00981 }

```

### 3.1.3.38 Image Image::filter\_order\_statistics ( int dim, int order, int num\_threads )

Definition at line 1873 of file [image.cpp](#).

```

01874 {
01875     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01876 );
01877     //int kernel [dim*dim];
01878
01879     int m = (dim-1)/2;
01880
01881     unsigned char pixel_values [dim*dim];
01882     unsigned char temp;
01883
01884     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01885     {
01886         for(unsigned int z = 0; z < this->get_depth(); z++)
01887         {
01888             for(unsigned int x = m; x < this->get_width(); x++)
01889             {
01890                 for(unsigned int y = m; y < this->get_height(); y++)
01891                 {
01892                     for(unsigned int i = x-m; i < x+m+1; i++)
01893                     {

```

```

01894         for(unsigned int j = y-m; j< y+m+1; j++)
01895         {
01896             pixel_values [(i-x+m)*dim + (j-y+m)]= this->get_pixel_value(i, j, z, c);
01897         }
01898     }
01899 }
01900 for(int k=0; k<dim*dim ; k++)
01901 {
01902     for(int p=k+1 ; p<dim*dim ; p++)
01903     {
01904         if(pixel_values[p] < pixel_values[k])
01905         {
01906             // Intercambiar los valores
01907             temp = pixel_values[k];
01908             pixel_values[k] = pixel_values[p];
01909             pixel_values[p] = temp;
01910         }
01911     }
01912 }
01913 unsigned char pixel = pixel_values[order];
01914 filtered.set_pixel_value(x, y, z, c, pixel);
01915 }
01916 }
01917 }
01918 }
01919 }
01920 }
01921 return filtered;
01922 }

```

### 3.1.3.39 Image Image::filter\_Prewitt\_E\_W ( int num\_threads )

Definition at line 590 of file [image.cpp](#).

```

00591 {
00592     //int kernel[9] = {1, 0, -1, 1, 0, -1, 1, 0, -1};
00593
00594     //return (this->filter(kernel, 3, 1));
00595
00596     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00597 );
00598     int m = 1;
00599
00600     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00601     {
00602         for(unsigned int z = 0; z < this->get_depth(); z++)
00603         {
00604             for(unsigned int x = m; x < this->get_width()-m; x++)
00605             {
00606                 for(unsigned int y = m; y < this->get_height()-m; y++)
00607                 {
00608                     int sum = get_pixel_value(x-1, y-1, z, c) + get_pixel_value(x-1, y, z, c) +
get_pixel_value(x-1, y+1, z, c) - (get_pixel_value(x+1, y-1, z, c) + get_pixel_value(x+1, y, z, c) + get_pixel_value(x+
1, y+1, z, c));
00609                     if (sum > 255 || sum < -255)
00610                     {
00611                         sum = 255;
00612                     }
00613                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
00614                     filtered.set_pixel_value(x, y, z, c, pixel);
00615                 }
00616             }
00617         }
00618     }
00619 }
00620 }
00621 }
00622 return filtered;
00623 }
00624 }

```

## 3.1.3.40 Image Image::filter\_Prewitt\_N\_S ( int num\_threads )

Definition at line 520 of file [image.cpp](#).

```

00521 {
00522     //int kernel[9] = {1, 1, 1, 0, 0, 0, -1, -1, -1};
00523
00524     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00525 );
00526     int m = 1;
00527
00528     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00529     {
00530         for(unsigned int z = 0; z < this->get_depth(); z++)
00531         {
00532             for(unsigned int x = m; x < this->get_width()-m; x++)
00533             {
00534                 for(unsigned int y = m; y < this->get_height()-m; y++)
00535                 {
00536                     int sum = get_pixel_value(x-1, y-1, z, c) + get_pixel_value(x, y-1, z, c) +
get_pixel_value(x+1, y-1, z, c) - (get_pixel_value(x-1, y+1, z, c) + get_pixel_value(x, y+1, z, c) + get_pixel_value(x+
1, y+1, z, c));
00537                     if (sum > 255 || sum < -255)
00538                     {
00539                         sum = 255;
00540                     }
00541                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
00542                     filtered.set_pixel_value(x, y, z, c, pixel);
00543                 }
00544             }
00545         }
00546     }
00547 }
00548 }
00549
00550 return filtered;
00551
00552 }
```

## 3.1.3.41 Image Image::filter\_Prewitt\_NE\_SW ( int num\_threads )

Definition at line 554 of file [image.cpp](#).

```

00555 {
00556     //int kernel[9] = {0, 1, 1, -1, 0, 1, -1, -1, 0};
00557
00558     //return (this->filter(kernel, 3, 1));
00559
00560     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00561 );
00562     int m = 1;
00563
00564     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00565     {
00566         for(unsigned int z = 0; z < this->get_depth(); z++)
00567         {
00568             for(unsigned int x = m; x < this->get_width()-m; x++)
00569             {
00570                 for(unsigned int y = m; y < this->get_height()-m; y++)
00571                 {
00572                     int sum = get_pixel_value(x, y-1, z, c) + get_pixel_value(x+1, y-1, z, c) +
get_pixel_value(x+1, y, z, c) - (get_pixel_value(x-1, y, z, c) + get_pixel_value(x-1, y+1, z, c) + get_pixel_value(x, y
+1, z, c));
00573                     if (sum > 255 || sum < -255)
00574                     {
00575                         sum = 255;
00576                     }
00577                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
00578                     filtered.set_pixel_value(x, y, z, c, pixel);
00579                 }
00580             }
00581         }
00582     }
00583 }
```

```

00582
00583     }
00584 }
00585
00586     return filtered;
00587
00588 }
```

### 3.1.3.42 Image Image::filter\_Prewitt\_NW\_SE ( int num\_threads )

Definition at line 626 of file [image.cpp](#).

```

00627 {
00628     //int kernel[9] = {-1, -1, 0, -1, 0, 1, 0, 1, 1};
00629
00630     // (this->filter(kernel, 3, 1));
00631
00632     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00633 );
00634     int m = 1;
00635
00636     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00637     {
00638         for(unsigned int z = 0; z < this->get_depth(); z++)
00639         {
00640             for(unsigned int x = m; x < this->get_width()-m; x++)
00641             {
00642                 for(unsigned int y = m; y < this->get_height()-m; y++)
00643                 {
00644                     int sum = get_pixel_value(x-1, y-1, z, c) + get_pixel_value(x-1, y, z, c) +
get_pixel_value(x, y-1, z, c) - (get_pixel_value(x+1, y, z, c) + get_pixel_value(x+1, y+1, z, c) + get_pixel_value(x, y
+1, z, c));
00645                     if (sum > 255 || sum < -255)
00646                     {
00647                         sum = 255;
00648                     }
00649                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
00650                     filtered.set_pixel_value(x, y, z, c, pixel);
00651                 }
00652             }
00653         }
00654     }
00655 }
00656 }
00657
00658     return filtered;
00659
00660 }
```

### 3.1.3.43 Image Image::filter\_vertical\_borders ( int num\_threads )

Definition at line 686 of file [image.cpp](#).

```

00687 {
00688     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00689 );
00690     int m = 1;
00691
00692     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00693     {
00694         for(unsigned int z = 0; z < this->get_depth(); z++)
00695         {
00696             for(unsigned int x = m; x < this->get_width()-m; x++)
00697             {
00698                 for(unsigned int y = 0; y < this->get_height()-m; y++)
00699                 {
00700                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(this->
get_pixel_value(x-1, y, z, c) - get_pixel_value(x+1, y, z, c)));
```



```

00701             filtered.set_pixel_value(x, y, z, c, pixel);
00702         }
00703     }
00704 }
00705
00706     }
00707 }
00708     return filtered;
00709 }

```

#### 3.1.3.44 void Image::gaussian\_noise ( double *variance*, int *num\_threads* )

Converts an RGB image to gray scale.

This function applies the gaussian noise to an image. The gaussian noise increases or decreases intensity to a pixel, depending of the variance.

##### Parameters

<i>variance</i>	this parameter is used to set the value of noise that is applied to the image.
-----------------	--

Definition at line 1970 of file [image.cpp](#).

```

01971 {
01972     srand(1);
01973     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01974     {
01975         for(unsigned int z = 0; z < this->get_depth(); z++)
01976         {
01977             for(unsigned int x = 0; x < this->get_width(); x++)
01978             {
01979                 for(unsigned int y = 0; y < this->get_height(); y++)
01980                 {
01981                     double random= variance*(rand()-RAND_MAX/variance)/RAND_MAX;
01982                     unsigned char pixel= this->get_pixel_value(x,y,z,c) + random;
01983
01984                     if((pixel<255) & (pixel>0))
01985                     {
01986                         (*(this->Img))(x, y, z, c)= pixel;
01987                     }
01988                 }
01989             }
01990         }
01991     }
01992 }
01993
01994 }
01995 }
01996
01997 }

```

#### 3.1.3.45 int \*Image::get\_histogram ( unsigned int *c*, unsigned int *z* )

This function returns an array containing the values of the histogram points, in the desired channel and depth. An Histogram is measure of the frequency of a intensity value in an image, and is often used as a parameter to improve the contrast and quality of the image. After observing the histogram ( see [plot\\_histogram\(\)](#) ) you could equalize the image.

Definition at line 1167 of file [image.cpp](#).

```

01168 {
01169     int histogram [256];
01170     for(int i = 0; i<256; i++)
01171     {
01172         histogram[i] = 0;
01173     }
01174 }

```

```

01175     if (c < this->get_spectrum() && z < this->get_depth())
01176     {
01177         for(unsigned int x = 0; x < this->get_width(); x++)
01178         {
01179             for(unsigned int y = 0; y < this->get_height(); y++)
01180             {
01181                 unsigned char pixel_value = this->get_pixel_value(x,y,z,c);
01182                 (histogram[pixel_value])++;
01183             }
01184         }
01185     }
01186
01187     int* histogram_pointer = histogram;
01188
01189     return histogram_pointer;
01190 }

```

### 3.1.3.46 Image Image::gray\_scale ( int num\_threads )

Assigns the lowest value in the neighborhood.

This function converts an RGB image to one in gray scale. The library uses this conversion:  $f(x,y) = 0.11R + 0.56G + 0.14B$  Where  $f$  is the intensity of the pixel on the gray scale and  $R, G$  and  $B$  the pixel values on the different channels.

#### Returns

This function returns the monochromatic image.

Definition at line 2136 of file [image.cpp](#).

```

02137 {
02138     Image gray_image (this->get_width() , this->get_height() , this->get_depth() , 1, 0);
02139
02140
02141     for(unsigned int z = 0; z < this->get_depth(); z++)
02142     {
02143         for(unsigned int x = 0; x < this->get_width(); x++)
02144         {
02145             for(unsigned int y = 0; y < this->get_height(); y++)
02146             {
02147
02148                 unsigned char pixel_intensity = 0.56*this->get_pixel_value(x,y,z,1)+0.14*this->
02149 get_pixel_value(x,y,z,0)+0.11*this->get_pixel_value(x,y,z,2);
02150                 gray_image.set_pixel_value(x, y, z, 0, pixel_intensity);
02151             }
02152         }
02153     }
02154
02155     return gray_image;
02156 }

```

### 3.1.3.47 int\* Image::histogram\_equalization ( int \*, const char \* title )

### 3.1.3.48 Image Image::interpolation ( int num\_threads )

This function doubles the size of the image and use the closer neighborhood interpolation.

#### Returns

This function returns the image interpolated.

Definition at line 2005 of file [image.cpp](#).

```

02006 {
02007     int i,j=0;
02008     Image result (2*this->get_width(),2*this->get_height(),this->get_depth(),this->get_spectrum(),0);
02009     for(unsigned int c = 0; c < this->get_spectrum(); c++)
02010     {
02011         for(unsigned int z = 0; z < this->get_depth(); z++)
02012         {
02013             for(unsigned int y = 0; y < this->get_width(); y++)
02014             {
02015                 for(unsigned int x = 0; x < this->get_height(); x++)
02016                 {
02017                     unsigned char pixel=this->get_pixel_value(x,y,z,c);
02018
02019                     result.set_pixel_value(x+i,y+j,z,c,pixel);
02020                     result.set_pixel_value(x+1+i,y+j,z,c,pixel);
02021                     result.set_pixel_value(x+i,y+1+j,z,c,pixel);
02022                     result.set_pixel_value(x+1+i,y+1+j,z,c,pixel);
02023                     i++;
02024                 }
02025                 i=0;
02026                 j++;
02027             }
02028             j=0;
02029         }
02030     }
02031     return result;
02032 }

```

#### 3.1.3.49 Image Image::inverse ( int num\_threads )

Executes this transformation:  $v(x,y,z,c) = 255 - u(x,y,z,c)$

##### Returns

An image object that contains the inverse of the original image, this means that every pixel value is subtracted to 255.

Definition at line 994 of file [image.cpp](#).

```

00995 {
00996     Image inverted (this->get_width() , this->get_height() , this->get_depth() , this->get_spectrum() , 0
00997 );
00998     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00999     {
01000         for(unsigned int z = 0; z < this->get_depth(); z++)
01001         {
01002             for(unsigned int x = 0; x < this->get_width(); x++)
01003             {
01004                 for(unsigned int y = 0; y < this->get_height(); y++)
01005                 {
01006                     unsigned char pixel= static_cast<unsigned int>(255-this->get_pixel_value(x,y,z,c));
01007                     inverted.set_pixel_value(x,y,z,c,pixel);
01008                 }
01009             }
01010         }
01011     }
01012     return inverted;
01013 }

```

#### 3.1.3.50 Image Image::log\_transformation ( int num\_threads )

Executes this transformation:  $v(x,y,z,c) = c \log(u(x,y,z,c) + 1)$  where  $v(x,y,z,c)$  is the transformed pixel, and  $u(x,y,z,c)$  is the original pixel.

**Returns**

An image object that contains the inverse of the original image, this means that every pixel value is subtracted to 255.

Definition at line 1020 of file [image.cpp](#).

```

01021 {
01022     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01023 );
01024     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01025     {
01026         for(unsigned int z = 0; z < this->get_depth(); z++)
01027         {
01028             for(unsigned int x = 0; x < this->get_width(); x++)
01029             {
01030                 for(unsigned int y = 0; y < this->get_height(); y++)
01031                 {
01032                     unsigned char pixel = static_cast<unsigned char>((255/log(256)) * log(1+this->
get_pixel_value(x, y, z, c)));
01033
01034                     filtered.set_pixel_value(x,y,z,c, pixel);
01035                 }
01036             }
01037         }
01038     }
01039     return filtered;
01040 }
```

**3.1.3.51 Image Image::median\_omp ( int , int )****3.1.3.52 Image Image::multiply\_img ( double *multiplier*, int *num\_threads* )**

This function multiplies the pixel values by a factor. If the pixel value is higher than 255, adjust the pixel value to 255.

**Parameters**

<i>double</i>	<i>multiplier</i> is the factor that mutiplies all the pixel values.
---------------	--

**Returns**

[Image](#) result: Is the result of multiply the image.

Definition at line 273 of file [image.cpp](#).

```

00274 {
00275     Image result (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0);
00276     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00277     {
00278         for(unsigned int z = 0; z < this->get_depth(); z++)
00279         {
00280             for(unsigned int x = 0; x < this->get_width(); x++)
00281             {
00282                 for(unsigned int y = 0; y < this->get_height(); y++)
00283                 {
00284                     unsigned char pixel= static_cast<unsigned int>(abs(this->get_pixel_value(x,y,z,c) *
multiplier));
00285
00286                     if (pixel >255)
00287                         pixel = 255;
00288                     result.set_pixel_value(x,y,z,c,pixel);
00289                 }
00290             }
00291         }
00292     }
00293     return result;
00294 }
```

3.1.3.53 void Image::plot\_histogram ( int *levels*, const char \* *title* )

This function plot the histogram, using the CImg histogram function.

**Parameters**

<i>levels</i>	is the number of bars or columns that appear in the histogram.
<i>title</i>	is the title of the histogram.

Definition at line 1197 of file [image.cpp](#).

```

01198 {
01199     CImg<unsigned char> img = this->Img->histogram(levels);
01200
01201     CImgDisplay main_display (*(this->Img), title);
01202
01203     img.display_graph(main_display, 3, 1, "Pixel Intensity", 0, 0, "Frequency", 0, 0);
01204 }
```

**3.1.3.54 void Image::plot\_histogram\_equalization ( int *levels*, const char \* *title* )**

Definition at line 1211 of file [image.cpp](#).

```

01212 {
01213     CImg<unsigned char> img = this->Img->equalize(levels);
01214
01215     CImgDisplay main_display (*(this->Img), title);
01216
01217     img.display_graph(main_display, 3, 1, "Pixel Intensity", 0, 0, "Frequency", 0, 0);
01218 }
```

**3.1.3.55 Image Image::power\_law\_transformation ( double *exponent*, int *num\_threads* )**

Executes this transformation:  $v(x,y,z,c) = c \log(u(x,y,z,c) + 1)$ .

Applies a transformation given by the equation  $v(x,y) = cu(x,y)^\gamma$  where  $u(x,y)$  is the value of the non filtered image, and  $v(xy)$  is the intensity value in the filtered image.  $\gamma, c$  are constants. In this case  $\gamma$  is a parameter.

**Returns**

A filtered image with the power law transformation

Definition at line 1091 of file [image.cpp](#).

```

01092 {
01093     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01094 );
01095     double k = (pow(255, 1-exponent));
01096
01097     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01098     {
01099         for(unsigned int z = 0; z < this->get_depth(); z++)
01100         {
01101             for(unsigned int x = 0; x < this->get_width(); x++)
01102             {
01103                 for(unsigned int y = 0; y < this->get_height(); y++)
01104                 {
01105                     double power_law = k * pow( (this->get_pixel_value(x,y,z,c)) , exponent);
01106                     unsigned char pixel = static_cast<unsigned char>(power_law);
01107                     filtered.set_pixel_value(x, y, z, c, pixel);
01108                 }
01109             }
01110         }
01111     }
01112 }
01113 }
01114 return filtered;
01115 }
```

**3.1.3.56** Image Image::rgb\_hsv ( )**3.1.3.57** void Image::salt\_pepper ( double *intensity*, int *num\_threads* )

Put pepper (black pixels) and salt(white pixels)

**Parameters**

<i>intensity</i>	is used to compute the percentage of salt and pepper that is applied to the image.
------------------	--

Definition at line 1933 of file [image.cpp](#).

```

01934 {
01935     srand(1);
01936     double percentage = 1-(intensity/100);
01937     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01938     {
01939         for(unsigned int z = 0; z < this->get_depth(); z++)
01940         {
01941             for(unsigned int x = 0; x < this->get_width(); x++)
01942             {
01943                 for(unsigned int y = 0; y < this->get_height(); y++)
01944                 {
01945                     double random= 2.0*(rand()-RAND_MAX/2.0)/RAND_MAX;
01946                     if(random > percentage)
01947                     {
01948                         (*(this->Img))(x, y, z, c)= 255;
01949                     }
01950
01951                     else if(random<1*percentage)
01952                     {
01953                         (*(this->Img))(x, y, z, c)= 0;
01954                     }
01955                 }
01956             }
01957         }
01958     }
01959 }
01960 }
01961 }
01962 }
01963 }
```

**3.1.3.58** void Image::set\_pixel\_value ( int *x*, int *y*, int *z*, int *c*, unsigned char *value* )

Definition at line 139 of file [image.cpp](#).

Referenced by [binarize\\_img\(\)](#), [color\\_slicing\(\)](#), [coorrelogram\(\)](#), [coorrelogram\\_ZC\(\)](#), [filter\\_average\(\)](#), [filter\\_dynamic\\_range\\_dilatation\(\)](#), [filter\\_edge\\_enhancement\\_displacement\(\)](#), [filter\\_freeman\\_0\(\)](#), [filter\\_freeman\\_1\(\)](#), [filter\\_freeman\\_2\(\)](#), [filter\\_freeman\\_3\(\)](#), [filter\\_freeman\\_4\(\)](#), [filter\\_freeman\\_5\(\)](#), [filter\\_freeman\\_6\(\)](#), [filter\\_freeman\\_7\(\)](#), [filter\\_gaussian\(\)](#), [filter\\_Gradient\\_horizontal\(\)](#), [filter\\_Gradient\\_vertical\(\)](#), [filter\\_horizontal\\_borders\(\)](#), [filter\\_kirsch\\_0\(\)](#), [filter\\_kirsch\\_135\(\)](#), [filter\\_kirsch\\_180\(\)](#), [filter\\_kirsch\\_225\(\)](#), [filter\\_kirsch\\_270\(\)](#), [filter\\_kirsch\\_315\(\)](#), [filter\\_kirsch\\_45\(\)](#), [filter\\_kirsch\\_90\(\)](#), [filter\\_Laplacian\(\)](#), [filter\\_Laplacian\\_no\\_diagonal\(\)](#), [filter\\_maximum\(\)](#), [filter\\_median\(\)](#), [filter\\_minimum\(\)](#), [filter\\_modal\(\)](#), [filter\\_order\\_statistics\(\)](#), [filter\\_Prewitt\\_E\\_W\(\)](#), [filter\\_Prewitt\\_N\\_S\(\)](#), [filter\\_Prewitt\\_NE\\_SW\(\)](#), [filter\\_Prewitt\\_NW\\_SE\(\)](#), [filter\\_vertical\\_borders\(\)](#), [gray\\_scale\(\)](#), [interpolation\(\)](#), [inverse\(\)](#), [log\\_transformation\(\)](#), [multiply\\_img\(\)](#), [power\\_law\\_transformation\(\)](#), [substract\\_img\(\)](#), [sum\\_img\(\)](#), and [variance\(\)](#).

```

00140 {
00141     (*(this->Img))(x, y, z, c)= value;
00142 }
```

**3.1.3.59** Image Image::substract\_img ( Image *image2*, int *num\_threads* )

This function substracts the pixel values of two images, that can be used to see the differences between them.

## Parameters

<a href="#"><i>Image</i></a>	image2: Is the image that will be subtracted to the original image.
------------------------------	---

## Returns

[\*Image\*](#) result: Is the result of the subtraction of both images.

Definition at line 207 of file [image.cpp](#).

```

00208 {
00209     Image result (this->get_width() , this->get_height() , this->get_depth() , this->get_spectrum(), 0);

00210
00211     if(this->get_width() == image2.get_width() && this->get_height() == image2.get_height() && this->
get_depth() == image2.get_depth() && this->get_spectrum() == image2.get_spectrum())
00212     {
00213         for(unsigned int c = 0; c < this->get_spectrum(); c++)
00214         {
00215             for(unsigned int z = 0; z < this->get_depth(); z++)
00216             {
00217                 for(unsigned int x = 0; x < this->get_width(); x++)
00218                 {
00219                     for(unsigned int y = 0; y < this->get_height(); y++)
00220                     {
00221                         unsigned char pixel= static_cast<unsigned int>(abs(this->get_pixel_value(x,y,z,c)-
image2.get_pixel_value(x,y,z,c)));
00222
00223                         result.set_pixel_value(x,y,z,c,pixel);
00224                     }
00225                 }
00226             }
00227         }
00228     }
00229     return result;
00230 }
```

### 3.1.3.60 [Image](#) [Image::sum\\_img](#) ( [Image](#) *image2*, int *num\_threads* )

Definition at line 233 of file [image.cpp](#).

```

00234 {
00235     Image result (this->get_width() , this->get_height() , this->get_depth() , this->get_spectrum(), 0);

00236
00237     if(this->get_width() == image2.get_width() && this->get_height() == image2.get_height() && this->
get_depth() == image2.get_depth() && this->get_spectrum() == image2.get_spectrum())
00238     {
00239         for(unsigned int c = 0; c < this->get_spectrum(); c++)
00240         {
00241             for(unsigned int z = 0; z < this->get_depth(); z++)
00242             {
00243                 for(unsigned int x = 0; x < this->get_width(); x++)
00244                 {
00245                     for(unsigned int y = 0; y < this->get_height(); y++)
00246                     {
00247                         unsigned char pixel;
00248                         int sum = this->get_pixel_value(x,y,z,c)+image2.get_pixel_value(x,y,z,c);
00249                         if (sum <= 255)
00250                         {
00251                             pixel = static_cast<unsigned int>(sum);
00252                         }
00253                         else
00254                         {
00255                             pixel = 255;
00256                         }
00257                         result.set_pixel_value(x,y,z,c,pixel);
00258                     }
00259                 }
00260             }
00261         }
00262     }
```



```

00262     }
00263     return result;
00264 }

```

### 3.1.3.61 Image Image::variance ( int dim, int num\_threads )

This function compute the variance of an image. The variance is given by the summation of the average multiplied by the subtraction of the average with the pixel value, squared.

#### Returns

This function returns the image interpolated.

Definition at line 2084 of file [image.cpp](#).

Referenced by [gaussian\\_noise\(\)](#).

```

02085 {
02086     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
02087 );
02088     for(unsigned int c = 0; c < this->get_spectrum(); c++)
02089     {
02090         for(unsigned int z = 0; z < this->get_depth(); z++)
02091         {
02092             for(unsigned int x = dim; x < this->get_width()-dim; x++)
02093             {
02094                 for(unsigned int y = dim; y < this->get_height()-dim; y++)
02095                 {
02096                     int sum = 0;
02097                     double variance=0;
02098                     int kernel_values[(dim*2+1)*(dim*2+1)];
02099                     int cont=0;
02100
02101                     for(unsigned int i = x-dim; i<= x+dim; i++)
02102                     {
02103                         for(unsigned int j = y-dim; j<= y+dim; j++)
02104                         {
02105                             sum += this->get_pixel_value(i, j, z, c);
02106                             kernel_values[cont]=this->get_pixel_value(i, j, z, c);
02107                             cont++;
02108                         }
02109                     }
02110
02111                     double average = sum/((dim*2+1)*(dim*2+1));
02112                     for(int i=0;i<(dim*2+1)*(dim*2+1);i++)
02113                     {
02114                         variance+=pow(kernel_values[i]-average,2)/((dim*2+1)*(dim*2+1));
02115                     }
02116
02117                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (variance);
02118                     filtered.set_pixel_value(x, y, z, c, pixel);
02119                 }
02120             }
02121         }
02122     }
02123 }
02124 }
02125 }
02126     return filtered;
02127 }

```

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

- [/home/fish/Documents/ParallelPic/Proyecto/include/ParallelPic.hh](#)
- [/home/fish/Documents/ParallelPic/Proyecto/src/image.cpp](#)
- [/home/fish/Documents/ParallelPic/Proyecto/src/ParallelPic.cpp](#)



## Chapter 4

# File Documentation

### 4.1 /home/fish/Documents/ParallelPic/Proyecto/include/ParallelPic.hh File Reference

```
#include "../include/CImg.h"
#include <string>
#include <cstdlib>
#include <iostream>
#include <omp.h>
```

#### Classes

- class [Image](#)

*The [Image](#) class is the abstraction of the image, contains a CImg object that defines the handling of the image.*

### 4.2 ParallelPic.hh

```
00001 // #include "mpi.h"
00002 #include "../include/CImg.h"
00003 #include <string>
00004 #include <cstdlib>
00005 #include <iostream>
00006 #include <omp.h>
00007
00012 using namespace cimg_library;
00016 using namespace std;
00017
00018 #ifndef IMAGE_CLASS
00019 #define IMAGE_CLASS
00020
00021 class Image
00022 {
00023 private:
00030     CImg<unsigned char> *Img;
00031     unsigned int width;
00032     unsigned int height;
00033     unsigned int depth;
00034     unsigned int spectrum;
00035     CImgList<unsigned char> complex;
00036
00037 public:
00038 // *****
00039 // ***** Constructors *****
00040 // *****
00041
```

```

00042     Image ();
00044
00045     Image (const char *const filename);
00047
00048     Image (const unsigned int width, const unsigned int height, const unsigned int depth, const
unsigned int spectrum, int value);
00053     ~Image(void);
00054
00055
00056 // *****
00057 // ***** Save and Display *****
00058 // *****
00059
00060     void save(const char *const savefilename);
00061
00062     void display(const char* message);
00063
00064 // *****
00065 // ***** GETs & SETs *****
00066 // *****
00067
00068     unsigned int get_pixel_value(int, int, int, int);
00069
00070     void set_pixel_value(int x, int y, int z, int c, unsigned char);
00071
00072     unsigned int get_width();
00073
00074     unsigned int get_height();
00075
00076     unsigned int get_depth();
00077
00078     unsigned int get_spectrum();
00079
00080 // *****
00081 // ***** Arithmetic & Logic *****
00082 // *****
00083
00084     Image subtract_img(Image,int);
00085
00086     Image sum_img(Image,int);
00087
00088     Image multiply_img(double,int);
00089
00090     Image binarize_img(unsigned int,int);
00091
00092 // *****
00093 // ***** SPACE DOMAIN FILTERS *****
00094 // *****
00095
00096 // *****
00097 // ***** Sharpening Spatial Filters *****
00098 // *****
00099
00100     Image filter_Laplacian(int);
00101
00102     Image filter_Laplacian_no_diagonal(int);
00103
00104     Image filter_Gradient_vertical(int);
00105
00106     Image filter_Gradient_horizontal(int);
00107
00108     Image filter_Prewitt_N_S(int);
00109
00110     Image filter_Prewitt_NE_SW(int);
00111
00112     Image filter_Prewitt_E_W(int);
00113
00114     Image filter_Prewitt_NW_SE(int);
00115
00116     Image filter_edge_enhancement_displacement(unsigned int, unsigned int,int);
00117
00118     Image filter_horizontal_borders(int);
00119
00120     Image filter_vertical_borders(int);
00121
00122 // *****
00123 // ***** Smoothing Spatial Filters *****
00124 // *****
00125
00126     Image filter_median(int,int);
00127

```

```

00128     Image filter_average(int,int);
00129
00130     Image average_omp(int,int);
00131
00132     Image filter_gaussian(int, int,int);
00133
00134     Image filter_modal(int,int);
00135
00136     Image median_omp(int,int);
00137
00138 // *****
00139 // ***** Dot to Dot Transformations *****
00140 // *****
00141     Image filter_dynamic_range_dilatation(unsigned char, unsigned char, double, double, double, int );
00142
00143     Image inverse(int);
00144
00145     Image log_transformation(int);
00146
00147     Image power_law_transformation(double exponent,int);
00148
00149     Image color_slicing (unsigned char [], unsigned char [], unsigned char [],int);
00150
00151 // *****
00152 // ***** HISTOGRAM AND EQUALIZATION *****
00153 // *****
00154
00155     int* get_histogram(unsigned int c, unsigned int z);
00156
00157     void plot_histogram(int, const char* title);
00158
00159     int* histogram_equalization(int*, const char* title);
00160
00161     CImg<float> autocovariance(int, int,int);
00162
00163     void plot_histogram_equalization(int, const char* title);
00164
00165
00166
00167
00168 // *****
00169 // ***** OTHER TRANSFORMATIONS *****
00170 // *****
00171
00172
00173     Image filter_order_statistics(int dim, int order,int);
00174
00175     Image variance(int, int);
00176
00177     Image filter_kirsch_0(int);
00178
00179     Image filter_kirsch_45(int);
00180
00181     Image filter_kirsch_90(int);
00182
00183     Image filter_kirsch_135(int);
00184
00185     Image filter_kirsch_180(int);
00186
00187     Image filter_kirsch_225(int);
00188
00189     Image filter_kirsch_270(int);
00190
00191     Image filter_kirsch_315(int);
00192
00193     Image filter_freeman_0(int);
00194
00195     Image filter_freeman_1(int);
00196
00197     Image filter_freeman_2(int);
00198
00199     Image filter_freeman_3(int);
00200
00201     Image filter_freeman_4(int);
00202
00203     Image filter_freeman_5(int);
00204
00205     Image filter_freeman_6(int);
00206
00207     Image filter_freeman_7(int);
00208

```

```

00209     Image filter_maximum(int);
00210
00211     Image filter_minimum(int);
00212
00213     Image gray_scale(int);
00214
00215
00216 // *****
00217 // ***** NOISES *****
00218 // *****
00219
00220     void gaussian_noise(double, int);
00221     void salt_pepper(double, int);
00222
00223     Image interpolation(int);
00224
00225     Image coorrelogram(unsigned int,unsigned int,int);
00226
00227     Image coorrelogram_ZC(unsigned int,unsigned int,unsigned int, unsigned int, int);
00228
00229     Image coorrelogram_par(unsigned int,unsigned int,unsigned int, unsigned int);
00230
00231     Image rgb_hsv();
00232
00233
00234 };
00235
00236 #endif

```

### 4.3 /home/fish/Documents/ParallelPic/Proyecto/src/image.cpp File Reference

```
#include "../include/image.hh"
```

### 4.4 image.cpp

```

00001 #include "../include/image.hh"
00002
00003
00008 // *****
00009 // ***** CONSTRUCTORS *****
00010 // *****
00011
00017 Image::Image()
00018 {
00019     this->Img = new CImg<unsigned char>();
00020     this->width = 0;
00021     this->height = 0;
00022     this->depth = 0;
00023     this->spectrum = 0;
00024 }
00025
00032 Image::Image(const char *const filename)
00033 {
00034     this->Img = new CImg<unsigned char>(filename);
00036     this->width = this->Img->width();
00038     this->height = this->Img->height();
00040     this->depth = this->Img->depth();
00042     this->spectrum = this->Img->spectrum();
00044 }
00049 Image::Image(const unsigned int width, const unsigned int height, const unsigned int depth,
const unsigned int spectrum, int value)
00050 {
00051     this->Img = new CImg<unsigned char>(width, height, depth, spectrum, value);
00052     this->width = width;
00053     this->height = height;
00054     this->depth = depth;
00055     this->spectrum = spectrum;
00056 }
00060 Image::~Image(void)
00061 {
00062

```

```

00063 }
00064 // *****
00065 // ***** SAVE & DISPLAY *****
00066 // *****
00072 void Image:: save(const char *const savefilename)
00073 {
00074     this->Img->save(savefilename);
00075 }
00076
00077 void Image :: display(const char* message)
00078 {
00079     CImgDisplay main (*(this->Img), message);
00080     while(!main.is_closed())
00081     {
00082         main.wait();
00083     }
00084 }
00085
00086 // *****
00087 // ***** GETs & SETs *****
00088 // *****
00089
00094 unsigned int Image:: get_width()
00095 {
00096     return this->width;
00097 }
00102 unsigned int Image:: get_height()
00103 {
00104     return this->height;
00105 }
00106
00111 unsigned int Image:: get_depth()
00112 {
00113     return this->depth;
00114 }
00115
00120 unsigned int Image:: get_spectrum()
00121 {
00122     return this->spectrum;
00123 }
00124
00125
00130 unsigned int Image:: get_pixel_value(int x, int y, int z, int c)
00131 {
00132     return this->Img->get_vector_at(x, y, z)[c];
00133 }
00134
00139 void Image:: set_pixel_value(int x, int y, int z, int c, unsigned char value)
00140 {
00141     (*(this->Img))(x, y, z, c)= value;
00142 }
00143
00144
00145 // *****
00146 // ***** FILTER *****
00147 // *****
00148
00149
00161 Image Image :: filter (int kernel [], int dim, float normalizer)
00162 {
00163     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
00164
00165     int m = (int)(dim-1)/2;
00166
00167     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00168     {
00169         for(unsigned int z = 0; z < this->get_depth(); z++)
00170         {
00171             for(unsigned int x = m; x < this->get_width()-m; x++)
00172             {
00173                 for(unsigned int y = m; y < this->get_height()-m; y++)
00174                 {
00175                     double sum_values = 0;
00176
00177                     for(unsigned int i = (x-m); i <= (x+m); i++)
00178                     {
00179                         for(unsigned int j = (y-m); j<= (y+m); j++)
00180                         {
00181                             sum_values += (this->get_pixel_value(i, j, z, c)) * (kernel[ (i-x+m)*dim + (j-y
+m)]);

```

```

00182         }
00183     }
00184
00185     unsigned char pixel = static_cast<unsigned char> (abs(sum_values/ normalizer));
00186
00187     filtered.set_pixel_value(x, y, z, c, pixel);
00188 }
00189
00190 }
00191
00192 }
00193 }
00194     return filtered;
00195 }
00196
00197
00198 // *****
00199 // *****Arithmetic and Logic *****
00200 // *****
00201
00207 Image Image :: subtract_img(Image image2)
00208 {
00209     Image result (this->get_width() , this->get_height() , this->get_depth() , this->get_spectrum(), 0);
00210
00211     if(this->get_width() == image2.get_width() && this->get_height() == image2.get_height() && this->
get_depth() == image2.get_depth() && this->get_spectrum() == image2.get_spectrum())
00212     {
00213         for(unsigned int c = 0; c < this->get_spectrum(); c++)
00214         {
00215             for(unsigned int z = 0; z < this->get_depth(); z++)
00216             {
00217                 for(unsigned int x = 0; x < this->get_width(); x++)
00218                 {
00219                     for(unsigned int y = 0; y < this->get_height(); y++)
00220                     {
00221                         unsigned char pixel= static_cast<unsigned int>(abs(this->get_pixel_value(x,y,z,c)-
image2.get_pixel_value(x,y,z,c)));
00222
00223                         result.set_pixel_value(x,y,z,c,pixel);
00224                     }
00225                 }
00226             }
00227         }
00228     }
00229     return result;
00230 }
00231
00232
00233 Image Image :: sum_img(Image image2)
00234 {
00235     Image result (this->get_width() , this->get_height() , this->get_depth() , this->get_spectrum(), 0);
00236
00237     if(this->get_width() == image2.get_width() && this->get_height() == image2.get_height() && this->
get_depth() == image2.get_depth() && this->get_spectrum() == image2.get_spectrum())
00238     {
00239         for(unsigned int c = 0; c < this->get_spectrum(); c++)
00240         {
00241             for(unsigned int z = 0; z < this->get_depth(); z++)
00242             {
00243                 for(unsigned int x = 0; x < this->get_width(); x++)
00244                 {
00245                     for(unsigned int y = 0; y < this->get_height(); y++)
00246                     {
00247                         unsigned char pixel;
00248                         int sum = this->get_pixel_value(x,y,z,c)+image2.get_pixel_value(x,y,z,c);
00249                         if (sum <= 255)
00250                         {
00251                             pixel = static_cast<unsigned int>(sum);
00252                         }
00253                         else
00254                         {
00255                             pixel = 255;
00256                         }
00257                         result.set_pixel_value(x,y,z,c,pixel);
00258                     }
00259                 }
00260             }
00261         }
00262     }

```



```

00263     return result;
00264 }
00265
00266
00273 Image Image::multiply_img(double multiplier)
00274 {
00275     Image result (this->get_width() , this->get_height() , this->get_depth() , this->get_spectrum(), 0);
00276     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00277     {
00278         for(unsigned int z = 0; z < this->get_depth(); z++)
00279         {
00280             for(unsigned int x = 0; x < this->get_width(); x++)
00281             {
00282                 for(unsigned int y = 0; y < this->get_height(); y++)
00283                 {
00284                     unsigned char pixel= static_cast<unsigned int>(abs(this->get_pixel_value(x,y,z,c)*
multiplier));
00285                     if (pixel >255)
00286                         pixel = 255;
00287                     result.set_pixel_value(x,y,z,c,pixel);
00288                 }
00289             }
00290         }
00291     }
00292     return result;
00293 }
00294
00301 Image Image::binarize_img(unsigned int cutoff_value)
00302 {
00303     Image result (this->get_width() , this->get_height() , this->get_depth() , this->get_spectrum(), 0);
00304     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00305     {
00306         for(unsigned int z = 0; z < this->get_depth(); z++)
00307         {
00308             for(unsigned int x = 0; x < this->get_width(); x++)
00309             {
00310                 for(unsigned int y = 0; y < this->get_height(); y++)
00311                 {
00312                     unsigned char pixel= static_cast<unsigned int>(this->get_pixel_value(x,y,z,c));
00313                     if(pixel >= cutoff_value)
00314                         pixel=255;
00315                     else
00316                         pixel=0;
00317                     result.set_pixel_value(x,y,z,c,pixel);
00318                 }
00319             }
00320         }
00321     }
00322 }
00323
00324
00325     return result;
00326 }
00327
00328
00329 // *****
00330 // ***** SPACE DOMAIN FILTERS *****
00331 // *****
00332
00333
00334
00335
00336 // *****
00337 // ***** Sharpening Spatial Filters *****
00338 // *****
00339
00347 Image Image::filter_Laplacian()
00348 {
00349     Image filtered (this->get_width() , this->get_height() , this->get_depth() , this->get_spectrum(), 0
);
00350
00351     int m = 1;
00352
00353     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00354     {
00355         for(unsigned int z = 0; z < this->get_depth(); z++)
00356         {
00357             for(unsigned int x = m; x < this->get_width()-m; x++)
00358             {
00359                 for(unsigned int y = m; y < this->get_height()-m; y++)
00360                 {

```

```

00361         int sum = 0;
00362
00363         for (unsigned int i = 0 ; i < 3; i++)
00364         {
00365             for(unsigned int j = 0 ; j < 3; j++)
00366             {
00367                 sum += -this->get_pixel_value(x+i-1, y+i-1, z, c);
00368             }
00369         }
00370
00371         sum += 9*(this->get_pixel_value(x,y,z,c));
00372
00373         if (sum > 255 || sum < -255)
00374         {
00375             sum = 255;
00376         }
00377         unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
00378         filtered.set_pixel_value(x, y, z, c, pixel);
00379     }
00380 }
00381 }
00382 }
00383 }
00384 }
00385 return filtered;
00386 }
00387
00393 Image Image :: filter_Laplacian_no_diagonal()
00394 {
00395     //int kernel[9] = {0, -1, 0, -1, 4, -1, 0, -1, 0};
00396     //return (this->filter(kernel, 3, 1));
00397
00398     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00399 );
00400
00401     int m = 1;
00402
00403     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00404     {
00405         for(unsigned int z = 0; z < this->get_depth(); z++)
00406         {
00407             for(unsigned int x = m; x < this->get_width()-m; x++)
00408             {
00409                 for(unsigned int y = m; y < this->get_height()-m; y++)
00410                 {
00411                     int sum = 4*(this->get_pixel_value(x,y,z,c)) - (this->get_pixel_value(x-1,y,z,c) +
this->get_pixel_value(x+1,y,z,c) + this->get_pixel_value(x,y-1,z,c) +this->get_pixel_value(x,y+1,z,c));
00412
00413                     if (sum > 255 || sum < -255)
00414                     {
00415                         sum = 255;
00416                     }
00417                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
00418                     filtered.set_pixel_value(x, y, z, c, pixel);
00419                 }
00420             }
00421         }
00422     }
00423 }
00424 }
00425 return filtered;
00426 }
00427 }
00428
00436 Image Image :: filter_Gradient_horizontal()
00437 {
00438     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00439 );
00440
00441     int m = 1;
00442
00443     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00444     {
00445         for(unsigned int z = 0; z < this->get_depth(); z++)
00446         {
00447             for(unsigned int x = m; x < this->get_width()-m; x++)
00448             {
00449                 for(unsigned int y = m; y < this->get_height()-m; y++)
00450                 {
00451                     int sum = this->get_pixel_value(x-1, y-1, z, c) + 2*(this->get_pixel_value(x, y-1, z, c)

```

```

    )) + this->get_pixel_value(x+1, y-1, z, c) - (this->get_pixel_value(x-1, y+1, z, c) + 2*(this->
get_pixel_value(x, y+1, z, c)) + this->get_pixel_value(x+1, y+1, z, c));
00451         if (sum > 255 || sum < -255)
00452         {
00453             sum = 255;
00454         }
00455         unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
00456         filtered.set_pixel_value(x, y, z, c, pixel);
00457     }
00458 }
00459 }
00460
00461 }
00462 }
00463     return filtered;
00464 }
00465
00473 Image Image :: filter_Gradient_vertical()
00474 {
00475     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
00476
00477     int m = 1;
00478
00479     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00480     {
00481         for(unsigned int z = 0; z < this->get_depth(); z++)
00482         {
00483             for(unsigned int x = m; x < this->get_width()-m; x++)
00484             {
00485                 for(unsigned int y = m; y < this->get_height()-m; y++)
00486                 {
00487                     int sum = get_pixel_value(x-1, y-1, z, c) + 2*get_pixel_value(x-1, y, z, c) +
get_pixel_value(x-1, y+1, z, c) - (get_pixel_value(x+1, y-1, z, c) + 2*get_pixel_value(x+1, y, z, c) +
get_pixel_value(x+1, y+1, z, c));
00488                     if (sum > 255 || sum < -255)
00489                     {
00490                         sum = 255;
00491                     }
00492                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
00493                     filtered.set_pixel_value(x, y, z, c, pixel);
00494                 }
00495             }
00496         }
00497     }
00498 }
00499 }
00500     return filtered;
00501 }
00502
00520 Image Image :: filter_Prewitt_N_S()
00521 {
00522     //int kernel[9] = {1, 1, 1, 0, 0, 0, -1, -1, -1};
00523
00524     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
00525
00526     int m = 1;
00527
00528     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00529     {
00530         for(unsigned int z = 0; z < this->get_depth(); z++)
00531         {
00532             for(unsigned int x = m; x < this->get_width()-m; x++)
00533             {
00534                 for(unsigned int y = m; y < this->get_height()-m; y++)
00535                 {
00536                     int sum = get_pixel_value(x-1, y-1, z, c) + get_pixel_value(x, y-1, z, c) +
get_pixel_value(x+1, y-1, z, c) - (get_pixel_value(x-1, y+1, z, c) + get_pixel_value(x, y+1, z, c) + get_pixel_value(x+
1, y+1, z, c));
00537                     if (sum > 255 || sum < -255)
00538                     {
00539                         sum = 255;
00540                     }
00541                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
00542                     filtered.set_pixel_value(x, y, z, c, pixel);
00543                 }
00544             }
00545         }
00546     }
00547 }

```

```

00548     }
00549
00550     return filtered;
00551 }
00552 }
00553
00554 Image Image ::filter_Prewitt_NE_SW()
00555 {
00556     //int kernel[9] = {0, 1, 1, -1, 0, 1, -1, -1, 0};
00557     //return (this->filter(kernel, 3, 1));
00558
00559     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00560 );
00561
00562     int m = 1;
00563
00564     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00565     {
00566         for(unsigned int z = 0; z < this->get_depth(); z++)
00567         {
00568             for(unsigned int x = m; x < this->get_width()-m; x++)
00569             {
00570                 for(unsigned int y = m; y < this->get_height()-m; y++)
00571                 {
00572                     int sum = get_pixel_value(x, y-1, z, c) + get_pixel_value(x+1, y-1, z, c) +
get_pixel_value(x+1, y, z, c) - (get_pixel_value(x-1, y, z, c) + get_pixel_value(x-1, y+1, z, c) + get_pixel_value(x, y
+1, z, c));
00573                     if (sum > 255 || sum < -255)
00574                     {
00575                         sum = 255;
00576                     }
00577                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
00578                     filtered.set_pixel_value(x, y, z, c, pixel);
00579                 }
00580             }
00581         }
00582     }
00583 }
00584 }
00585
00586     return filtered;
00587 }
00588 }
00589
00590 Image Image ::filter_Prewitt_E_W()
00591 {
00592     //int kernel[9] = {1, 0, -1, 1, 0, -1, 1, 0, -1};
00593     //return (this->filter(kernel, 3, 1));
00594
00595     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00596 );
00597
00598     int m = 1;
00599
00600     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00601     {
00602         for(unsigned int z = 0; z < this->get_depth(); z++)
00603         {
00604             for(unsigned int x = m; x < this->get_width()-m; x++)
00605             {
00606                 for(unsigned int y = m; y < this->get_height()-m; y++)
00607                 {
00608                     int sum = get_pixel_value(x-1, y-1, z, c) + get_pixel_value(x-1, y, z, c) +
get_pixel_value(x-1, y+1, z, c) - (get_pixel_value(x+1, y-1, z, c) + get_pixel_value(x+1, y, z, c) + get_pixel_value(x+
1, y+1, z, c));
00609                     if (sum > 255 || sum < -255)
00610                     {
00611                         sum = 255;
00612                     }
00613                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
00614                     filtered.set_pixel_value(x, y, z, c, pixel);
00615                 }
00616             }
00617         }
00618     }
00619 }
00620 }
00621
00622     return filtered;

```

```

00623
00624 }
00625
00626 Image Image ::filter_Prewitt_NW_SE()
00627 {
00628     //int kernel[9] = {-1, -1, 0, -1, 0, 1, 0, 1, 1};
00629     // (this->filter(kernel, 3, 1));
00630
00631     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00632 );
00633
00634     int m = 1;
00635
00636     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00637     {
00638         for(unsigned int z = 0; z < this->get_depth(); z++)
00639         {
00640             for(unsigned int x = m; x < this->get_width()-m; x++)
00641             {
00642                 for(unsigned int y = m; y < this->get_height()-m; y++)
00643                 {
00644                     int sum = get_pixel_value(x-1, y-1, z, c) + get_pixel_value(x-1, y, z, c) +
get_pixel_value(x, y-1, z, c) - (get_pixel_value(x+1, y, z, c) + get_pixel_value(x+1, y+1, z, c) + get_pixel_value(x, y
+1, z, c));
00645                     if (sum > 255 || sum < -255)
00646                     {
00647                         sum = 255;
00648                     }
00649                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
00650                     filtered.set_pixel_value(x, y, z, c, pixel);
00651                 }
00652             }
00653         }
00654     }
00655 }
00656
00657     return filtered;
00658
00659 }
00660 }
00661
00662 Image Image ::filter_edge_enhancement_displacement(unsigned
int horizontal_dis, unsigned int vertical_dis)
00663 {
00664     Image result (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0);
00665
00666     if((horizontal_dis < this->get_width()) && (vertical_dis < this->get_height()))
00667     {
00668         for(unsigned int c = 0; c < this->get_spectrum(); c++)
00669         {
00670             for(unsigned int z = 0; z < this->get_depth(); z++)
00671             {
00672                 for(unsigned int x = horizontal_dis; x < this->get_width(); x++)
00673                 {
00674                     for(unsigned int y = vertical_dis; y < this->get_height(); y++)
00675                     {
00676                         unsigned char value = static_cast<unsigned char>(abs(this->get_pixel_value(x,y,z,c)
- this->get_pixel_value(x-horizontal_dis, y-vertical_dis, z, c)));
00677                         result.set_pixel_value(x,y,z,c, value);
00678                     }
00679                 }
00680             }
00681         }
00682     }
00683     return result;
00684 }
00685
00686 Image Image :: filter_vertical_borders()
00687 {
00688     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00689 );
00690
00691     int m = 1;
00692
00693     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00694     {
00695         for(unsigned int z = 0; z < this->get_depth(); z++)
00696         {
00697             for(unsigned int x = m; x < this->get_width()-m; x++)

```

```

00697         {
00698             for(unsigned int y = 0; y < this->get_height()-m; y++)
00699             {
00700                 unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(this->
get_pixel_value(x-1, y, z, c) - get_pixel_value(x+1, y, z, c)));
00701                 filtered.set_pixel_value(x, y, z, c, pixel);
00702             }
00703         }
00704     }
00705 }
00706 }
00707 }
00708     return filtered;
00709 }
00710
00711 Image Image :: filter_horizontal_borders()
00712 {
00713     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
00714
00715     int m = 1;
00716
00717     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00718     {
00719         for(unsigned int z = 0; z < this->get_depth(); z++)
00720         {
00721             for(unsigned int x = 0; x < this->get_width()-m; x++)
00722             {
00723                 for(unsigned int y = m; y < this->get_height()-m; y++)
00724                 {
00725                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(this->
get_pixel_value(x, y-1, z, c) - get_pixel_value(x, y+1, z, c)));
00726                     filtered.set_pixel_value(x, y, z, c, pixel);
00727                 }
00728             }
00729         }
00730     }
00731 }
00732 }
00733     return filtered;
00734 }
00735
00736 // *****
00737 // ***** Smoothing Spatial Filters *****
00738 // *****
00739
00740
00741 Image Image :: filter_median (int dim)
00742 {
00743     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
00744
00745     //int kernel [dim*dim];
00746
00747     int m = (dim-1)/2;
00748     unsigned char pixel_values [dim*dim];
00749     unsigned char temp;
00750
00751     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00752     {
00753         for(unsigned int z = 0; z < this->get_depth(); z++)
00754         {
00755             for(unsigned int x = m; x < this->get_width(); x++)
00756             {
00757                 for(unsigned int y = m; y < this->get_height(); y++)
00758                 {
00759                     for(unsigned int i = x-m; i < x+m; i++)
00760                     {
00761                         for(unsigned int j = y-m; j < y+m; j++)
00762                         {
00763                             pixel_values [(i-x+m)*dim + (j-y+m)] = this->get_pixel_value(i, j, z, c);
00764                         }
00765                     }
00766                 }
00767                 for(int k=0; k<dim*dim ; k++)
00768                 {
00769                     for(int p=k+1 ; p<dim*dim ; p++)
00770                     {
00771                         if(pixel_values[p] < pixel_values[k])
00772                         {
00773                             // Intercambiar los valores

```

```

00780             temp = pixel_values[k];
00781             pixel_values[k] = pixel_values[p];
00782             pixel_values[p] = temp;
00783         }
00784     }
00785 }
00786 unsigned char pixel = pixel_values[((dim*dim-1)/2)-1];
00787 filtered.set_pixel_value(x, y, z, c, pixel);
00788 }
00789 }
00790 }
00791 }
00792 }
00793 }
00794 return filtered;
00795 }
00796
00803 Image Image :: filter_average(int dim)
00804 {
00805     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00806 );
00807     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00808     {
00809         for(unsigned int z = 0; z < this->get_depth(); z++)
00810         {
00811             for(unsigned int x = dim; x < this->get_width()-dim; x++)
00812             {
00813                 for(unsigned int y = dim; y < this->get_height()-dim; y++)
00814                 {
00815                     int sum = 0;
00816                     for(unsigned int i = x-dim; i<= x+dim; i++)
00817                     {
00818                         for(unsigned int j = y-dim; j<= y+dim; j++)
00819                         {
00820                             sum += this->get_pixel_value(i, j, z, c);
00821                         }
00822                     }
00823
00824                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (sum/((dim*2+1)*(dim*2+
00825 1)));
00826                     filtered.set_pixel_value(x, y, z, c, pixel);
00827                 }
00828             }
00829         }
00830     }
00831 }
00832 return filtered;
00833 }
00834
00841 Image Image :: filter_gaussian(int o, int dim_kernel)
00842 {
00843     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00844 );
00845     double kernel[dim_kernel*dim_kernel];
00846
00847     int m = (dim_kernel-1)/2;
00848
00849     double gaussian =1/pow(3.1415*2*o*o,0.5);
00850
00851     for(int i =-m; i <=m; i++)
00852     {
00853         for(int j =-m; j<=+m; j++)
00854         {
00855             double exp= -(i*i+j*j)*0.5/(o*o);
00856             kernel[(i+m)*dim_kernel + (j+m)]=gaussian*pow(2.7,exp);
00857         }
00858     }
00859
00860
00861
00862     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00863     {
00864         for(unsigned int z = 0; z < this->get_depth(); z++)
00865         {
00866             for(unsigned int x = m; x < this->get_width(); x++)
00867             {
00868                 for(unsigned int y = m; y < this->get_height(); y++)

```

```

00870         {
00871             int cont=0;
00872             unsigned char pixel=0;
00873
00874             for(unsigned int i = x-m; i < x+m; i++)
00875             {
00876                 for(unsigned int j = y-m; j < y+m; j++)
00877                 {
00878                     pixel+= this->get_pixel_value(i, j, z, c)*(kernel[cont]);
00879                     cont++;
00880                 }
00881             }
00882             filtered.set_pixel_value(x, y, z, c, (pixel/2));
00883         }
00884     }
00885 }
00886
00887 }
00888 }
00889 return filtered;
00890 }
00891
00892 Image Image :: filter_modal(int dim)
00893 {
00894     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00895 );
00896     unsigned char pixel_values[dim*dim];
00897     unsigned char moda;
00898     unsigned char average=0;
00899     int m=(dim-1)/2;
00900     unsigned char copy_pixels[dim*dim];
00901
00902     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00903     {
00904         for(unsigned int z = 0; z < this->get_depth(); z++)
00905         {
00906             for(unsigned int x = m; x < this->get_width(); x++)
00907             {
00908                 for(unsigned int y = m; y < this->get_height(); y++)
00909                 {
00910                     for(unsigned int i = x-m; i < x+m; i++)
00911                     {
00912                         for(unsigned int j = y-m; j < y+m; j++)
00913                         {
00914                             pixel_values [(i-x+m)*dim + (j-y+m)]= this->get_pixel_value(i, j, z, c);
00915
00916                             int frequency[dim*dim];
00917                             moda=0;
00918
00919                             for(int k=0;k<dim*dim;k++)
00920                             {
00921                                 copy_pixels[k]= pixel_values[k];
00922                                 frequency[k]=0;
00923                             }
00924
00925                             for(int p=0;p<dim*dim;p++)
00926                             {
00927                                 for(int q=p+1;q<dim*dim;q++)
00928                                 {
00929                                     if(copy_pixels[p]==pixel_values[q]){
00930                                         frequency[p]++;
00931                                     }
00932                                 }
00933                             }
00934
00935                             for(int s=0; s<dim*dim ; s++)
00936                             {
00937                                 for(int e=s+1 ; e<dim*dim ; e++)
00938                                 {
00939                                     if(frequency[e] < frequency[s])
00940                                     {
00941                                         moda = copy_pixels[s];
00942                                         average=copy_pixels[s];
00943                                     }
00944                                 }
00945                             }
00946                         }
00947                     }
00948                 }
00949             }
00950         }
00951     }
00952 }

```



```

00956
00957
00958         if(moda==0)
00959         {
00960             for(int k=0;k<dim*dim;k++)
00961             {
00962                 moda += pixel_values[k];
00963             }
00964             average=(moda/dim);
00965         }
00966
00967     }
00968 }
00969
00970
00971     filtered.set_pixel_value(x, y, z, c, average);
00972 }
00973
00974 }
00975
00976 }
00977 }
00978
00979 return filtered;
00980 }
00981 }
00982
00983
00984
00985 // *****
00986 // ***** Dot to Dot Transformations *****
00987 // *****
00988
00994 Image Image ::inverse()
00995 {
00996     Image inverted (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00997 );
00998     for(unsigned int c = 0; c < this->get_spectrum(); c++)
00999     {
01000         for(unsigned int z = 0; z < this->get_depth(); z++)
01001         {
01002             for(unsigned int x = 0; x < this->get_width(); x++)
01003             {
01004                 for(unsigned int y = 0; y < this->get_height(); y++)
01005                 {
01006                     unsigned char pixel= static_cast<unsigned int>(255-this->get_pixel_value(x,y,z,c));
01007                     inverted.set_pixel_value(x,y,z,c,pixel);
01008                 }
01009             }
01010         }
01011     }
01012     return inverted;
01013 }
01014
01020 Image Image :: log_transformation()
01021 {
01022     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01023 );
01024     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01025     {
01026         for(unsigned int z = 0; z < this->get_depth(); z++)
01027         {
01028             for(unsigned int x = 0; x < this->get_width(); x++)
01029             {
01030                 for(unsigned int y = 0; y < this->get_height(); y++)
01031                 {
01032                     unsigned char pixel = static_cast<unsigned char>((255/log(256)) * log(1+this->
get_pixel_value(x, y, z, c)));
01033
01034                     filtered.set_pixel_value(x,y,z,c, pixel);
01035                 }
01036             }
01037         }
01038     }
01039     return filtered;
01040 }
01041
01052 Image Image ::filter_dynamic_range_dilatation(unsigned char a,
unsigned char b, double alpha, double beta, double gamma)

```

```

01053 {
01054     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01055 );
01056     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01057     {
01058         for(unsigned int z = 0; z < this->get_depth(); z++)
01059         {
01060             for(unsigned int x = 0; x < this->get_width(); x++)
01061             {
01062                 for(unsigned int y = 0; y < this->get_height(); y++)
01063                 {
01064                     unsigned char pixel=0;
01065
01066                     if(this->get_pixel_value(x,y,z,c)<a)
01067                         pixel =abs(alpha*this->get_pixel_value(x,y,z,c));
01068
01069                     else if(this->get_pixel_value(x,y,z,c)>=a && this->get_pixel_value(x,y,z,c)<b)
01070                         pixel=abs(beta*(this->get_pixel_value(x,y,z,c)-a)+alpha*a);
01071
01072                     else if(this->get_pixel_value(x,y,z,c)<=b)
01073
01074                         pixel=abs(gamma*(this->get_pixel_value(x,y,z,c)-b)+((beta*(b-a))+alpha*a));
01075                     filtered.set_pixel_value(x,y,z,c,static_cast<unsigned int>(pixel));
01076                 }
01077             }
01078         }
01079     }
01080     return filtered;
01081 }
01082
01083
01091 Image Image :: power_law_transformation(double exponent)
01092 {
01093     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01094 );
01095     double k = (pow(255, 1-exponent));
01096
01097     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01098     {
01099         for(unsigned int z = 0; z < this->get_depth(); z++)
01100         {
01101             for(unsigned int x = 0; x < this->get_width(); x++)
01102             {
01103                 for(unsigned int y = 0; y < this->get_height(); y++)
01104                 {
01105                     double power_law = k * pow( (this->get_pixel_value(x,y,z,c)) , exponent);
01106                     unsigned char pixel = static_cast<unsigned char>(power_law);
01107                     filtered.set_pixel_value(x, y, z, c, pixel);
01108                 }
01109             }
01110         }
01111     }
01112 }
01113 }
01114 return filtered;
01115 }
01116
01124 Image Image :: color_slicing(unsigned char color1[], unsigned char color2[],
01125 unsigned char neutral[])
01126 {
01127     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01128 );
01129     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01130     {
01131         for(unsigned int z = 0; z < this->get_depth(); z++)
01132         {
01133             for(unsigned int x = 0; x < this->get_width(); x++)
01134             {
01135                 for(unsigned int y = 0; y < this->get_height(); y++)
01136                 {
01137                     unsigned char pixel = this->get_pixel_value(x,y,z,c);
01138                     if(pixel > color1[c] && pixel < color2[c] )
01139                     {
01140                         filtered.set_pixel_value(x, y, z, c, pixel);
01141                     }
01142                     else
01143                     {
01144                         filtered.set_pixel_value(x,y,z,c, neutral[c]);
01145                     }
01146                 }
01147             }
01148         }
01149     }
01150     return filtered;
01151 }

```

```

01144         }
01145     }
01146
01147     }
01148
01149     }
01150 }
01151
01152     return filtered;
01153
01154 }
01155
01156 // *****
01157 // ***** HISTOGRAM AND EQUALIZATION *****
01158 // *****
01159
01167 int* Image :: get_histogram(unsigned int c, unsigned int z)
01168 {
01169     int histogram [256];
01170     for(int i = 0; i<256; i++)
01171     {
01172         histogram[i] = 0;
01173     }
01174
01175     if (c < this->get_spectrum() && z < this->get_depth())
01176     {
01177         for(unsigned int x = 0; x < this->get_width(); x++)
01178         {
01179             for(unsigned int y = 0; y < this->get_height(); y++)
01180             {
01181                 unsigned char pixel_value = this->get_pixel_value(x,y,z,c);
01182                 (histogram[pixel_value])++;
01183             }
01184         }
01185     }
01186
01187     int* histogram_pointer = histogram;
01188
01189     return histogram_pointer;
01190 }
01191
01197 void Image :: plot_histogram(int levels,const char* title)
01198 {
01199     CImg<unsigned char> img = this->Img->histogram(levels);
01200
01201     CImgDisplay main_display (*(this->Img), title);
01202
01203     img.display_graph(main_display, 3, 1, "Pixel Intensity", 0, 0, "Frequency", 0, 0);
01204 }
01205
01211 void Image :: plot_histogram_equalization(int levels, const char* title
01212 )
01213 {
01214     CImg<unsigned char> img = this->Img->equalize(levels);
01215
01216     CImgDisplay main_display (*(this->Img), title);
01217
01218     img.display_graph(main_display, 3, 1, "Pixel Intensity", 0, 0, "Frequency", 0, 0);
01219 }
01220 // *****
01221 // ***** OTHER TRANSFORMATIONS *****
01222 // *****
01223
01228 Image Image ::filter_kirsch_0()
01229 {
01230     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01231 );
01232
01233     int m = 1;
01234
01235     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01236     {
01237         for(unsigned int z = 0; z < this->get_depth(); z++)
01238         {
01239             for(unsigned int x = m; x < this->get_width()-m; x++)
01240             {
01241                 for(unsigned int y = m; y < this->get_height()-m; y++)
01242                 {
01243                     int sum = -3*(get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x-1, y, z, c)+
get_pixel_value(x-1, y+1, z, c)+get_pixel_value(x, y-1, z, c)+get_pixel_value(x, y+1, z, c))+5*(get_pixel_value(x+1, y-

```

```

1, z, c)+get_pixel_value(x+1, y, z, c)+get_pixel_value(x+1, y+1, z, c));
01243         if (sum > 255 || sum < -255)
01244         {
01245             sum = 255;
01246         }
01247         unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01248         filtered.set_pixel_value(x, y, z, c, pixel);
01249     }
01250 }
01251 }
01252 }
01253 }
01254 }
01255 }
01256 return filtered;
01257 }
01258 }
01259
01264 Image Image ::filter_kirsch_45()
01265 {
01266     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
01267     int m = 1;
01268
01269     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01270     {
01271         for(unsigned int z = 0; z < this->get_depth(); z++)
01272         {
01273             for(unsigned int x = m; x < this->get_width()-m; x++)
01274             {
01275                 for(unsigned int y = m; y < this->get_height()-m; y++)
01276                 {
01277                     int sum = -3*(get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x-1, y, z, c)+
get_pixel_value(x-1, y+1, z, c)+get_pixel_value(x+1, y+1, z, c)+get_pixel_value(x, y+1, z, c))+5*(get_pixel_value(x+1,
y-1, z, c)+get_pixel_value(x+1, y, z, c)+get_pixel_value(x, y-1, z, c));
01278                     if (sum > 255 || sum < -255)
01279                     {
01280                         sum = 255;
01281                     }
01282                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01283                     filtered.set_pixel_value(x, y, z, c, pixel);
01284                 }
01285             }
01286         }
01287     }
01288 }
01289 }
01290
01291 return filtered;
01292 }
01293 }
01294 }
01295
01300 Image Image ::filter_kirsch_90()
01301 {
01302     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
01303     int m = 1;
01304
01305     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01306     {
01307         for(unsigned int z = 0; z < this->get_depth(); z++)
01308         {
01309             for(unsigned int x = m; x < this->get_width()-m; x++)
01310             {
01311                 for(unsigned int y = m; y < this->get_height()-m; y++)
01312                 {
01313                     int sum = -3*(get_pixel_value(x-1, y, z, c)+get_pixel_value(x-1, y+1, z, c)+
get_pixel_value(x, y+1, z, c)+get_pixel_value(x+1, y+1, z, c)+get_pixel_value(x+1, y, z, c))+5*(get_pixel_value(x+1, y-
1, z, c)+get_pixel_value(x, y-1, z, c)+get_pixel_value(x-1, y-1, z, c));
01314                     if (sum > 255 || sum < -255)
01315                     {
01316                         sum = 255;
01317                     }
01318                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01319                     filtered.set_pixel_value(x, y, z, c, pixel);
01320                 }
01321             }
01322         }
01323     }
01324 }

```

```

01325     }
01326
01327     return filtered;
01328 }
01329 }
01330
01335 Image Image ::filter_kirsch_135()
01336 {
01337     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01338 );
01339     int m = 1;
01340     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01341     {
01342         for(unsigned int z = 0; z < this->get_depth(); z++)
01343         {
01344             for(unsigned int x = m; x < this->get_width()-m; x++)
01345             {
01346                 for(unsigned int y = m; y < this->get_height()-m; y++)
01347                 {
01348                     int sum = -3*(get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x+1, y, z, c)+
get_pixel_value(x+1, y+1, z, c)+get_pixel_value(x, y+1, z, c)+get_pixel_value(x-1, y+1, z, c))+5*(get_pixel_value(x-1,
y, z, c)+get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x, y-1, z, c));
01349                     if (sum > 255 || sum < -255)
01350                     {
01351                         sum = 255;
01352                     }
01353                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01354                     filtered.set_pixel_value(x, y, z, c, pixel);
01355                 }
01356             }
01357         }
01358     }
01359 }
01360 }
01361
01362     return filtered;
01363 }
01364 }
01365
01370 Image Image ::filter_kirsch_180()
01371 {
01372     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01373 );
01374     int m = 1;
01375     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01376     {
01377         for(unsigned int z = 0; z < this->get_depth(); z++)
01378         {
01379             for(unsigned int x = m; x < this->get_width()-m; x++)
01380             {
01381                 for(unsigned int y = m; y < this->get_height()-m; y++)
01382                 {
01383                     int sum = -3*(get_pixel_value(x, y-1, z, c)+get_pixel_value(x+1, y-1, z, c)+
get_pixel_value(x+1, y, z, c)+get_pixel_value(x+1, y+1, z, c)+get_pixel_value(x, y+1, z, c))+5*(get_pixel_value(x-1, y,
z, c)+get_pixel_value(x-1, y+1, z, c)+get_pixel_value(x-1, y-1, z, c));
01384                     if (sum > 255 || sum < -255)
01385                     {
01386                         sum = 255;
01387                     }
01388                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01389                     filtered.set_pixel_value(x, y, z, c, pixel);
01390                 }
01391             }
01392         }
01393     }
01394 }
01395 }
01396
01397     return filtered;
01398 }
01399 }
01400
01405 Image Image ::filter_kirsch_225()
01406 {
01407     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01408 );
01409     int m = 1;
01410     for(unsigned int c = 0; c < this->get_spectrum(); c++)

```

```

01411     {
01412         for(unsigned int z = 0; z < this->get_depth(); z++)
01413         {
01414             for(unsigned int x = m; x < this->get_width()-m; x++)
01415             {
01416                 for(unsigned int y = m; y < this->get_height()-m; y++)
01417                 {
01418                     int sum = -3*(get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x, y-1, z, c)+
get_pixel_value(x+1, y-1, z, c)+get_pixel_value(x+1, y, z, c)+get_pixel_value(x+1, y+1, z, c))+5*(get_pixel_value(x-1,
y, z, c)+get_pixel_value(x-1, y+1, z, c)+get_pixel_value(x, y+1, z, c));
01419                     if (sum > 255 || sum < -255)
01420                     {
01421                         sum = 255;
01422                     }
01423                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01424                     filtered.set_pixel_value(x, y, z, c, pixel);
01425                 }
01426             }
01427         }
01428     }
01429 }
01430 }
01431
01432     return filtered;
01433 }
01434 }
01435
01440 Image Image ::filter_kirsch_270()
01441 {
01442     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
01443     int m = 1;
01444
01445     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01446     {
01447         for(unsigned int z = 0; z < this->get_depth(); z++)
01448         {
01449             for(unsigned int x = m; x < this->get_width()-m; x++)
01450             {
01451                 for(unsigned int y = m; y < this->get_height()-m; y++)
01452                 {
01453                     int sum = -3*(get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x-1, y, z, c)+
get_pixel_value(x, y-1, z, c)+get_pixel_value(x+1, y-1, z, c)+get_pixel_value(x+1, y, z, c))+5*(get_pixel_value(x, y+1,
z, c)+get_pixel_value(x+1, y+1, z, c)+get_pixel_value(x-1, y+1, z, c));
01454                     if (sum > 255 || sum < -255)
01455                     {
01456                         sum = 255;
01457                     }
01458                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01459                     filtered.set_pixel_value(x, y, z, c, pixel);
01460                 }
01461             }
01462         }
01463     }
01464 }
01465 }
01466
01467     return filtered;
01468 }
01469 }
01470
01475 Image Image ::filter_kirsch_315()
01476 {
01477     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
01478     int m = 1;
01479
01480     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01481     {
01482         for(unsigned int z = 0; z < this->get_depth(); z++)
01483         {
01484             for(unsigned int x = m; x < this->get_width()-m; x++)
01485             {
01486                 for(unsigned int y = m; y < this->get_height()-m; y++)
01487                 {
01488                     int sum = -3*(get_pixel_value(x-1, y+1, z, c)+get_pixel_value(x-1, y, z, c)+
get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x, y-1, z, c)+get_pixel_value(x+1, y-1, z, c))+5*(get_pixel_value(x, y+
1, z, c)+get_pixel_value(x+1, y+1, z, c)+get_pixel_value(x+1, y, z, c));
01489                     if (sum > 255 || sum < -255)
01490                     {
01491                         sum = 255;

```

```

01492         }
01493         unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01494         filtered.set_pixel_value(x, y, z, c, pixel);
01495     }
01496 }
01497 }
01498 }
01499 }
01500 }
01501 }
01502     return filtered;
01503 }
01504 }
01505
01510 Image Image ::filter_freeman_0()
01511 {
01512     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01513 );
01514     int m = 1;
01515     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01516     {
01517         for(unsigned int z = 0; z < this->get_depth(); z++)
01518         {
01519             for(unsigned int x = m; x < this->get_width()-m; x++)
01520             {
01521                 for(unsigned int y = m; y < this->get_height()-m; y++)
01522                 {
01523                     int sum = (get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x-1, y, z, c)+
01524 get_pixel_value(x-1, y+1, z, c)+get_pixel_value(x, y-1, z, c)+get_pixel_value(x+1, y-1, z, c)+get_pixel_value(x+1,y,z,c))-
01525 1*(get_pixel_value(x+1, y+1, z, c)+get_pixel_value(x, y+1, z, c))+2*get_pixel_value(x, y, z, c);
01526                     if (sum > 255 || sum < -255)
01527                     {
01528                         sum = 255;
01529                     }
01529                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01530                     filtered.set_pixel_value(x, y, z, c, pixel);
01531                 }
01532             }
01533         }
01534     }
01535 }
01536 }
01537 }
01538     return filtered;
01539 }
01540 }
01541
01545 Image Image ::filter_freeman_1()
01546 {
01547     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01548 );
01549     int m = 1;
01550     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01551     {
01552         for(unsigned int z = 0; z < this->get_depth(); z++)
01553         {
01554             for(unsigned int x = m; x < this->get_width()-m; x++)
01555             {
01556                 for(unsigned int y = m; y < this->get_height()-m; y++)
01557                 {
01558                     int sum = (get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x+1, y+1, z, c)+
01559 get_pixel_value(x-1, y+1, z, c)+get_pixel_value(x, y-1, z, c)+get_pixel_value(x+1, y-1, z, c)+get_pixel_value(x+1,y,z,c)
01560 )-1*(get_pixel_value(x-1, y, z, c)+get_pixel_value(x, y+1, z, c))+2*get_pixel_value(x, y, z, c);
01561                     if (sum > 255 || sum < -255)
01562                     {
01563                         sum = 255;
01564                     }
01564                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01565                     filtered.set_pixel_value(x, y, z, c, pixel);
01566                 }
01567             }
01568         }
01569     }
01570 }
01571 }
01572 }
01573     return filtered;

```

```

01574
01575 }
01576
01580 Image Image ::filter_freeman_2()
01581 {
01582     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01583 );
01584     int m = 1;
01585     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01586     {
01587         for(unsigned int z = 0; z < this->get_depth(); z++)
01588         {
01589             for(unsigned int x = m; x < this->get_width()-m; x++)
01590             {
01591                 for(unsigned int y = m; y < this->get_height()-m; y++)
01592                 {
01593                     int sum = (get_pixel_value(x, y-1, z, c)+get_pixel_value(x+1, y-1, z, c)+
01594 get_pixel_value(x+1, y, z, c)+get_pixel_value(x+1, y+1, z, c)+get_pixel_value(x, y+1, z, c)+get_pixel_value(x-1,y+1,z,c))-
01595 1*(get_pixel_value(x-1, y, z, c)+get_pixel_value(x-1, y-1, z, c))+2*get_pixel_value(x, y, z, c);
01596                     if (sum > 255 || sum < -255)
01597                     {
01598                         sum = 255;
01599                     }
01600                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01601                     filtered.set_pixel_value(x, y, z, c, pixel);
01602                 }
01603             }
01604         }
01605     }
01606 }
01607
01608 return filtered;
01609
01610 }
01611
01615 Image Image ::filter_freeman_3()
01616 {
01617     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01618 );
01619     int m = 1;
01620     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01621     {
01622         for(unsigned int z = 0; z < this->get_depth(); z++)
01623         {
01624             for(unsigned int x = m; x < this->get_width()-m; x++)
01625             {
01626                 for(unsigned int y = m; y < this->get_height()-m; y++)
01627                 {
01628                     int sum = (get_pixel_value(x+1, y-1, z, c)+get_pixel_value(x+1, y, z, c)+
01629 get_pixel_value(x+1, y+1, z, c)+get_pixel_value(x, y+1, z, c)+get_pixel_value(x-1,y+1,z,c))-1*(get_pixel_value(x-1, y, z,
01630 c)+get_pixel_value(x, y-1, z, c)+get_pixel_value(x-1,y-1,z,c))+2*get_pixel_value(x, y, z, c);
01631                     if (sum > 255 || sum < -255)
01632                     {
01633                         sum = 255;
01634                     }
01635                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01636                     filtered.set_pixel_value(x, y, z, c, pixel);
01637                 }
01638             }
01639         }
01640     }
01641 }
01642
01643 return filtered;
01644
01645 }
01646
01650 Image Image ::filter_freeman_4()
01651 {
01652     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01653 );
01654     int m = 1;
01655     for(unsigned int c = 0; c < this->get_spectrum(); c++)

```



```

01657     {
01658         for(unsigned int z = 0; z < this->get_depth(); z++)
01659         {
01660             for(unsigned int x = m; x < this->get_width()-m; x++)
01661             {
01662                 for(unsigned int y = m; y < this->get_height()-m; y++)
01663                 {
01664                     int sum = (get_pixel_value(x-1, y, z, c)+get_pixel_value(x-1, y+1, z, c)+
get_pixel_value(x, y+1, z, c)+get_pixel_value(x+1, y+1, z, c)+get_pixel_value(x+1, y, z, c))-1*(get_pixel_value(x-1, y-1,
z, c)+get_pixel_value(x, y-1, z, c)+get_pixel_value(x+1,y-1,z,c)))-2*get_pixel_value(x, y, z, c);
01665                     if (sum > 255 || sum < -255)
01666                     {
01667                         sum = 255;
01668                     }
01669                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01670                     filtered.set_pixel_value(x, y, z, c, pixel);
01671                 }
01672             }
01673         }
01674     }
01675 }
01676 }
01677
01678 return filtered;
01679
01680 }
01681
01682
01686 Image Image ::filter_freeman_5()
01687 {
01688     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
01689
01690     int m = 1;
01691
01692     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01693     {
01694         for(unsigned int z = 0; z < this->get_depth(); z++)
01695         {
01696             for(unsigned int x = m; x < this->get_width()-m; x++)
01697             {
01698                 for(unsigned int y = m; y < this->get_height()-m; y++)
01699                 {
01700                     int sum = (get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x-1, y, z, c)+
get_pixel_value(x-1, y+1, z, c)+get_pixel_value(x, y+1, z, c)+get_pixel_value(x+1, y+1, z, c))-1*(get_pixel_value(x+1, y,
z, c)+get_pixel_value(x, y-1, z, c)+get_pixel_value(x+1,y-1,z,c)))-2*get_pixel_value(x, y, z, c);
01701                     if (sum > 255 || sum < -255)
01702                     {
01703                         sum = 255;
01704                     }
01705                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01706                     filtered.set_pixel_value(x, y, z, c, pixel);
01707                 }
01708             }
01709         }
01710     }
01711 }
01712 }
01713
01714 return filtered;
01715
01716 }
01717
01718
01722 Image Image ::filter_freeman_6()
01723 {
01724     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
01725
01726     int m = 1;
01727
01728     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01729     {
01730         for(unsigned int z = 0; z < this->get_depth(); z++)
01731         {
01732             for(unsigned int x = m; x < this->get_width()-m; x++)
01733             {
01734                 for(unsigned int y = m; y < this->get_height()-m; y++)
01735                 {
01736                     int sum = (get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x-1, y, z, c)+
get_pixel_value(x-1, y+1, z, c)+get_pixel_value(x, y+1, z, c)+get_pixel_value(x, y-1, z, c))-1*(get_pixel_value(x+1, y-1,

```

```

z, c)+get_pixel_value(x+1, y, z, c)+get_pixel_value(x+1,y+1,z,c))+-2*get_pixel_value(x, y, z, c);
01737         if (sum > 255 || sum < -255)
01738         {
01739             sum = 255;
01740         }
01741         unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01742         filtered.set_pixel_value(x, y, z, c, pixel);
01743     }
01744 }
01745 }
01746 }
01747 }
01748 }
01749 }
01750 return filtered;
01751 }
01752 }
01753 }
01757 Image Image ::filter_freeman_7()
01758 {
01759     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
01760
01761     int m = 1;
01762
01763     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01764     {
01765         for(unsigned int z = 0; z < this->get_depth(); z++)
01766         {
01767             for(unsigned int x = m; x < this->get_width()-m; x++)
01768             {
01769                 for(unsigned int y = m; y < this->get_height()-m; y++)
01770                 {
01771                     int sum = (get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x-1, y, z, c)+
get_pixel_value(x-1, y+1, z, c)+get_pixel_value(x+1, y-1, z, c)+get_pixel_value(x, y-1, z, c))-1*(get_pixel_value(x, y+1,
z, c)+get_pixel_value(x+1, y, z, c)+get_pixel_value(x+1,y+1,z,c))+-2*get_pixel_value(x, y, z, c);
01772                     if (sum > 255 || sum < -255)
01773                     {
01774                         sum = 255;
01775                     }
01776                     unsigned char pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01777                     filtered.set_pixel_value(x, y, z, c, pixel);
01778                 }
01779             }
01780         }
01781     }
01782 }
01783 }
01784 }
01785 return filtered;
01786 }
01787 }
01788 }
01795 Image Image :: filter_maximum()
01796 {
01797     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
01798
01799     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01800     {
01801         for(unsigned int z = 0; z < this->get_depth(); z++)
01802         {
01803             for(unsigned int x = 1; x < this->get_width()-1; x++)
01804             {
01805                 for(unsigned int y = 1; y < this->get_height()-1; y++)
01806                 {
01807                     unsigned char max = 0;
01808
01809                     for (unsigned int i = x-1; i < x+2; i++)
01810                     {
01811                         for (unsigned int j = y-1; j < y+2; j++)
01812                         {
01813                             unsigned char pixel = (this->get_pixel_value(i, j, z, c));
01814
01815                             if (pixel > max)
01816                             {
01817                                 max = this->get_pixel_value(i, j, z, c);
01818                             }
01819                         }
01820                     }
01821                 }
01822             }
01823         }
01824     }
01825 }
01826 }
01827 }

```

```

01822         filtered.set_pixel_value(x, y, z, c, max);
01823     }
01824
01825     }
01826
01827     }
01828 }
01829 return filtered;
01830 }
01831
01832
01838 Image Image :: filter_minimum()
01839 {
01840     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01841 );
01842     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01843     {
01844         for(unsigned int z = 0; z < this->get_depth(); z++)
01845         {
01846             for(unsigned int x = 1; x < this->get_width()-1; x++)
01847             {
01848                 for(unsigned int y = 1; y < this->get_height()-1; y++)
01849                 {
01850                     unsigned char minimum = 255;
01851
01852                     for (unsigned int i = x-1; i < x+2; i++)
01853                     {
01854                         for (unsigned int j = y-1; j < y+2; j++)
01855                         {
01856                             if ((this->get_pixel_value(i, j, z, c)) < minimum)
01857                             {
01858                                 minimum = this->get_pixel_value(i, j, z, c);
01859                             }
01860                         }
01861                     }
01862                     filtered.set_pixel_value(x, y, z, c, minimum);
01863                 }
01864             }
01865         }
01866     }
01867
01868 }
01869 }
01870 return filtered;
01871 }
01872
01873 Image Image :: filter_order_statistics(int dim, int order)
01874 {
01875     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01876 );
01877     //int kernel [dim*dim];
01878
01879     int m = (dim-1)/2;
01880
01881     unsigned char pixel_values [dim*dim];
01882     unsigned char temp;
01883
01884     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01885     {
01886         for(unsigned int z = 0; z < this->get_depth(); z++)
01887         {
01888             for(unsigned int x = m; x < this->get_width(); x++)
01889             {
01890                 for(unsigned int y = m; y < this->get_height(); y++)
01891                 {
01892                     for(unsigned int i = x-m; i < x+m+1; i++)
01893                     {
01894                         for(unsigned int j = y-m; j < y+m+1; j++)
01895                         {
01896                             pixel_values [(i-x+m)*dim + (j-y+m)] = this->get_pixel_value(i, j, z, c);
01897                         }
01898                     }
01899                 }
01900                 for(int k=0; k<dim*dim ; k++)
01901                 {
01902                     for(int p=k+1 ; p<dim*dim ; p++)
01903                     {
01904                         if(pixel_values[p] < pixel_values[k])
01905                         {

```

```

01906             // Intercambiar los valores
01907             temp = pixel_values[k];
01908             pixel_values[k] = pixel_values[p];
01909             pixel_values[p] = temp;
01910         }
01911     }
01912 }
01913 unsigned char pixel = pixel_values[order];
01914 filtered.set_pixel_value(x, y, z, c, pixel);
01915 }
01916 }
01917 }
01918 }
01919 }
01920 }
01921 return filtered;
01922 }
01923
01924 // *****
01925 // ***** NOISES *****
01926 // *****
01927 // *****
01928
01933 void Image :: salt_pepper(double intensity)
01934 {
01935     srand(1);
01936     double percentage = 1-(intensity/100);
01937     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01938     {
01939         for(unsigned int z = 0; z < this->get_depth(); z++)
01940         {
01941             for(unsigned int x = 0; x < this->get_width(); x++)
01942             {
01943                 for(unsigned int y = 0; y < this->get_height(); y++)
01944                 {
01945                     double random= 2.0*(rand()-RAND_MAX/2.0)/RAND_MAX;
01946                     if(random > percentage)
01947                     {
01948                         (*(this->Img))(x, y, z, c)= 255;
01949                     }
01950
01951                     else if(random<-1*percentage)
01952                     {
01953                         (*(this->Img))(x, y, z, c)= 0;
01954                     }
01955                 }
01956             }
01957         }
01958     }
01959 }
01960 }
01961 }
01962
01963 }
01964
01970 void Image :: gaussian_noise(double variance)
01971 {
01972     srand(1);
01973     for(unsigned int c = 0; c < this->get_spectrum(); c++)
01974     {
01975         for(unsigned int z = 0; z < this->get_depth(); z++)
01976         {
01977             for(unsigned int x = 0; x < this->get_width(); x++)
01978             {
01979                 for(unsigned int y = 0; y < this->get_height(); y++)
01980                 {
01981                     double random= variance*(rand()-RAND_MAX/2.0)/RAND_MAX;
01982                     unsigned char pixel= this->get_pixel_value(x,y,z,c) + random;
01983
01984                     if((pixel<255) & (pixel>0))
01985                     {
01986                         (*(this->Img))(x, y, z, c)= pixel;
01987                     }
01988                 }
01989             }
01990         }
01991     }
01992 }
01993
01994 }
01995 }

```

```

01996
01997 }
01998
01999
02005 Image Image :: interpolation()
02006 {
02007     int i,j=0;
02008     Image result (2*this->get_width(),2*this->get_height(),this->get_depth(),this->get_spectrum(),0);
02009     for(unsigned int c = 0; c < this->get_spectrum(); c++)
02010     {
02011         for(unsigned int z = 0; z < this->get_depth(); z++)
02012         {
02013             for(unsigned int y = 0; y < this->get_width(); y++)
02014             {
02015                 for(unsigned int x = 0; x < this->get_height(); x++)
02016                 {
02017                     unsigned char pixel=this->get_pixel_value(x,y,z,c);
02018
02019                     result.set_pixel_value(x+i,y+j,z,c,pixel);
02020                     result.set_pixel_value(x+1+i,y+j,z,c,pixel);
02021                     result.set_pixel_value(x+i,y+1+j,z,c,pixel);
02022                     result.set_pixel_value(x+1+i,y+1+j,z,c,pixel);
02023                     i++;
02024                 }
02025                 i=0;
02026                 j++;
02027             }
02028             j=0;
02029         }
02030     }
02031     return result;
02032 }
02033
02047 CImg<float> Image:: autocovariance (int hor_dis, int ver_dis)
02048 {
02049     CImg<float> autocovariance (this->get_width() , this->get_height(), this->get_depth(),
this->get_spectrum(), 0);
02050
02051     Image average = this->filter_average(1);
02052
02053     for(unsigned int c = 0; c < this->get_spectrum(); c++)
02054     {
02055         for(unsigned int z = 0; z < this->get_depth(); z++)
02056         {
02057             for(unsigned int x = 3+hor_dis; x < this->get_width()-(3+hor_dis); x++)
02058             {
02059                 for(unsigned int y = 3+ver_dis; y < this->get_height()-(3+ver_dis); y++)
02060                 {
02061                     int sum = 0;
02062                     for(unsigned int i = x-3; i<x+4; i++)
02063                     {
02064                         for(unsigned int j = y-3; j<y+4; j++)
02065                         {
02066                             sum += ( (this->get_pixel_value(i,j,z,c)) - average.get_pixel_value(i,j,z,c))
* ( (this->get_pixel_value(i+hor_dis,j+ver_dis,z,c)) - average.get_pixel_value(i+hor_dis,j+ver_dis,z,c))
;
02067                         }
02068                     }
02069
02070                     autocovariance(x,y,z,c) = sum/49;
02071                 }
02072             }
02073         }
02074     }
02075     return autocovariance;
02076 }
02077
02084 Image Image :: variance(int dim)
02085 {
02086     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
02087
02088     for(unsigned int c = 0; c < this->get_spectrum(); c++)
02089     {
02090         for(unsigned int z = 0; z < this->get_depth(); z++)
02091         {
02092             for(unsigned int x = dim; x < this->get_width()-dim; x++)
02093             {
02094                 for(unsigned int y = dim; y < this->get_height()-dim; y++)
02095                 {
02096                     int sum = 0;

```

```

02097         double variance=0;
02098         int kernel_values[(dim*2+1)*(dim*2+1)];
02099         int cont=0;
02100
02101         for(unsigned int i = x-dim; i<= x+dim; i++)
02102         {
02103             for(unsigned int j = y-dim; j<= y+dim; j++)
02104             {
02105                 sum += this->get_pixel_value(i, j, z, c);
02106                 kernel_values[cont]=this->get_pixel_value(i, j, z, c);
02107                 cont++;
02108             }
02109         }
02110
02111         double average = sum/((dim*2+1)*(dim*2+1));
02112         for(int i=0;i<(dim*2+1)*(dim*2+1);i++)
02113         {
02114             variance+=pow(kernel_values[i]-average,2)/((dim*2+1)*(dim*2+1));
02115         }
02116
02117         unsigned char pixel = (unsigned char)static_cast<unsigned char> (variance);
02118         filtered.set_pixel_value(x, y, z, c, pixel);
02119     }
02120 }
02121 }
02122 }
02123 }
02124 }
02125 }
02126     return filtered;
02127 }
02128
02136 Image Image :: gray_scale()
02137 {
02138     Image gray_image (this->get_width() , this->get_height(), this->get_depth(), 1, 0);
02139
02140     for(unsigned int z = 0; z < this->get_depth(); z++)
02141     {
02142         for(unsigned int x = 0; x < this->get_width(); x++)
02143         {
02144             for(unsigned int y = 0; y < this->get_height(); y++)
02145             {
02146                 unsigned char pixel_intensity = 0.56*this->get_pixel_value(x,y,z,1)+0.14*this->
02147                 get_pixel_value(x,y,z,0)+0.11*this->get_pixel_value(x,y,z,2);
02148                 gray_image.set_pixel_value(x, y, z, 0, pixel_intensity);
02149             }
02150         }
02151     }
02152 }
02153 }
02154 }
02155     return gray_image;
02156 }
02157
02164 Image Image :: coorrelogram(unsigned int ver,unsigned int hor)
02165 {
02166     Image result (256,256, 1, 1, 0);
02167
02168     for(unsigned int i = 0; i < 256; i++)
02169     {
02170         for(unsigned int j=0; j< 256; j++)
02171         {
02172             unsigned int pixel = 0;
02173
02174             for(unsigned int x=0; x< (this->get_width()-hor);++x)
02175             {
02176                 for(unsigned int y=0; y< (this->get_height()-ver);++y)
02177                 {
02178                     unsigned char first = (this->get_pixel_value(x,y,0,0));
02179                     unsigned char secnd = (this->get_pixel_value(x+hor, y+ver, 0, 0));
02180
02181                     if(first == i && secnd == j)
02182                     {
02183                         pixel ++;
02184                     }
02185                 }
02186             }
02187         }
02188     }
02189     if(pixel>255)

```

```

02190         {
02191             pixel=255;
02192         }
02193
02194         result.set_pixel_value(i, j, 0, 0, pixel);
02195
02196     }
02197
02198     cout<<"\n"<<i<<"\n"<<endl;
02199 }
02200 return result;
02201
02202 }
02203
02212 Image Image::coorrelogram_ZC(unsigned int ver,unsigned int hor, unsigned int
z, unsigned int c)
02213 {
02214
02215     Image result (256,256, 1, 1, 0);
02216
02217     for(unsigned int i = 0; i < 256; i++)
02218     {
02219         for(unsigned int j=0; j< 256; j++)
02220         {
02221             unsigned int pixel = 0;
02222
02223             for(unsigned int x=0; x< (this->get_width()-hor);++x)
02224             {
02225
02226                 for(unsigned int y=0; y< (this->get_height()-ver);++y)
02227                 {
02228                     unsigned char first = (this->get_pixel_value(x,y,z,c));
02229                     unsigned char secnd = (this->get_pixel_value(x+hor, y+ver, z, c));
02230                     if(first == i && secnd == j)
02231                     {
02232                         pixel ++;
02233                     }
02234                 }
02235             }
02236             if(pixel>255)
02237             {
02238                 pixel=255;
02239             }
02240
02241             result.set_pixel_value(i, j, 0, 0, pixel);
02242
02243         }
02244
02245     }
02246     return result;
02247
02248 }
02249
02250
02251
02252

```

## 4.5 /home/fish/Documents/ParallelPic/Proyecto/src/ParallelPic.cpp File Reference

```
#include "../include/ParallelPic.hh"
```

## 4.6 ParallelPic.cpp

```

00001 #include "../include/ParallelPic.hh"
00002
00003
00008 // *****
00009 // ***** CONTRUCTORS *****
00010 // *****
00011
00017 Image::Image()

```

```

00018 {
00019     this->Img = new CImg<unsigned char>();
00020     this->width = 0;
00021     this->height = 0;
00022     this->depth = 0;
00023     this->spectrum = 0;
00024 }
00025 }
00032 Image::Image(const char *const filename)
00033 {
00034     this->Img = new CImg<unsigned char>(filename);
00036     this->width = this->Img->width();
00038     this->height = this->Img->height();
00040     this->depth = this->Img->depth();
00042     this->spectrum = this->Img->spectrum();
00044 }
00049 Image::Image(const unsigned int width, const unsigned int height, const unsigned int depth,
const unsigned int spectrum, int value)
00050 {
00051     this->Img = new CImg<unsigned char>(width, height, depth, spectrum, value);
00052     this->width = width;
00053     this->height = height;
00054     this->depth = depth;
00055     this->spectrum = spectrum;
00056 }
00060 Image::~Image(void)
00061 {
00062 }
00063 }
00064 // *****
00065 // ***** SAVE & DISPLAY *****
00066 // *****
00072 void Image:: save(const char *const savefilename)
00073 {
00074     this->Img->save(savefilename);
00075 }
00076 }
00077 void Image :: display(const char* message)
00078 {
00079     CImgDisplay main (*(this->Img), message);
00080     while(!main.is_closed())
00081     {
00082         main.wait();
00083     }
00084 }
00085 }
00086 // *****
00087 // ***** GETs & SETs *****
00088 // *****
00089 }
00094 unsigned int Image:: get_width()
00095 {
00096     return this->width;
00097 }
00102 unsigned int Image:: get_height()
00103 {
00104     return this->height;
00105 }
00106 }
00111 unsigned int Image:: get_depth()
00112 {
00113     return this->depth;
00114 }
00115 }
00120 unsigned int Image:: get_spectrum()
00121 {
00122     return this->spectrum;
00123 }
00124 }
00125 }
00130 unsigned int Image:: get_pixel_value(int x, int y, int z, int c)
00131 {
00132     return this->Img->get_vector_at(x, y, z)[c];
00133 }
00134 }
00139 void Image:: set_pixel_value(int x, int y, int z, int c, unsigned char value)
00140 {
00141     (*(this->Img))(x, y, z, c)= value;
00142 }
00143 }
00144 }

```



```

00145 // *****
00146 // *****Arithmetic and Logic *****
00147 // *****
00148
00154 Image Image :: subtract_img(Image image2, int num_threads)
00155 {
00156     unsigned int x,y,c,z;
00157     unsigned char pixel;
00158     int chunk= (this->get_width()/num_threads);
00159     omp_set_num_threads(num_threads);
00160
00161     Image result (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0);
00162
00163     if(this->get_width() == image2.get_width() && this->get_height() == image2.get_height() && this->
get_depth() == image2.get_depth() && this->get_spectrum() == image2.get_spectrum())
00164     {
00165         for(c = 0; c < this->get_spectrum(); c++)
00166         {
00167             for(z = 0; z < this->get_depth(); z++)
00168             {
00169                 #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,x,y)
shared(z,c,result,chunk)
00170                 for(x = 0; x < this->get_width(); x++)
00171                 {
00172                     for(y = 0; y < this->get_height(); y++)
00173                     {
00174                         pixel= static_cast<unsigned int>(abs(this->get_pixel_value(x,y,z,c)-image2.
get_pixel_value(x,y,z,c)));
00175
00176                         #pragma omp ordered
00177                         result.set_pixel_value(x,y,z,c,pixel);
00178                     }
00179                 }
00180             }
00181         }
00182     }
00183     return result;
00184 }
00185
00186
00187 Image Image :: sum_img(Image image2, int num_threads)
00188 {
00189     unsigned int x,y,c,z,sum;
00190     unsigned char pixel;
00191     int chunk= (this->get_width()/num_threads);
00192     omp_set_num_threads(num_threads);
00193
00194     Image result (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0);
00195
00196     if(this->get_width() == image2.get_width() && this->get_height() == image2.get_height() && this->
get_depth() == image2.get_depth() && this->get_spectrum() == image2.get_spectrum())
00197     {
00198         for(c = 0; c < this->get_spectrum(); c++)
00199         {
00200             for(z = 0; z < this->get_depth(); z++)
00201             {
00202                 #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,sum)
shared(z,c,result,chunk)
00203                 for(x = 0; x < this->get_width(); x++)
00204                 {
00205                     for(y = 0; y < this->get_height(); y++)
00206                     {
00207                         sum = this->get_pixel_value(x,y,z,c)+image2.get_pixel_value(x,y,z,c);
00208                         if (sum <= 255)
00209                         {
00210                             pixel = static_cast<unsigned int>(sum);
00211                         }
00212                         else
00213                         {
00214                             pixel = 255;
00215                         }
00216                         #pragma omp ordered
00217                         result.set_pixel_value(x,y,z,c,pixel);
00218                     }
00219                 }
00220             }
00221         }
00222     }
00223 }

```

```

00224     return result;
00225 }
00226
00227
00234 Image Image :: multiply_img(double multiplier,int num_threads)
00235 {
00236     unsigned int x,y,c,z;
00237     unsigned char pixel;
00238     int chunk= (this->get_width()/num_threads);
00239     omp_set_num_threads(num_threads);
00240
00241     Image result (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0);
00242     for(c = 0; c < this->get_spectrum(); c++)
00243     {
00244         for(z = 0; z < this->get_depth(); z++)
00245         {
00246             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel)
00247             shared(z,c,result,chunk)
00248             for(x = 0; x < this->get_width(); x++)
00249             {
00250                 for(y = 0; y < this->get_height(); y++)
00251                 {
00252                     pixel= static_cast<unsigned int>(abs(this->get_pixel_value(x,y,z,c)*multiplier));
00253                     if (pixel >255)
00254                         pixel = 255;
00255                     #pragma omp parallel
00256                     result.set_pixel_value(x,y,z,c,pixel);
00257                 }
00258             }
00259         }
00260     }
00261     return result;
00262 }
00269 Image Image :: binarize_img(unsigned int cutoff_value , int num_threads)
00270 {
00271     unsigned int x,y,c,z;
00272     unsigned char pixel;
00273     int chunk= (this->get_width()/num_threads);
00274     omp_set_num_threads(num_threads);
00275
00276     Image result (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0);
00277     for(c = 0; c < this->get_spectrum(); c++)
00278     {
00279         for(z = 0; z < this->get_depth(); z++)
00280         {
00281             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel)
00282             shared(z,c,result,chunk)
00283             for(x = 0; x < this->get_width(); x++)
00284             {
00285                 for(y = 0; y < this->get_height(); y++)
00286                 {
00287                     pixel= static_cast<unsigned int>(this->get_pixel_value(x,y,z,c));
00288                     if(pixel >= cutoff_value)
00289                         pixel=255;
00290                     else
00291                         pixel=0;
00292                     #pragma omp ordered
00293                     result.set_pixel_value(x,y,z,c,pixel);
00294                 }
00295             }
00296         }
00297     }
00298 }
00299
00300
00301     return result;
00302 }
00303
00304
00305 // *****
00306 // ***** SPACE DOMAIN FILTERS *****
00307 // *****
00308
00309
00310
00311
00312 // *****
00313 // ***** Sharpening Spatial Filters *****
00314 // *****

```

```

00315
00323 Image Image::filter_Laplacian(int num_threads)
00324 {
00325     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
00326
00327     int m = 1;
00328     unsigned int x,y,c,z;
00329     int sum;
00330     unsigned char pixel;
00331     int chunk= (this->get_width()/num_threads);
00332     omp_set_num_threads(num_threads);
00333
00334     for(c = 0; c < this->get_spectrum(); c++)
00335     {
00336         for(z = 0; z < this->get_depth(); z++)
00337         {
00338             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,sum)
shared(z,c,filtered,chunk,m)
00339             for(x = m; x < this->get_width()-m; x++)
00340             {
00341                 for(y = m; y < this->get_height()-m; y++)
00342                 {
00343                     sum = 0;
00344
00345                     for (unsigned int i = 0 ; i < 3; i++)
00346                     {
00347                         for(unsigned int j = 0 ; j < 3; j++)
00348                         {
00349                             sum += -this->get_pixel_value(x+i-1, y+i-1, z, c);
00350                         }
00351                     }
00352
00353                     sum += 9*(this->get_pixel_value(x,y,z,c));
00354
00355                     if (sum > 255 || sum < -255)
00356                     {
00357                         sum = 255;
00358                     }
00359
00360                     pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
00361                     #pragma omp ordered
00362                     filtered.set_pixel_value(x, y, z, c, pixel);
00363                 }
00364             }
00365         }
00366     }
00367 }
00368
00369 return filtered;
00370 }
00371
00377 Image Image :: filter_Laplacian_no_diagonal(int num_threads)
00378 {
00379     unsigned int x,y,c,z;
00380     int sum;
00381     unsigned char pixel;
00382     int chunk= (this->get_width()/num_threads);
00383     omp_set_num_threads(num_threads);
00384
00385     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
00386
00387     int m = 1;
00388
00389     for(c = 0; c < this->get_spectrum(); c++)
00390     {
00391         for(z = 0; z < this->get_depth(); z++)
00392         {
00393             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,sum)
shared(z,c,filtered,chunk,m)
00394             for(x = m; x < this->get_width()-m; x++)
00395             {
00396                 for(y = m; y < this->get_height()-m; y++)
00397                 {
00398                     sum = 4*(this->get_pixel_value(x,y,z,c)) - (this->get_pixel_value(x-1,y,z,c) + this->
get_pixel_value(x+1,y,z,c) + this->get_pixel_value(x,y-1,z,c) +this->get_pixel_value(x,y+1,z,c));
00400
00401                     if (sum > 255 || sum < -255)
00402                     {

```

```

00403             sum = 255;
00404         }
00405         pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
00406
00407         #pragma omp ordered
00408         filtered.set_pixel_value(x, y, z, c, pixel);
00409     }
00410
00411     }
00412
00413     }
00414 }
00415 return filtered;
00416
00417 }
00418
00426 Image Image :: filter_Gradient_horizontal(int num_threads)
00427 {
00428     unsigned int x,y,c,z;
00429     int sum;
00430     unsigned char pixel;
00431     int chunk= (this->get_width()/num_threads);
00432     omp_set_num_threads(num_threads);
00433
00434     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00435 );
00436     int m = 1;
00437
00438     for(c = 0; c < this->get_spectrum(); c++)
00439     {
00440         for(z = 0; z < this->get_depth(); z++)
00441         {
00442             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,sum)
00443             shared(z,c,filtered,chunk,m)
00444
00445             for(x = m; x < this->get_width()-m; x++)
00446             {
00447                 for(y = m; y < this->get_height()-m; y++)
00448                 {
00449                     sum = this->get_pixel_value(x-1, y-1, z, c) + 2*(this->get_pixel_value(x, y-1, z, c)) +
00450                     this->get_pixel_value(x+1, y-1, z, c) - (this->get_pixel_value(x-1, y+1, z, c) + 2*(this->get_pixel_value(x
00451 , y+1, z, c)) + this->get_pixel_value(x+1, y+1, z, c));
00452                     if (sum > 255 || sum < -255)
00453                     {
00454                         sum = 255;
00455                     }
00456                     pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
00457                     #pragma omp ordered
00458                     filtered.set_pixel_value(x, y, z, c, pixel);
00459                 }
00460             }
00461         }
00462     }
00463     return filtered;
00464 }
00472 Image Image :: filter_Gradient_vertical(int num_threads)
00473 {
00474     unsigned int x,y,c,z;
00475     int sum;
00476     unsigned char pixel;
00477     int chunk= (this->get_width()/num_threads);
00478     omp_set_num_threads(num_threads);
00479     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00480 );
00481     int m = 1;
00482
00483     for(c = 0; c < this->get_spectrum(); c++)
00484     {
00485         for(z = 0; z < this->get_depth(); z++)
00486         {
00487             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,sum)
00488             shared(z,c,filtered,chunk,m)
00489
00490             for(x = m; x < this->get_width()-m; x++)
00491             {
00492                 for(y = m; y < this->get_height()-m; y++)

```

```

00492         {
00493             sum = get_pixel_value(x-1, y-1, z, c) + 2*get_pixel_value(x-1, y, z, c) +
get_pixel_value(x-1, y+1, z, c) - (get_pixel_value(x+1, y-1, z, c) + 2*get_pixel_value(x+1, y, z, c) + get_pixel_value(x+
1, y+1, z, c));
00494             if (sum > 255 || sum < -255)
00495             {
00496                 sum = 255;
00497             }
00498             pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
00499             #pragma omp ordered
00500             filtered.set_pixel_value(x, y, z, c, pixel);
00501         }
00502     }
00503 }
00504
00505 }
00506 }
00507     return filtered;
00508 }
00509
00527 Image Image :: filter_Prewitt_N_S(int num_threads)
00528 {
00529     unsigned int x,y,c,z;
00530     int sum;
00531     unsigned char pixel;
00532     int chunk= (this->get_width())/num_threads;
00533     omp_set_num_threads(num_threads);
00534
00535     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
00536
00537     int m = 1;
00538
00539     for(c = 0; c < this->get_spectrum(); c++)
00540     {
00541         for(z = 0; z < this->get_depth(); z++)
00542         {
00543             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,sum)
shared(z,c,filtered,chunk,m)
00544
00545             for(x = m; x < this->get_width()-m; x++)
00546             {
00547                 for(y = m; y < this->get_height()-m; y++)
00548                 {
00549                     sum = get_pixel_value(x-1, y-1, z, c) + get_pixel_value(x, y-1, z, c) + get_pixel_value
(x+1, y-1, z, c) - (get_pixel_value(x-1, y+1, z, c) + get_pixel_value(x, y+1, z, c) + get_pixel_value(x+1, y
+1, z, c));
00550                     if (sum > 255 || sum < -255)
00551                     {
00552                         sum = 255;
00553                     }
00554                     pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
00555                     #pragma omp ordered
00556                     filtered.set_pixel_value(x, y, z, c, pixel);
00557                 }
00558             }
00559         }
00560     }
00561 }
00562 }
00563
00564     return filtered;
00565 }
00566 }
00567
00568 Image Image ::filter_Prewitt_NE_SW(int num_threads)
00569 {
00570     unsigned int x,y,c,z;
00571     int sum;
00572     unsigned char pixel;
00573     int chunk= (this->get_width())/num_threads;
00574     omp_set_num_threads(num_threads);
00575
00576     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
00577
00578     int m = 1;
00579
00580     for(c = 0; c < this->get_spectrum(); c++)
00581     {
00582         for(z = 0; z < this->get_depth(); z++)

```

```

00583     {
00584         #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,sum)
00585         shared(z,c,filtered,chunk,m)
00586         for(x = m; x < this->get_width()-m; x++)
00587         {
00588             for(y = m; y < this->get_height()-m; y++)
00589             {
00590                 sum = get_pixel_value(x, y-1, z, c) + get_pixel_value(x+1, y-1, z, c) + get_pixel_value
(x+1, y, z, c) - (get_pixel_value(x-1, y, z, c) + get_pixel_value(x-1, y+1, z, c) + get_pixel_value(x, y+1,
z, c));
00591                 if (sum > 255 || sum < -255)
00592                 {
00593                     sum = 255;
00594                 }
00595                 pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
00596                 #pragma omp ordered
00597                 filtered.set_pixel_value(x, y, z, c, pixel);
00598             }
00599         }
00600     }
00601 }
00602 }
00603 }
00604
00605     return filtered;
00606 }
00607 }
00608
00609 Image Image ::filter_Prewitt_E_W(int num_threads)
00610 {
00611     unsigned int x,y,c,z;
00612     int sum;
00613     unsigned char pixel;
00614     int chunk= (this->get_width())/num_threads;
00615     omp_set_num_threads(num_threads);
00616
00617     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
00618
00619     int m = 1;
00620
00621     for(c = 0; c < this->get_spectrum(); c++)
00622     {
00623         for(z = 0; z < this->get_depth(); z++)
00624         {
00625             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,sum)
00626             shared(z,c,filtered,chunk,m)
00627             for(x = m; x < this->get_width()-m; x++)
00628             {
00629                 for(y = m; y < this->get_height()-m; y++)
00630                 {
00631                     sum = get_pixel_value(x-1, y-1, z, c) + get_pixel_value(x-1, y, z, c) + get_pixel_value
(x-1, y+1, z, c) - (get_pixel_value(x+1, y-1, z, c) + get_pixel_value(x+1, y, z, c) + get_pixel_value(x+1, y
+1, z, c));
00632                     if (sum > 255 || sum < -255)
00633                     {
00634                         sum = 255;
00635                     }
00636                     pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
00637                     #pragma omp ordered
00638                     filtered.set_pixel_value(x, y, z, c, pixel);
00639                 }
00640             }
00641         }
00642     }
00643 }
00644 }
00645
00646     return filtered;
00647 }
00648 }
00649
00650 Image Image ::filter_Prewitt_NW_SE(int num_threads)
00651 {
00652     unsigned int x,y,c,z;
00653     int sum;
00654     unsigned char pixel;
00655     int chunk= (this->get_width())/num_threads;
00656     omp_set_num_threads(num_threads);

```

```

00657
00658     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00659 );
00660     int m = 1;
00661     for(c = 0; c < this->get_spectrum(); c++)
00662     {
00663         for(z = 0; z < this->get_depth(); z++)
00664         {
00665             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,sum)
00666             shared(z,c,filtered,chunk,m)
00667             for(x = m; x < this->get_width()-m; x++)
00668             {
00669                 for(y = m; y < this->get_height()-m; y++)
00670                 {
00671                     sum = get_pixel_value(x-1, y-1, z, c) + get_pixel_value(x-1, y, z, c) + get_pixel_value
00672 (x, y-1, z, c) - (get_pixel_value(x+1, y, z, c) + get_pixel_value(x+1, y+1, z, c) + get_pixel_value(x, y+1,
z, c));
00673                     if (sum > 255 || sum < -255)
00674                     {
00675                         sum = 255;
00676                     }
00677                     pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
00678                     #pragma omp ordered
00679                     filtered.set_pixel_value(x, y, z, c, pixel);
00680                 }
00681             }
00682         }
00683     }
00684 }
00685 }
00686
00687     return filtered;
00688 }
00689 }
00690
00691 Image Image ::filter_edge_enhacement_displacement(unsigned
int horizontal_dis, unsigned int vertical_dis, int num_threads)
00692 {
00693     unsigned int x,y,c,z;
00694     unsigned char value;
00695     int chunk= (this->get_width()/num_threads);
00696     omp_set_num_threads(num_threads);
00697     Image result (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0);
00698
00699     if((horizontal_dis < this->get_width()) && (vertical_dis < this->get_height()))
00700     {
00701         for(c = 0; c < this->get_spectrum(); c++)
00702         {
00703             for(z = 0; z < this->get_depth(); z++)
00704             {
00705                 #pragma omp parallel for ordered schedule(dynamic,chunk) private(value)
00706                 shared(z,c,result,chunk)
00707                 for(x = horizontal_dis; x < this->get_width(); x++)
00708                 {
00709                     for(y = vertical_dis; y < this->get_height(); y++)
00710                     {
00711                         value = static_cast<unsigned char>(abs(this->get_pixel_value(x,y,z,c) - this->
get_pixel_value(x-horizontal_dis, y-vertical_dis, z, c)));
00712                         #pragma omp ordered
00713                         result.set_pixel_value(x,y,z,c, value);
00714                     }
00715                 }
00716             }
00717         }
00718     }
00719     return result;
00720 }
00721
00722 Image Image :: filter_vertical_borders(int num_threads)
00723 {
00724     unsigned int x,y,c,z;
00725     unsigned char pixel;
00726     int chunk= (this->get_width()/num_threads);
00727     omp_set_num_threads(num_threads);
00728     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);

```

```

00729
00730     int m = 1;
00731
00732     for(c = 0; c < this->get_spectrum(); c++)
00733     {
00734         for(z = 0; z < this->get_depth(); z++)
00735         {
00736             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel)
00737             shared(z,c,filtered,chunk,m)
00738             for(x = m; x < this->get_width()-m; x++)
00739             {
00740                 for(y = 0; y < this->get_height()-m; y++)
00741                 {
00742                     pixel = (unsigned char)static_cast<unsigned char> (abs(this->get_pixel_value(x-1, y, z,
00743 c) - get_pixel_value(x+1, y, z, c)));
00744                     filtered.set_pixel_value(x, y, z, c, pixel);
00745                 }
00746             }
00747         }
00748     }
00749     return filtered;
00750 }
00751 }
00752
00753 Image Image :: filter_horizontal_borders(int num_threads)
00754 {
00755     unsigned int x,y,c,z;
00756     unsigned char pixel;
00757     int chunk= (this->get_width()/num_threads);
00758     omp_set_num_threads(num_threads);
00759     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00760 );
00761
00762     int m = 1;
00763
00764     for(c = 0; c < this->get_spectrum(); c++)
00765     {
00766         for(z = 0; z < this->get_depth(); z++)
00767         {
00768             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel)
00769             shared(z,c,filtered,chunk,m)
00770             for(x = 0; x < this->get_width()-m; x++)
00771             {
00772                 for(y = m; y < this->get_height()-m; y++)
00773                 {
00774                     pixel = (unsigned char)static_cast<unsigned char> (abs(this->get_pixel_value(x, y-1, z,
00775 c) - get_pixel_value(x, y+1, z, c)));
00776                     #pragma omp ordered
00777                     filtered.set_pixel_value(x, y, z, c, pixel);
00778                 }
00779             }
00780         }
00781     }
00782     return filtered;
00783 }
00784 }
00785
00786 // *****
00787 // ***** Smoothing Spatial Filters *****
00788 // *****
00789
00790
00791 Image Image :: filter_median (int dim, int num_threads)
00792 {
00793     unsigned int x,y,c,z;
00794     int chunk= (this->get_width()/num_threads);
00795     omp_set_num_threads(num_threads);
00796     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00797 );
00798
00799     int m = (dim-1)/2;
00800     unsigned char pixel_values [dim*dim];
00801     unsigned char temp,pixel;
00802
00803     for(c = 0; c < this->get_spectrum(); c++)
00804     {

```



```

00810         for(z = 0; z < this->get_depth(); z++)
00811         {
00812             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel_values,temp,x,y)
00813             shared(z,c,filtered,chunk)
00814             for(x = m; x < this->get_width(); x++)
00815             {
00816                 for(y = m; y < this->get_height(); y++)
00817                 {
00818                     for(unsigned int i = x-m; i < x+m; i++)
00819                     {
00820                         for(unsigned int j = y-m; j < y+m; j++)
00821                         {
00822                             pixel_values [(i-x+m)*dim + (j-y+m)] = this->get_pixel_value(i, j, z, c);
00823                         }
00824                     }
00825                 }
00826                 for(int k=0; k<dim*dim ; k++)
00827                 {
00828                     for(int p=k+1 ; p<dim*dim ; p++)
00829                     {
00830                         if(pixel_values[p] < pixel_values[k])
00831                         {
00832                             // Intercambiar los valores
00833                             temp = pixel_values[k];
00834                             pixel_values[k] = pixel_values[p];
00835                             pixel_values[p] = temp;
00836                         }
00837                     }
00838                 }
00839                 pixel = pixel_values[((dim*dim-1)/2)-1];
00840                 #pragma omp ordered
00841                 filtered.set_pixel_value(x, y, z, c, pixel);
00842             }
00843         }
00844     }
00845 }
00846 }
00847 }
00848 return filtered;
00849 }
00850
00851 Image Image :: filter_average(int dim, int num_threads)
00852 {
00853     unsigned int y,c,z,sum;
00854     unsigned char pixel;
00855     int chunk= (this->get_width()/num_threads);
00856     omp_set_num_threads(num_threads);
00857     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00858 );
00859
00860     for(c = 0; c < this->get_spectrum(); c++)
00861     {
00862         for(z = 0; z < this->get_depth(); z++)
00863         {
00864             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,y,sum)
00865             shared(z,c,filtered,chunk)
00866             for(unsigned int x = dim; x < this->get_width()-dim; x++)
00867             {
00868                 for(y = dim; y < this->get_height()-dim; y++)
00869                 {
00870                     sum = 0;
00871                     for(unsigned int i = x-dim; i<= x+dim; i++)
00872                     {
00873                         for(unsigned int j = y-dim; j<= y+dim; j++)
00874                         {
00875                             sum += this->get_pixel_value(i, j, z, c);
00876                         }
00877                     }
00878                 }
00879                 pixel = (unsigned char)static_cast<unsigned char> (sum/((dim*2+1)*(dim*2+1)));
00880                 #pragma omp ordered
00881                 filtered.set_pixel_value(x, y, z, c, pixel);
00882             }
00883         }
00884     }
00885 }
00886 }
00887 }
00888 }
00889 }
00890 }
00891 }
00892 }
00893 return filtered;

```

```

00894 }
00895
00902 Image Image :: filter_gaussian(int o, int dim_kernel, int num_threads)
00903 {
00904     unsigned int x,y,c,z,cont;
00905     unsigned char pixel;
00906     int chunk= (this->get_width()/num_threads);
00907     omp_set_num_threads(num_threads);
00908
00909     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00910 );
00911     double kernel[dim_kernel*dim_kernel];
00912
00913     int m = (dim_kernel-1)/2;
00914
00915     double gaussian =1/pow(3.1415*2*o*o,0.5);
00916
00917     for(int i =-m; i <=m; i++)
00918     {
00919         for(int j =-m; j<=m; j++)
00920         {
00921             double exp= -(i*i+j*j)*0.5/(o*o);
00922             kernel[(i+m)*dim_kernel + (j+m)]=gaussian*pow(2.7,exp);
00923         }
00924     }
00925
00926
00927
00928
00929     for(c = 0; c < this->get_spectrum(); c++)
00930     {
00931         for(z = 0; z < this->get_depth(); z++)
00932         {
00933             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,x,y,cont)
00934             shared(z,c,filtered,chunk,kernel)
00935             for(x = m; x < this->get_width(); x++)
00936             {
00937                 for(y = m; y < this->get_height(); y++)
00938                 {
00939                     cont=0;
00940                     pixel=0;
00941
00942                     for(unsigned int i = x-m; i < x+m; i++)
00943                     {
00944                         for(unsigned int j = y-m; j< y+m; j++)
00945                         {
00946                             pixel+= this->get_pixel_value(i, j, z, c)*(kernel[cont]);
00947                             cont++;
00948                         }
00949                     }
00950                     #pragma omp ordered
00951                     filtered.set_pixel_value(x, y, z, c, (pixel/2));
00952                 }
00953             }
00954         }
00955     }
00956 }
00957
00958     return filtered;
00959 }
00960
00967 Image Image :: filter_modal(int dim, int num_threads)
00968 {
00969
00970     unsigned int x,y,c,z;
00971     unsigned char average=0;
00972     int chunk= (this->get_width()/num_threads);
00973     omp_set_num_threads(num_threads);
00974     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
00975 );
00976     unsigned char pixel_values[dim*dim];
00977     unsigned char moda;
00978
00979     int m=(dim-1)/2;
00980     unsigned char copy_pixels[dim*dim];
00981
00982     for(c = 0; c < this->get_spectrum(); c++)
00983     {
00984         for(z = 0; z < this->get_depth(); z++)

```

```

00984     {
00985         #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel_values,moda,x,y)
00986         shared(z,c,filtered,chunk,m)
00987         for(x = m; x < this->get_width(); x++)
00988         {
00989             for(y = m; y < this->get_height(); y++)
00990             {
00991                 for(unsigned int i = x-m; i < x+m; i++)
00992                 {
00993                     for(unsigned int j = y-m; j < y+m; j++)
00994                     {
00995                         pixel_values [(i-x+m)*dim + (j-y+m)] = this->get_pixel_value(i, j, z, c);
00996
00997                         int frequency[dim*dim];
00998                         moda=0;
00999
01000                         for(int k=0;k<dim*dim;k++)
01001                         {
01002                             copy_pixels[k] = pixel_values[k];
01003                             frequency[k]=0;
01004                         }
01005
01006                         for(int p=0;p<dim*dim;p++)
01007                         {
01008                             for(int q=p+1;q<dim*dim;q++)
01009                             {
01010                                 if(copy_pixels[p]==pixel_values[q]){
01011                                     frequency[p]++;
01012                                 }
01013                             }
01014                         }
01015
01016                         for(int s=0; s<dim*dim ; s++)
01017                         {
01018                             for(int e=s+1 ; e<dim*dim ; e++)
01019                             {
01020                                 if(frequency[e] < frequency[s])
01021                                 {
01022                                     moda = copy_pixels[s];
01023                                     average=copy_pixels[s];
01024                                 }
01025                             }
01026                         }
01027
01028                         if(modas==0)
01029                         {
01030                             for(int k=0;k<dim*dim;k++)
01031                             {
01032                                 moda += pixel_values[k];
01033                             }
01034                             average=(moda/dim);
01035                         }
01036
01037                     }
01038                 }
01039             }
01040             filtered.set_pixel_value(x, y, z, c, average);
01041         }
01042     }
01043 }
01044
01045 #pragma omp ordered
01046 filtered.set_pixel_value(x, y, z, c, average);
01047 }
01048 }
01049 }
01050 }
01051 }
01052 }
01053 }
01054 }
01055 return filtered;
01056 }
01057 }
01058 }
01059 }
01060 }
01061 // *****
01062 // ***** Dot to Dot Transformations *****
01063 // *****

```

```

01064
01070 Image Image ::inverse(int num_threads)
01071 {
01072     unsigned int y,c,z,x;
01073     unsigned char pixel;
01074     int chunk= (this->get_width()/num_threads);
01075     omp_set_num_threads(num_threads);
01076     Image inverted (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
01077
01078     for(c = 0; c < this->get_spectrum(); c++)
01079     {
01080         for(z = 0; z < this->get_depth(); z++)
01081         {
01082             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,x,y)
shared(z,c,inverted,chunk)
01083
01084             for(x = 0; x < this->get_width(); x++)
01085             {
01086                 for(y = 0; y < this->get_height(); y++)
01087                 {
01088                     pixel= static_cast<unsigned int>(255-this->get_pixel_value(x,y,z,c));
01089                     #pragma omp ordered
01090                     inverted.set_pixel_value(x,y,z,c,pixel);
01091                 }
01092             }
01093         }
01094     }
01095     return inverted;
01096 }
01097
01103 Image Image :: log_transformation(int num_threads)
01104 {
01105     unsigned int y,c,z,x;
01106     unsigned char pixel;
01107     int chunk= (this->get_width()/num_threads);
01108     omp_set_num_threads(num_threads);
01109     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
01110
01111     for(c = 0; c < this->get_spectrum(); c++)
01112     {
01113         for(z = 0; z < this->get_depth(); z++)
01114         {
01115             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,x,y)
shared(z,c,filtered,chunk)
01116
01117             for(x = 0; x < this->get_width(); x++)
01118             {
01119                 for(y = 0; y < this->get_height(); y++)
01120                 {
01121                     pixel = static_cast<unsigned char>((255/log(256)) * log(1+this->get_pixel_value(x, y, z
, c)));
01122                     #pragma omp ordered
01123                     filtered.set_pixel_value(x,y,z,c, pixel);
01124                 }
01125             }
01126         }
01127     }
01128     return filtered;
01129 }
01130
01141 Image Image ::filter_dynamic_range_dilatation(unsigned char a,
unsigned char b, double alpha, double beta, double gamma, int num_threads)
01142 {
01143     unsigned int y,c,z,x;
01144     unsigned char pixel;
01145     int chunk= (this->get_width()/num_threads);
01146     omp_set_num_threads(num_threads);
01147     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
01148
01149     for(c = 0; c < this->get_spectrum(); c++)
01150     {
01151         for(z = 0; z < this->get_depth(); z++)
01152         {
01153             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,x,y)
shared(z,c,filtered,chunk)
01154             for(x = 0; x < this->get_width(); x++)
01155             {
01156                 for(y = 0; y < this->get_height(); y++)

```

```

01157         {
01158             pixel=0;
01159
01160             if (this->get_pixel_value(x,y,z,c)<a)
01161                 pixel =abs(alpha*this->get_pixel_value(x,y,z,c));
01162
01163             else if (this->get_pixel_value(x,y,z,c)>=a && this->get_pixel_value(x,y,z,c)<b)
01164                 pixel=abs(beta*(this->get_pixel_value(x,y,z,c)-a)+alpha*a);
01165
01166             else if (this->get_pixel_value(x,y,z,c)<=b)
01167
01168                 pixel=abs(gamma*(this->get_pixel_value(x,y,z,c)-b)+((beta*(b-a))+alpha*a));
01169             #pragma omp ordered
01170             filtered.set_pixel_value(x,y,z,c,static_cast<unsigned int>(pixel));
01171         }
01172     }
01173 }
01174 }
01175 return filtered;
01176 }
01177
01178
01186 Image Image :: power_law_transformation(double exponent, int
num_threads)
01187 {
01188     unsigned int y,c,z,x;
01189     unsigned char pixel;
01190     int chunk= (this->get_width()/num_threads);
01191     omp_set_num_threads(num_threads);
01192     double power_law;
01193     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
01194
01195     double k = (pow(255, 1-exponent));
01196
01197     for(c = 0; c < this->get_spectrum(); c++)
01198     {
01199         for(z = 0; z < this->get_depth(); z++)
01200         {
01201             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,x,y,power_law)
shared(z,c,filtered,chunk,k)
01202
01203             for(x = 0; x < this->get_width(); x++)
01204             {
01205                 for(y = 0; y < this->get_height(); y++)
01206                 {
01207                     power_law = k * pow( (this->get_pixel_value(x,y,z,c)) , exponent);
01208                     pixel = static_cast<unsigned char>(power_law);
01209                     filtered.set_pixel_value(x, y, z, c, pixel);
01210                 }
01211             }
01212         }
01213     }
01214 }
01215 }
01216 return filtered;
01217 }
01218
01226 Image Image :: color_slicing(unsigned char color1[], unsigned char color2[],
unsigned char neutral[], int num_threads)
01227 {
01228     unsigned int y,c,z,x;
01229     unsigned char pixel;
01230     int chunk= (this->get_width()/num_threads);
01231     omp_set_num_threads(num_threads);
01232     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
01233
01234     for(c = 0; c < this->get_spectrum(); c++)
01235     {
01236         for(z = 0; z < this->get_depth(); z++)
01237         {
01238             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,x,y)
shared(z,c,filtered,chunk)
01239
01240             for(x = 0; x < this->get_width(); x++)
01241             {
01242                 for(y = 0; y < this->get_height(); y++)
01243                 {
01244                     pixel = this->get_pixel_value(x,y,z,c);
01245                     if(pixel > color1[c] && pixel < color2[c] )

```

```

01246         {
01247             #pragma omp ordered
01248             filtered.set_pixel_value(x, y, z, c, pixel);
01249         }
01250     else
01251     {
01252         #pragma omp ordered
01253         filtered.set_pixel_value(x,y,z,c, neutral[c]);
01254     }
01255 }
01256
01257     }
01258 }
01259 }
01260 }
01261
01262     return filtered;
01263 }
01264 }
01265
01266 // *****
01267 // ***** HISTOGRAM AND EQUALIZATION *****
01268 // *****
01269
01270 int* Image :: get_histogram(unsigned int c, unsigned int z)
01271 {
01272     int histogram [256];
01273     for(int i = 0; i<256; i++)
01274     {
01275         histogram[i] = 0;
01276     }
01277
01278     if (c < this->get_spectrum() && z < this->get_depth())
01279     {
01280         for(unsigned int x = 0; x < this->get_width(); x++)
01281         {
01282             for(unsigned int y = 0; y < this->get_height(); y++)
01283             {
01284                 unsigned char pixel_value = this->get_pixel_value(x,y,z,c);
01285                 (histogram[pixel_value])++;
01286             }
01287         }
01288     }
01289
01290     int* histogram_pointer = histogram;
01291
01292     return histogram_pointer;
01293 }
01294
01295 void Image :: plot_histogram(int levels,const char* title)
01296 {
01297     CImg<unsigned char> img = this->Img->histogram(levels);
01298
01299     CImgDisplay main_display (*(this->Img), title);
01300
01301     img.display_graph(main_display, 3, 1, "Pixel Intensity", 0, 0, "Frequency", 0, 0);
01302 }
01303
01304 void Image :: plot_histogram_equalization(int levels, const char* title)
01305 {
01306     CImg<unsigned char> img = this->Img->equalize(levels);
01307
01308     CImgDisplay main_display (*(this->Img), title);
01309
01310     img.display_graph(main_display, 3, 1, "Pixel Intensity", 0, 0, "Frequency", 0, 0);
01311 }
01312
01313 // *****
01314 // ***** OTHER TRANSFORMATIONS *****
01315 // *****
01316
01317 Image Image ::filter_kirsch_0(int num_threads)
01318 {
01319     unsigned int y,c,z,x;
01320     int sum;
01321     unsigned char pixel;
01322     int chunk= (this->get_width()/num_threads);
01323     omp_set_num_threads(num_threads);
01324     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01325 );

```

```

01346
01347     int m = 1;
01348
01349     for(c = 0; c < this->get_spectrum(); c++)
01350     {
01351         for(z = 0; z < this->get_depth(); z++)
01352         {
01353             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,x,y,sum)
01354             shared(z,c,filtered,chunk)
01355             for(x = m; x < this->get_width()-m; x++)
01356             {
01357                 for(y = m; y < this->get_height()-m; y++)
01358                 {
01359                     sum = -3*(get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x-1, y, z, c)+get_pixel_value
01360                     (x-1, y+1, z, c)+get_pixel_value(x, y-1, z, c)+get_pixel_value(x, y+1, z, c))+5*(get_pixel_value(x+1, y-1, z
01361                     , c)+get_pixel_value(x+1, y, z, c)+get_pixel_value(x+1, y+1, z, c));
01362                     if (sum > 255 || sum < -255)
01363                     {
01364                         sum = 255;
01365                     }
01366                     pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01367                     #pragma omp ordered
01368                     filtered.set_pixel_value(x, y, z, c, pixel);
01369                 }
01370             }
01371         }
01372     }
01373     return filtered;
01374 }
01375
01376 }
01377
01382 Image Image ::filter_kirsch_45(int num_threads)
01383 {
01384     unsigned int y,c,z,x;
01385     int sum;
01386     unsigned char pixel;
01387     int chunk= (this->get_width()/num_threads);
01388     omp_set_num_threads(num_threads);
01389     Image filtered (this->get_width() , this->get_height() , this->get_depth() , this->get_spectrum(), 0
01390 );
01391     int m = 1;
01392     for(c = 0; c < this->get_spectrum(); c++)
01393     {
01394         for(z = 0; z < this->get_depth(); z++)
01395         {
01396             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,x,y,sum)
01397             shared(z,c,filtered,chunk)
01398             for(x = m; x < this->get_width()-m; x++)
01399             {
01400                 for(y = m; y < this->get_height()-m; y++)
01401                 {
01402                     sum = -3*(get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x-1, y, z, c)+get_pixel_value
01403                     (x-1, y+1, z, c)+get_pixel_value(x+1, y+1, z, c)+get_pixel_value(x, y+1, z, c))+5*(get_pixel_value(x+1, y-1,
01404                     z, c)+get_pixel_value(x+1, y, z, c)+get_pixel_value(x, y-1, z, c));
01405                     if (sum > 255 || sum < -255)
01406                     {
01407                         sum = 255;
01408                     }
01409                     pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01410                     #pragma omp ordered
01411                     filtered.set_pixel_value(x, y, z, c, pixel);
01412                 }
01413             }
01414         }
01415     }
01416     return filtered;
01417 }
01418
01419 }
01420
01421
01426 Image Image ::filter_kirsch_90(int num_threads)
01427 {

```

```

01428     unsigned int y,c,z,x;
01429     int sum;
01430     unsigned char pixel;
01431     int chunk= (this->get_width()/num_threads);
01432     omp_set_num_threads(num_threads);
01433     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
01434     int m = 1;
01435
01436     for(c = 0; c < this->get_spectrum(); c++)
01437     {
01438         for(z = 0; z < this->get_depth(); z++)
01439         {
01440             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,x,y,sum)
shared(z,c,filtered,chunk)
01441
01442             for(x = m; x < this->get_width()-m; x++)
01443             {
01444                 for(y = m; y < this->get_height()-m; y++)
01445                 {
01446                     sum = -3*(get_pixel_value(x-1, y, z, c)+get_pixel_value(x-1, y+1, z, c)+get_pixel_value
(x, y+1, z, c)+get_pixel_value(x+1, y+1, z, c)+get_pixel_value(x+1, y, z, c))+5*(get_pixel_value(x+1, y-1, z
, c)+get_pixel_value(x, y-1, z, c)+get_pixel_value(x-1, y-1, z, c));
01447                     if (sum > 255 || sum < -255)
01448                     {
01449                         sum = 255;
01450                     }
01451                     pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01452                     #pragma omp ordered
01453                     filtered.set_pixel_value(x, y, z, c, pixel);
01454                 }
01455             }
01456         }
01457     }
01458 }
01459 }
01460
01461     return filtered;
01462 }
01463 }
01464
01465 Image Image ::filter_kirsch_135(int num_threads)
01466 {
01467     unsigned int y,c,z,x;
01468     int sum;
01469     unsigned char pixel;
01470     int chunk= (this->get_width()/num_threads);
01471     omp_set_num_threads(num_threads);
01472     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
01473     int m = 1;
01474
01475     for(c = 0; c < this->get_spectrum(); c++)
01476     {
01477         for(z = 0; z < this->get_depth(); z++)
01478         {
01479             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,x,y,sum)
shared(z,c,filtered,chunk)
01480
01481             for(x = m; x < this->get_width()-m; x++)
01482             {
01483                 for(y = m; y < this->get_height()-m; y++)
01484                 {
01485                     sum = -3*(get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x+1, y, z, c)+get_pixel_value
(x+1, y+1, z, c)+get_pixel_value(x, y+1, z, c)+get_pixel_value(x-1, y+1, z, c))+5*(get_pixel_value(x-1, y, z
, c)+get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x, y-1, z, c));
01490                     if (sum > 255 || sum < -255)
01491                     {
01492                         sum = 255;
01493                     }
01494                     pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01495                     #pragma omp ordered
01496                     filtered.set_pixel_value(x, y, z, c, pixel);
01497                 }
01498             }
01499         }
01500     }
01501 }
01502 }
01503
01504     return filtered;

```



```

01505
01506 }
01507
01512 Image Image ::filter_kirsch_180(int num_threads)
01513 {
01514     unsigned int y,c,z,x;
01515     int sum;
01516     unsigned char pixel;
01517     int chunk= (this->get_width()/num_threads);
01518     omp_set_num_threads(num_threads);
01519     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
01520     int m = 1;
01521
01522     for(c = 0; c < this->get_spectrum(); c++)
01523     {
01524         for(z = 0; z < this->get_depth(); z++)
01525         {
01526             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,x,y,sum)
01527             shared(z,c,filtered,chunk)
01528
01529             for(x = m; x < this->get_width()-m; x++)
01530             {
01531                 for(y = m; y < this->get_height()-m; y++)
01532                 {
01533                     sum = -3*(get_pixel_value(x, y-1, z, c)+get_pixel_value(x+1, y-1, z, c)+get_pixel_value
(x+1, y, z, c)+get_pixel_value(x+1, y+1, z, c)+get_pixel_value(x, y+1, z, c))+5*(get_pixel_value(x-1, y, z,
c)+get_pixel_value(x-1, y+1, z, c)+get_pixel_value(x-1, y-1, z, c));
01534                     if (sum > 255 || sum < -255)
01535                     {
01536                         sum = 255;
01537                     }
01538                     pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01539                     #pragma omp ordered
01540                     filtered.set_pixel_value(x, y, z, c, pixel);
01541                 }
01542             }
01543         }
01544     }
01545     return filtered;
01546 }
01547
01555 Image Image ::filter_kirsch_225(int num_threads)
01556 {
01557     unsigned int y,c,z,x;
01558     int sum;
01559     unsigned char pixel;
01560     int chunk= (this->get_width()/num_threads);
01561     omp_set_num_threads(num_threads);
01562     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
01563     int m = 1;
01564
01565     for(c = 0; c < this->get_spectrum(); c++)
01566     {
01567         for(z = 0; z < this->get_depth(); z++)
01568         {
01569             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,x,y,sum)
01570             shared(z,c,filtered,chunk)
01571
01572             for(x = m; x < this->get_width()-m; x++)
01573             {
01574                 for(y = m; y < this->get_height()-m; y++)
01575                 {
01576                     sum = -3*(get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x, y-1, z, c)+get_pixel_value
(x+1, y-1, z, c)+get_pixel_value(x+1, y, z, c)+get_pixel_value(x+1, y+1, z, c))+5*(get_pixel_value(x-1, y, z
, c)+get_pixel_value(x-1, y+1, z, c)+get_pixel_value(x, y+1, z, c));
01577                     if (sum > 255 || sum < -255)
01578                     {
01579                         sum = 255;
01580                     }
01581                     pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01582                     #pragma omp ordered
01583                     filtered.set_pixel_value(x, y, z, c, pixel);
01584                 }
01585             }
01586         }
01587     }
01588 }

```

```

01586
01587     }
01588 }
01589
01590     return filtered;
01591
01592 }
01593
01598 Image Image ::filter_kirsch_270(int num_threads)
01599 {
01600     unsigned int y,c,z,x;
01601     int sum;
01602     unsigned char pixel;
01603     int chunk= (this->get_width()/num_threads);
01604     omp_set_num_threads(num_threads);
01605     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01606 );
01607     int m = 1;
01608     for(c = 0; c < this->get_spectrum(); c++)
01609     {
01610         for(z = 0; z < this->get_depth(); z++)
01611         {
01612             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,x,y,sum)
01613             shared(z,c,filtered,chunk)
01614             for(x = m; x < this->get_width()-m; x++)
01615             {
01616                 for(y = m; y < this->get_height()-m; y++)
01617                 {
01618                     sum = -3*(get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x-1, y, z, c)+get_pixel_value
01619 (x, y-1, z, c)+get_pixel_value(x+1, y-1, z, c)+get_pixel_value(x+1, y, z, c))+5*(get_pixel_value(x, y+1, z,
01620 c)+get_pixel_value(x+1, y+1, z, c)+get_pixel_value(x-1, y+1, z, c));
01621                     if (sum > 255 || sum < -255)
01622                     {
01623                         sum = 255;
01624                     }
01625                     pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01626                     #pragma omp ordered
01627                     filtered.set_pixel_value(x, y, z, c, pixel);
01628                 }
01629             }
01630         }
01631     }
01632     return filtered;
01633 }
01634
01635 }
01636
01641 Image Image ::filter_kirsch_315(int num_threads)
01642 {
01643     unsigned int y,c,z,x;
01644     int sum;
01645     unsigned char pixel;
01646     int chunk= (this->get_width()/num_threads);
01647     omp_set_num_threads(num_threads);
01648     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01649 );
01650     int m = 1;
01651     for(c = 0; c < this->get_spectrum(); c++)
01652     {
01653         for(z = 0; z < this->get_depth(); z++)
01654         {
01655             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,x,y,sum)
01656             shared(z,c,filtered,chunk)
01657             for(x = m; x < this->get_width()-m; x++)
01658             {
01659                 for(y = m; y < this->get_height()-m; y++)
01660                 {
01661                     sum = -3*(get_pixel_value(x-1, y+1, z, c)+get_pixel_value(x-1, y, z, c)+get_pixel_value
01662 (x-1, y-1, z, c)+get_pixel_value(x, y-1, z, c)+get_pixel_value(x+1, y-1, z, c))+5*(get_pixel_value(x, y+1, z,
01663 , c)+get_pixel_value(x+1, y+1, z, c)+get_pixel_value(x+1, y, z, c));
01664                     if (sum > 255 || sum < -255)
01665                     {
01666                         sum = 255;
01667                     }
01668                     pixel = (unsigned char)static_cast<unsigned char> (abs(sum));

```

```

01667             #pragma omp ordered
01668             filtered.set_pixel_value(x, y, z, c, pixel);
01669         }
01670     }
01671 }
01672 }
01673 }
01674 }
01675 }
01676 return filtered;
01677 }
01678 }
01679
01684 Image Image::filter_freeman_0(int num_threads)
01685 {
01686     unsigned int y,c,z,x;
01687     int sum;
01688     unsigned char pixel;
01689     int chunk= (this->get_width()/num_threads);
01690     omp_set_num_threads(num_threads);
01691     Image filtered (this->get_width(), this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
01692
01693     int m = 1;
01694
01695     for(c = 0; c < this->get_spectrum(); c++)
01696     {
01697         for(z = 0; z < this->get_depth(); z++)
01698         {
01699             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,x,y,sum)
01700             shared(z,c,filtered,chunk)
01701             for(x = m; x < this->get_width()-m; x++)
01702             {
01703                 for(y = m; y < this->get_height()-m; y++)
01704                 {
01705                     sum = (get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x-1, y, z, c)+get_pixel_value(x-
1, y+1, z, c)+get_pixel_value(x, y-1, z, c)+get_pixel_value(x+1, y-1, z, c)+get_pixel_value(x+1,y,z,c))-1*(
get_pixel_value(x+1, y+1, z, c)+get_pixel_value(x, y+1, z, c))+2*get_pixel_value(x, y, z, c);
01706                     if (sum > 255 || sum < -255)
01707                     {
01708                         sum = 255;
01709                     }
01710                     pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01711                     #pragma omp ordered
01712                     filtered.set_pixel_value(x, y, z, c, pixel);
01713                 }
01714             }
01715         }
01716     }
01717 }
01718 }
01719
01720 return filtered;
01721 }
01722 }
01723
01727 Image Image::filter_freeman_1(int num_threads)
01728 {
01729     unsigned int y,c,z,x;
01730     int sum;
01731     unsigned char pixel;
01732     int chunk= (this->get_width()/num_threads);
01733     omp_set_num_threads(num_threads);
01734     Image filtered (this->get_width(), this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
01735
01736     int m = 1;
01737
01738     for(c = 0; c < this->get_spectrum(); c++)
01739     {
01740         for(z = 0; z < this->get_depth(); z++)
01741         {
01742             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,x,y,sum)
01743             shared(z,c,filtered,chunk)
01744             for(x = m; x < this->get_width()-m; x++)
01745             {
01746                 for(y = m; y < this->get_height()-m; y++)
01747                 {
01748                     sum = (get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x+1, y+1, z, c)+get_pixel_value(

```

```

x-1, y+1, z, c)+get_pixel_value(x, y-1, z, c)+get_pixel_value(x+1, y-1, z, c)+get_pixel_value(x+1,y,z,c))-1*
(get_pixel_value(x-1, y, z, c)+get_pixel_value(x, y+1, z, c))+2*get_pixel_value(x, y, z, c);
01749         if (sum > 255 || sum < -255)
01750         {
01751             sum = 255;
01752         }
01753         pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01754         #pragma omp ordered
01755         filtered.set_pixel_value(x, y, z, c, pixel);
01756     }
01757 }
01758 }
01759 }
01760 }
01761 }
01762 }
01763 return filtered;
01764 }
01765 }
01766
01770 Image Image ::filter_freeman_2(int num_threads)
01771 {
01772     unsigned int y,c,z,x;
01773     int sum;
01774     unsigned char pixel;
01775     int chunk= (this->get_width()/num_threads);
01776     omp_set_num_threads(num_threads);
01777     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
01778
01779     int m = 1;
01780
01781     for(c = 0; c < this->get_spectrum(); c++)
01782     {
01783         for(z = 0; z < this->get_depth(); z++)
01784         {
01785             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,x,y,sum)
shared(z,c,filtered,chunk)
01786
01787             for(x = m; x < this->get_width()-m; x++)
01788             {
01789                 for(y = m; y < this->get_height()-m; y++)
01790                 {
01791                     sum = (get_pixel_value(x, y-1, z, c)+get_pixel_value(x+1, y-1, z, c)+get_pixel_value(x+
1, y, z, c)+get_pixel_value(x+1, y+1, z, c)+get_pixel_value(x, y+1, z, c)+get_pixel_value(x-1,y+1,z,c))-1*(
get_pixel_value(x-1, y, z, c)+get_pixel_value(x-1, y-1, z, c))+2*get_pixel_value(x, y, z, c);
01792                     if (sum > 255 || sum < -255)
01793                     {
01794                         sum = 255;
01795                     }
01796                     pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01797                     #pragma omp ordered
01798                     filtered.set_pixel_value(x, y, z, c, pixel);
01799                 }
01800             }
01801         }
01802     }
01803 }
01804 }
01805
01806 return filtered;
01807 }
01808 }
01809
01813 Image Image ::filter_freeman_3(int num_threads)
01814 {
01815     unsigned int y,c,z,x;
01816     int sum;
01817     unsigned char pixel;
01818     int chunk= (this->get_width()/num_threads);
01819     omp_set_num_threads(num_threads);
01820     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
01821
01822     int m = 1;
01823
01824     for(c = 0; c < this->get_spectrum(); c++)
01825     {
01826         for(z = 0; z < this->get_depth(); z++)
01827         {
01828             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,x,y,sum)

```

```

        shared(z,c,filtered,chunk)
01829
01830         for(x = m; x < this->get_width()-m; x++)
01831         {
01832             for(y = m; y < this->get_height()-m; y++)
01833             {
01834                 sum = (get_pixel_value(x+1, y-1, z, c)+get_pixel_value(x+1, y, z, c)+get_pixel_value(x+
1, y+1, z, c)+get_pixel_value(x, y+1, z, c)+get_pixel_value(x-1,y+1,z,c))-1*(get_pixel_value(x-1, y, z, c)+
get_pixel_value(x, y-1, z, c)+get_pixel_value(x-1,y-1,z,c))+2*get_pixel_value(x, y, z, c);
01835                 if (sum > 255 || sum < -255)
01836                 {
01837                     sum = 255;
01838                 }
01839                 pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01840                 #pragma omp ordered
01841                 filtered.set_pixel_value(x, y, z, c, pixel);
01842             }
01843         }
01844     }
01845 }
01846 }
01847 }
01848
01849     return filtered;
01850 }
01851 }
01852
01856 Image Image ::filter_freeman_4(int num_threads)
01857 {
01858     unsigned int y,c,z,x;
01859     int sum;
01860     unsigned char pixel;
01861     int chunk= (this->get_width()/num_threads);
01862     omp_set_num_threads(num_threads);
01863     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
01864
01865     int m = 1;
01866
01867     for(c = 0; c < this->get_spectrum(); c++)
01868     {
01869         for(z = 0; z < this->get_depth(); z++)
01870         {
01871             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,x,y,sum)
shared(z,c,filtered,chunk)
01872
01873             for(x = m; x < this->get_width()-m; x++)
01874             {
01875                 for(y = m; y < this->get_height()-m; y++)
01876                 {
01877                     sum = (get_pixel_value(x-1, y, z, c)+get_pixel_value(x-1, y+1, z, c)+get_pixel_value(x,
y+1, z, c)+get_pixel_value(x+1, y+1, z, c)+get_pixel_value(x+1, y, z, c))-1*(get_pixel_value(x-1, y-1, z, c
)+get_pixel_value(x, y-1, z, c)+get_pixel_value(x+1,y-1,z,c))+2*get_pixel_value(x, y, z, c);
01878                     if (sum > 255 || sum < -255)
01879                     {
01880                         sum = 255;
01881                     }
01882                     pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01883                     #pragma omp ordered
01884                     filtered.set_pixel_value(x, y, z, c, pixel);
01885                 }
01886             }
01887         }
01888     }
01889 }
01890 }
01891
01892     return filtered;
01893 }
01894 }
01895
01896
01900 Image Image ::filter_freeman_5(int num_threads)
01901 {
01902     unsigned int y,c,z,x;
01903     int sum;
01904     unsigned char pixel;
01905     int chunk= (this->get_width()/num_threads);
01906     omp_set_num_threads(num_threads);
01907     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);

```

```

01908
01909     int m = 1;
01910
01911     for(c = 0; c < this->get_spectrum(); c++)
01912     {
01913         for(z = 0; z < this->get_depth(); z++)
01914         {
01915             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,x,y,sum)
01916             shared(z,c,filtered,chunk)
01917             for(x = m; x < this->get_width()-m; x++)
01918             {
01919                 for(y = m; y < this->get_height()-m; y++)
01920                 {
01921                     sum = (get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x-1, y, z, c)+get_pixel_value(x-
01922                     1, y+1, z, c)+get_pixel_value(x, y+1, z, c)+get_pixel_value(x+1, y+1, z, c))-1*(get_pixel_value(x+1, y, z, c
01923                     )+get_pixel_value(x, y-1, z, c)+get_pixel_value(x+1,y-1,z,c))+2*get_pixel_value(x, y, z, c);
01924                     if (sum > 255 || sum < -255)
01925                     {
01926                         sum = 255;
01927                     }
01928                     pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01929                     #pragma omp ordered
01930                     filtered.set_pixel_value(x, y, z, c, pixel);
01931                 }
01932             }
01933         }
01934     }
01935     return filtered;
01936 }
01937
01938 }
01939
01940
01941 Image Image ::filter_freeman_6(int num_threads)
01942 {
01943     unsigned int y,c,z,x;
01944     int sum;
01945     unsigned char pixel;
01946     int chunk= (this->get_width()/num_threads);
01947     omp_set_num_threads(num_threads);
01948     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
01949 );
01950
01951     int m = 1;
01952
01953     for(c = 0; c < this->get_spectrum(); c++)
01954     {
01955         for(z = 0; z < this->get_depth(); z++)
01956         {
01957             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,x,y,sum)
01958             shared(z,c,filtered,chunk)
01959             for(x = m; x < this->get_width()-m; x++)
01960             {
01961                 for(y = m; y < this->get_height()-m; y++)
01962                 {
01963                     sum = (get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x-1, y, z, c)+get_pixel_value(x-
01964                     1, y+1, z, c)+get_pixel_value(x, y+1, z, c)+get_pixel_value(x, y-1, z, c))-1*(get_pixel_value(x+1, y-1, z, c
01965                     )+get_pixel_value(x+1, y, z, c)+get_pixel_value(x+1,y+1,z,c))+2*get_pixel_value(x, y, z, c);
01966                     if (sum > 255 || sum < -255)
01967                     {
01968                         sum = 255;
01969                     }
01970                     pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
01971                     #pragma omp ordered
01972                     filtered.set_pixel_value(x, y, z, c, pixel);
01973                 }
01974             }
01975         }
01976     }
01977     return filtered;
01978 }
01979
01980 }
01981
01982 Image Image ::filter_freeman_7(int num_threads)

```

```

01988 {
01989     unsigned int y,c,z,x;
01990     int sum;
01991     unsigned char pixel;
01992     int chunk= (this->get_width()/num_threads);
01993     omp_set_num_threads(num_threads);
01994     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
01995
01996     int m = 1;
01997
01998     for(c = 0; c < this->get_spectrum(); c++)
01999     {
02000         for(z = 0; z < this->get_depth(); z++)
02001         {
02002             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,x,y,sum)
shared(z,c,filtered,chunk)
02003
02004             for(x = m; x < this->get_width()-m; x++)
02005             {
02006                 for(y = m; y < this->get_height()-m; y++)
02007                 {
02008                     sum = (get_pixel_value(x-1, y-1, z, c)+get_pixel_value(x-1, y, z, c)+get_pixel_value(x-
1, y+1, z, c)+get_pixel_value(x+1, y-1, z, c)+get_pixel_value(x, y-1, z, c))-1*(get_pixel_value(x, y+1, z, c
)+get_pixel_value(x+1, y, z, c)+get_pixel_value(x+1,y+1,z,c))+2*get_pixel_value(x, y, z, c);
02009                     if (sum > 255 || sum < -255)
02010                     {
02011                         sum = 255;
02012                     }
02013                     pixel = (unsigned char)static_cast<unsigned char> (abs(sum));
02014                     #pragma omp ordered
02015                     filtered.set_pixel_value(x, y, z, c, pixel);
02016                 }
02017             }
02018         }
02019     }
02020 }
02021 }
02022
02023 return filtered;
02024 }
02025 }
02026
02033 Image Image :: filter_maximum(int num_threads)
02034 {
02035     unsigned int y,c,z,x;
02036     unsigned char max,pixel;
02037     int chunk= (this->get_width()/num_threads);
02038     omp_set_num_threads(num_threads);
02039     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
02040
02041     for(c = 0; c < this->get_spectrum(); c++)
02042     {
02043         for(z = 0; z < this->get_depth(); z++)
02044         {
02045             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,max,x,y)
shared(z,c,filtered,chunk)
02046
02047             for(x = 1; x < this->get_width()-1; x++)
02048             {
02049                 for(y = 1; y < this->get_height()-1; y++)
02050                 {
02051                     max = 0;
02052
02053                     for (unsigned int i = x-1; i < x+2; i++)
02054                     {
02055                         for (unsigned int j = y-1; j < y+2; j++)
02056                         {
02057                             pixel = (this->get_pixel_value(i, j, z, c));
02058
02059                             if (pixel > max)
02060                             {
02061                                 max = this->get_pixel_value(i, j, z, c);
02062                             }
02063                         }
02064                     }
02065                     #pragma omp ordered
02066                     filtered.set_pixel_value(x, y, z, c, max);
02067                 }
02068             }

```

```

02069         }
02070     }
02071 }
02072 }
02073 return filtered;
02074 }
02075
02076
02082 Image Image :: filter_minimum(int num_threads)
02083 {
02084     unsigned int y,c,z,x;
02085     unsigned char minimun;
02086     int chunk= (this->get_width()/num_threads);
02087     omp_set_num_threads(num_threads);
02088     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
02089
02090     for(c = 0; c < this->get_spectrum(); c++)
02091     {
02092         for(z = 0; z < this->get_depth(); z++)
02093         {
02094             #pragma omp parallel for ordered schedule(dynamic,chunk) private(minimun,x,y)
shared(z,c,filtered,chunk)
02095
02096             for(x = 1; x < this->get_width()-1; x++)
02097             {
02098                 for(y = 1; y < this->get_height()-1; y++)
02099                 {
02100                     minimun = 255;
02101
02102                     for (unsigned int i = x-1; i< x+2; i++)
02103                     {
02104                         for (unsigned int j = y-1; j< y+2; j++)
02105                         {
02106                             if ((this->get_pixel_value(i, j, z, c)) < minimun)
02107                             {
02108                                 minimun = this->get_pixel_value(i, j, z, c);
02109                             }
02110                         }
02111                     }
02112
02113                     filtered.set_pixel_value(x, y, z, c, minimun);
02114                 }
02115             }
02116         }
02117     }
02118 }
02119 return filtered;
02120 }
02121 }
02122
02123 Image Image :: filter_order_stadistics(int dim, int order, int
num_threads)
02124 {
02125     unsigned int y,c,z,x;
02126     unsigned char pixel;
02127     int chunk= (this->get_width()/num_threads);
02128     omp_set_num_threads(num_threads);
02129     Image filtered (this->get_width() , this->get_height(), this->get_depth(), this->get_spectrum(), 0
);
02130
02131     int m = (dim-1)/2;
02132
02133     unsigned char pixel_values [dim*dim];
02134     unsigned char temp;
02135
02136     for(c = 0; c < this->get_spectrum(); c++)
02137     {
02138         for(z = 0; z < this->get_depth(); z++)
02139         {
02140             #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel,x,y,pixel_values,temp)
shared(z,c,filtered,chunk)
02142
02143             for(x = m; x < this->get_width(); x++)
02144             {
02145                 for(y = m; y < this->get_height(); y++)
02146                 {
02147                     for(unsigned int i = x-m; i < x+m+1; i++)
02148                     {
02149                         for(unsigned int j = y-m; j< y+m+1; j++)

```



```

02150         {
02151             pixel_values [(i-x+m)*dim + (j-y+m)] = this->get_pixel_value(i, j, z, c);
02152         }
02153     }
02154 }
02155 for(int k=0; k<dim*dim ; k++)
02156 {
02157     for(int p=k+1 ; p<dim*dim ; p++)
02158     {
02159         if(pixel_values[p] < pixel_values[k])
02160         {
02161             // Intercambiar los valores
02162             temp = pixel_values[k];
02163             pixel_values[k] = pixel_values[p];
02164             pixel_values[p] = temp;
02165         }
02166     }
02167 }
02168 pixel = pixel_values[order];
02169 #pragma omp ordered
02170 filtered.set_pixel_value(x, y, z, c, pixel);
02171 }
02172 }
02173 }
02174 }
02175 }
02176 }
02177 return filtered;
02178 }
02179 }
02180 }
02181 // *****
02182 // ***** NOISES *****
02183 // *****
02184 void Image :: salt_pepper(double intensity, int num_threads)
02185 {
02186     srand(1);
02187     unsigned int x,y,z,c;
02188     int chunk= (this->get_width()/num_threads);
02189     double percentage = 1-(intensity/100);
02190     omp_set_num_threads(num_threads);
02191     for(c = 0; c < this->get_spectrum(); c++)
02192     {
02193         for(z = 0; z < this->get_depth(); z++)
02194         {
02195             #pragma omp parallel for ordered schedule(dynamic,chunk) private(x,y) shared(z,c,chunk)
02196             for(x = 0; x < this->get_width(); x++)
02197             {
02198                 for(y = 0; y < this->get_height(); y++)
02199                 {
02200                     double random= 2.0*(rand()-RAND_MAX/2.0)/RAND_MAX;
02201                     if(random > percentage)
02202                     {
02203                         #pragma omp ordered
02204                         (*(this->Img))(x, y, z, c) = 255;
02205                     }
02206                     else if(random<-1*percentage)
02207                     {
02208                         #pragma omp ordered
02209                         (*(this->Img))(x, y, z, c) = 0;
02210                     }
02211                 }
02212             }
02213         }
02214     }
02215 }
02216 }
02217 }
02218 }
02219 }
02220 }
02221 }
02222 }
02223 }
02224 }
02225 }
02226 }
02227 void Image :: gaussian_noise(double variance, int num_threads)
02228 {
02229     srand(1);
02230     unsigned int x,y,z,c;
02231     unsigned char pixel;
02232     double random;
02233     omp_set_num_threads(num_threads);

```

```

02240     int chunk= (this->get_width()/num_threads);
02241     for(c = 0; c < this->get_spectrum(); c++)
02242     {
02243         for(z = 0; z < this->get_depth(); z++)
02244         {
02245             #pragma omp parallel for ordered schedule(dynamic,chunk) private(x,y,pixel, random)
02246             shared(z,c,chunk)
02247             for(x = 0; x < this->get_width(); x++)
02248             {
02249                 for(y = 0; y < this->get_height(); y++)
02250                 {
02251                     random= variance*(rand()-RAND_MAX/variante)/RAND_MAX;
02252                     pixel= this->get_pixel_value(x,y,z,c) + random;
02253
02254                     if((pixel<255) & (pixel>0))
02255                     {
02256                         #pragma omp ordered
02257                         (*(this->Img))(x, y, z, c)= pixel;
02258                     }
02259                 }
02260             }
02261         }
02262     }
02263 }
02264
02265 }
02266 }
02267 }
02268 }
02269
02270
02276 Image Image :: interpolation(int num_threads)
02277 {
02278     int i,j=0;
02279     unsigned int x,y,z,c;
02280     unsigned char pixel;
02281     int chunk= (this->get_width()/num_threads);
02282     omp_set_num_threads(num_threads);
02283     Image result (2*this->get_width(),2*this->get_height(),this->get_depth(),this->get_spectrum(),0);
02284     for(c = 0; c < this->get_spectrum(); c++)
02285     {
02286         for(z = 0; z < this->get_depth(); z++)
02287         {
02288             #pragma omp parallel for ordered schedule(dynamic,chunk) private(x,y,pixel) shared(z,c,chunk)
02289
02290             for(y = 0; y < this->get_width(); y++)
02291             {
02292                 for(x = 0; x < this->get_height(); x++)
02293                 {
02294                     pixel=this->get_pixel_value(x,y,z,c);
02295
02296                     #pragma omp ordered
02297                     {
02298                         result.set_pixel_value(x+i,y+j,z,c,pixel);
02299                         result.set_pixel_value(x+1+i,y+j,z,c,pixel);
02300                         result.set_pixel_value(x+i,y+1+j,z,c,pixel);
02301                         result.set_pixel_value(x+1+i,y+1+j,z,c,pixel);
02302                     }
02303                     ++i;
02304                 }
02305                 i=0;
02306                 ++j;
02307             }
02308             j=0;
02309         }
02310     }
02311     return result;
02312 }
02313
02327 CImg<float> Image:: autocovariance (int hor_dis, int ver_dis,int num_threads)
02328 {
02329     CImg<float> autocovariance (this->get_width() , this->get_height(), this->get_depth(),
02330     this->get_spectrum(), 0);
02331     int chunk= (this->get_width()/num_threads);
02332     unsigned int x,y,z,c;
02333     int sum;
02334     omp_set_num_threads(num_threads);
02335     Image average = this->filter_average(1,num_threads);
02336     for(c = 0; c < this->get_spectrum(); c++)

```

```

02337     {
02338         for(z = 0; z < this->get_depth(); z++)
02339         {
02340             #pragma omp parallel for ordered schedule(dynamic,chunk) private(x,y,sum) shared(z,c,chunk)
02341
02342             for(x = 3+hor_dis; x < this->get_width()-(3+hor_dis); x++)
02343             {
02344                 for(y = 3+ver_dis; y < this->get_height()-(3+ver_dis); y++)
02345                 {
02346                     sum = 0;
02347                     for(unsigned int i = x-3; i<x+4; i++)
02348                     {
02349                         for(unsigned int j = y-3; j<y+4; j++)
02350                         {
02351                             sum += ( (this->get_pixel_value(i,j,z,c)) - average.get_pixel_value(i,j,z,c))
* ( (this->get_pixel_value(i+hor_dis,j+ver_dis,z,c)) - average.get_pixel_value(i+hor_dis,j+ver_dis,z,c))
;
02352                             }
02353                         }
02354                     #pragma omp ordered
02355                     autocovariance(x,y,z,c) = sum/49;
02356                 }
02357             }
02358         }
02359     }
02360     return autocovariance;
02361 }
02362
02369 Image Image :: variance(int dim, int num_threads)
02370 {
02371     Image filtered (this->get_width() , this->get_height() , this->get_depth() , this->get_spectrum(), 0
);
02372     int chunk= (this->get_width()/num_threads);
02373     unsigned int x,y,z,c;
02374     unsigned char pixel;
02375     int sum;
02376     omp_set_num_threads(num_threads);
02377     for(c = 0; c < this->get_spectrum(); c++)
02378     {
02379         for(z = 0; z < this->get_depth(); z++)
02380         {
02381             #pragma omp parallel for ordered schedule(dynamic,chunk) private(x,y,sum, pixel)
shared(z,c,chunk)
02382
02383             for(x = dim; x < this->get_width()-dim; x++)
02384             {
02385                 for(y = dim; y < this->get_height()-dim; y++)
02386                 {
02387                     sum = 0;
02388                     double variance=0;
02389                     int kernel_values[(dim*2+1)*(dim*2+1)];
02390                     int cont=0;
02391
02392                     for(unsigned int i = x-dim; i<= x+dim; i++)
02393                     {
02394                         for(unsigned int j = y-dim; j<= y+dim; j++)
02395                         {
02396                             sum += this->get_pixel_value(i, j, z, c);
02397                             kernel_values[cont]=this->get_pixel_value(i, j, z, c);
02398                             cont++;
02399                         }
02400                     }
02401
02402                     double average = sum/((dim*2+1)*(dim*2+1));
02403                     for(int i=0;i<(dim*2+1)*(dim*2+1);i++)
02404                     {
02405                         variance+=pow(kernel_values[i]-average,2)/((dim*2+1)*(dim*2+1));
02406                     }
02407
02408                     pixel = (unsigned char)static_cast<unsigned char> (variance);
02409                     #pragma omp ordered
02410                     filtered.set_pixel_value(x, y, z, c, pixel);
02411                 }
02412             }
02413         }
02414     }
02415 }
02416
02417 }
02418     return filtered;
02419 }

```

```

02420
02428 Image Image :: gray_scale(int num_threads)
02429 {
02430     unsigned int y,z,x;
02431     unsigned char pixel_intensity;
02432     int chunk= (this->get_width()/num_threads);
02433     omp_set_num_threads(num_threads);
02434     Image gray_image (this->get_width() , this->get_height(), this->get_depth(), 1, 0);
02435
02436     for(z = 0; z < this->get_depth(); z++)
02437     {
02438         #pragma omp parallel for ordered schedule(dynamic,chunk) private(pixel_intensity,x,y)
02439         shared(z,gray_image,chunk)
02440
02441         for(x = 0; x < this->get_width(); x++)
02442         {
02443             for(y = 0; y < this->get_height(); y++)
02444             {
02445                 pixel_intensity = 0.56*this->get_pixel_value(x,y,z,1)+0.14*this->get_pixel_value(x,y,z,0)+0
02446                 .11*this->get_pixel_value(x,y,z,2);
02447                 #pragma omp ordered
02448                 gray_image.set_pixel_value(x, y, z, 0, pixel_intensity);
02449             }
02450         }
02451     }
02452 }
02453 }
02454 return gray_image;
02455 }
02456
02463 Image Image :: coorrelogram(unsigned int ver,unsigned int hor, int num_threads)
02464 {
02465     Image result (256,256, 1, 1, 0);
02466     int chunk= (this->get_width()/num_threads);
02467     unsigned int i,j,x,y;
02468     unsigned int pixel;
02469     unsigned char first, secnd;
02470     omp_set_num_threads(num_threads);
02471     for(i = 0; i < 256; i++)
02472     {
02473         for(j=0; j< 256; j++)
02474         {
02475             pixel = 0;
02476
02477             #pragma omp parallel for ordered schedule(dynamic,chunk) private(x,y,first, secnd, pixel)
02478             shared(i,j,chunk)
02479
02480             for(x=0; x< (this->get_width()-hor);++x)
02481             {
02482                 for(y=0; y< (this->get_height()-ver);++y)
02483                 {
02484                     first = (this->get_pixel_value(x,y,0,0));
02485                     secnd = (this->get_pixel_value(x+hor, y+ver, 0, 0));
02486
02487                     if(first == i && secnd == j)
02488                     {
02489                         pixel ++;
02490                     }
02491                 }
02492             }
02493         }
02494         if(pixel>255)
02495         {
02496             pixel=255;
02497         }
02498         #pragma omp ordered
02499         result.set_pixel_value(i, j, 0, 0, pixel);
02500     }
02501 }
02502
02503 cout<<"\n"<<i<<"\n"<<endl;
02504 }
02505 return result;
02506
02507 }
02508
02517 Image Image :: coorrelogram_ZC(unsigned int ver,unsigned int hor, unsigned int
    z, unsigned int c, int num_threads)

```

```

02518 {
02519     Image result (256,256, 1, 1, 0);
02520     int chunk= (this->get_width()/num_threads);
02521     unsigned int i,j,x,y;
02522     unsigned int pixel;
02523     unsigned char first, secnd;
02524     omp_set_num_threads(num_threads);
02525     for(i = 0; i < 256; i++)
02526     {
02527         for(j=0; j< 256; j++)
02528         {
02529             pixel = 0;
02530             #pragma omp parallel for ordered schedule(dynamic,chunk) private(x,y,first, secnd, pixel)
02531             shared(i,j,chunk)
02532             for(x=0; x< (this->get_width()-hor);++x)
02533             {
02534                 for(y=0; y< (this->get_height()-ver);++y)
02535                 {
02536                     first = (this->get_pixel_value(x,y,z,c));
02537                     secnd = (this->get_pixel_value(x+hor, y+ver, z, c));
02538                     if(first == i && secnd == j)
02539                     {
02540                         pixel ++;
02541                     }
02542                 }
02543             }
02544             if(pixel>255)
02545             {
02546                 pixel=255;
02547             }
02548             #pragma omp ordered
02549             result.set_pixel_value(i, j, 0, 0, pixel);
02550         }
02551     }
02552     return result;
02553 }
02554 }
02555 }
02556 }
02557 }
02558 }
02559 }
02560 }
02561 }
02562 }

```

# Index

~Image  
    Image, [9](#)  
/home/fish/Documents/ParallelPic/Proyecto/include/-  
    ParallelPic.hh, [45](#)  
/home/fish/Documents/ParallelPic/Proyecto/src/Parallel-  
    Pic.cpp, [73](#)  
/home/fish/Documents/ParallelPic/Proyecto/src/image.-  
    cpp, [48](#)

autocovariance  
    Image, [9](#)  
average\_omp  
    Image, [10](#)

binarize\_img  
    Image, [10](#)

color\_slicing  
    Image, [10](#)  
coorrelogram  
    Image, [11](#)  
coorrelogram\_ZC  
    Image, [12](#)  
coorrelogram\_par  
    Image, [12](#)

display  
    Image, [13](#)

filter\_Gradient\_horizontal  
    Image, [20](#)  
filter\_Gradient\_vertical  
    Image, [21](#)  
filter\_Laplacian  
    Image, [26](#)  
filter\_Laplacian\_no\_diagonal  
    Image, [27](#)  
filter\_Prewitt\_E\_W  
    Image, [32](#)  
filter\_Prewitt\_N\_S  
    Image, [32](#)  
filter\_Prewitt\_NE\_SW  
    Image, [33](#)  
filter\_Prewitt\_NW\_SE  
    Image, [34](#)  
filter\_average  
    Image, [13](#)  
filter\_dynamic\_range\_dilatation  
    Image, [13](#)  
filter\_edge\_enhancement\_displacement  
    Image, [14](#)  
filter\_freeman\_0  
    Image, [15](#)  
filter\_freeman\_1  
    Image, [15](#)  
filter\_freeman\_2  
    Image, [16](#)  
filter\_freeman\_3  
    Image, [16](#)  
filter\_freeman\_4  
    Image, [17](#)  
filter\_freeman\_5  
    Image, [17](#)  
filter\_freeman\_6  
    Image, [18](#)  
filter\_freeman\_7  
    Image, [19](#)  
filter\_gaussian  
    Image, [19](#)  
filter\_horizontal\_borders  
    Image, [21](#)  
filter\_kirsch\_0  
    Image, [22](#)  
filter\_kirsch\_135  
    Image, [22](#)  
filter\_kirsch\_180  
    Image, [23](#)  
filter\_kirsch\_225  
    Image, [23](#)  
filter\_kirsch\_270  
    Image, [24](#)  
filter\_kirsch\_315  
    Image, [25](#)  
filter\_kirsch\_45  
    Image, [25](#)  
filter\_kirsch\_90  
    Image, [26](#)  
filter\_maximum  
    Image, [28](#)  
filter\_median  
    Image, [28](#)

- filter\_minimum
  - Image, 29
- filter\_modal
  - Image, 30
- filter\_order\_statistics
  - Image, 31
- filter\_vertical\_borders
  - Image, 34
- gaussian\_noise
  - Image, 35
- get\_histogram
  - Image, 35
- gray\_scale
  - Image, 36
- histogram\_equalization
  - Image, 36
- Image, 5
  - ~Image, 9
  - autocovariance, 9
  - average\_omp, 10
  - binarize\_img, 10
  - color\_slicing, 10
  - coorrelogram, 11
  - coorrelogram\_ZC, 12
  - coorrelogram\_par, 12
  - display, 13
  - filter\_Gradient\_horizontal, 20
  - filter\_Gradient\_vertical, 21
  - filter\_Laplacian, 26
  - filter\_Laplacian\_no\_diagonal, 27
  - filter\_Prewitt\_E\_W, 32
  - filter\_Prewitt\_N\_S, 32
  - filter\_Prewitt\_NE\_SW, 33
  - filter\_Prewitt\_NW\_SE, 34
  - filter\_average, 13
  - filter\_dynamic\_range\_dilatation, 13
  - filter\_edge\_enhancement\_displacement, 14
  - filter\_freeman\_0, 15
  - filter\_freeman\_1, 15
  - filter\_freeman\_2, 16
  - filter\_freeman\_3, 16
  - filter\_freeman\_4, 17
  - filter\_freeman\_5, 17
  - filter\_freeman\_6, 18
  - filter\_freeman\_7, 19
  - filter\_gaussian, 19
  - filter\_horizontal\_borders, 21
  - filter\_kirsch\_0, 22
  - filter\_kirsch\_135, 22
  - filter\_kirsch\_180, 23
  - filter\_kirsch\_225, 23
  - filter\_kirsch\_270, 24
  - filter\_kirsch\_315, 25
  - filter\_kirsch\_45, 25
  - filter\_kirsch\_90, 26
  - filter\_maximum, 28
  - filter\_median, 28
  - filter\_minimum, 29
  - filter\_modal, 30
  - filter\_order\_statistics, 31
  - filter\_vertical\_borders, 34
  - gaussian\_noise, 35
  - get\_histogram, 35
  - gray\_scale, 36
  - histogram\_equalization, 36
  - Image, 8
  - interpolation, 36
  - inverse, 37
  - log\_transformation, 37
  - median\_omp, 38
  - multiply\_img, 38
  - plot\_histogram, 38
  - plot\_histogram\_equalization, 40
  - power\_law\_transformation, 40
  - rgb\_hsv, 40
  - salt\_pepper, 41
  - set\_pixel\_value, 41
  - subtract\_img, 41
  - sum\_img, 42
  - variance, 43
- interpolation
  - Image, 36
- inverse
  - Image, 37
- log\_transformation
  - Image, 37
- median\_omp
  - Image, 38
- multiply\_img
  - Image, 38
- plot\_histogram
  - Image, 38
- plot\_histogram\_equalization
  - Image, 40
- power\_law\_transformation
  - Image, 40
- rgb\_hsv
  - Image, 40
- salt\_pepper
  - Image, 41
- set\_pixel\_value
  - Image, 41

subtract\_img  
    Image, [41](#)  
sum\_img  
    Image, [42](#)  
variance  
    Image, [43](#)