PROJECT 2

ECE 763

VARIABLE CONDUCTANCE DIFFUSION

AUTHOR:

PRATHAMESH PRABHUDESAI
(ppprabhu@ncsu.edu)

Index

1. Introduction

The main aim of the project is to remove the dark current noise, present in the image. In the Charged Coupled Devices (CCD), dark current arises due to the thermal energy stored in the semiconductor lattice. Dark current noise is just the statistical fluctuation of this quantity.

Dark current noise is inevitable in nature. It has a Gaussian distribution and it is present in every image. Its presence can be seen as roughness in the image. This roughness has to be removed for object detection and information extraction. Blurring or smoothing can help us in this. Noise can be considered a high frequency component in an image. Smoothing can help us reducing these high frequency points making transition from one intensity value to other intensity value gradual. Wait! Edges are also the high frequency regions in an image. Then smoothing will destroy edge content too. Edges are the prime source of information as with the help of them we detect and analyze objects. We need something which can preserve the edges and still smooth the image. Here comes in picture Edge Preserving Smoothing algorithms.

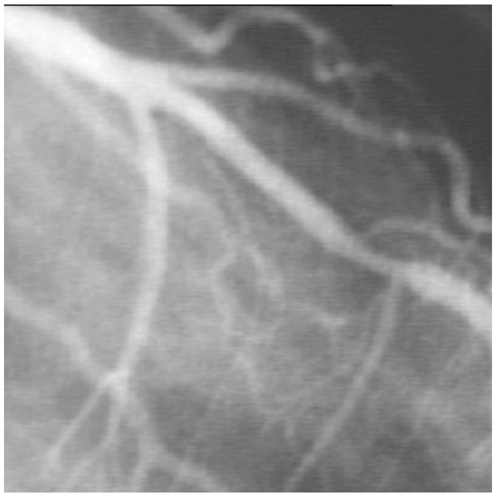
There are different methods such as use of Bilateral Filtering, use of optimization method etc. In this project we will focus on Variable Conductance Diffusion Edge Preserving Smoothing.

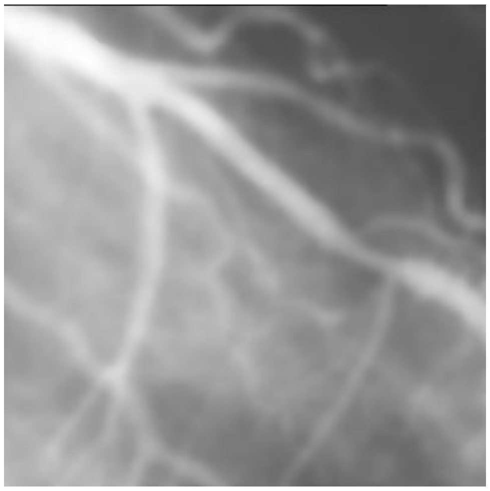
This algorithm suggests a simulation of a two-dimensional partial derivative equation called a 'Heat Equation' or 'Diffusion Equation' on an image which will end up giving a smoothened image after certain number of iterations with preserved edges. The Diffusion Equation is given by,

$$\frac{\partial f_i}{\partial x} = \alpha(\nabla^T(c\nabla f)|i$$

where, c is a scalar known as 'conductance' and $\nabla = \left[\frac{\partial}{\partial x}\ \frac{\partial}{\partial y}\right]^T$

After the simulation of this PDE, we get the smoothened image with our preserved edges as seen below.


Input Image


Output Image

2. Approach

2.1. The Diffusion Equation:

All the equations in section 2.1 and 2.2 are taken from "Fundamental Principles of Computer Vision" by Wesley Snyder & Hairong Qi.

The Diffusion Equation or Heat Equation in one dimension can be written as follows:

$$\frac{\partial f(x,t)}{\partial x} = \frac{\partial}{\partial x}\left[c\frac{\partial f(x,t)}{\partial x}\right]$$

Where c is a scalar quantity known as 'conductance'.

Since image is a two dimensional quantity, we will need to modify this equation a little bit. We will introduce a ∇ operator which is nothing but a partial derivative operator in two dimensions.

Our equation then becomes

$$\frac{\partial f_i}{\partial x} = \alpha(\nabla^T(c\nabla f)|_{\mathrm{i}}$$

$$\text{where } \nabla = \left[\frac{\partial}{\partial x}\ \frac{\partial}{\partial y}\right]^T$$

Here we introduced α which is a rate of change. We are supposed to simulate this equation over i instances of an image. But it is still unclear what this equation is trying to suggest.

Consider an image f. We will take operate ∇ on it, which will give us the derivative of that image in x and y direction. We will then multiply these images with c some scalar which will calculate in next section. After the multiplication we will again take the derivatives and add those two images. After multiplication with alpha we will add the obtained image to the original image. We will simulate this over number of iterations and finally get a smooth image. This is the basic idea behind VCD algorithm.

2.2. Calculation of conductance:

Consider c as a scalar function of position i.e. $c(x,y)$. We can see this function as an amount of smoothness you want at a particular point in the image.

We want to preserve edges in our image. The c value should be small at our edges and we want to blur rest of the things, hence c should be higher in the area other than the edges.

$c(x,y)$ is a function of space. We want it to be near zero in the vicinity of the edges where apparently the intensity values are higher and we want it to be high in the other areas where the intensity values are comparatively less. Which function has a behavior like that. A negative exponential function might help us in this case.

Let's define the function for calculation of $c(x,y)$.

$$c(x,y) = \exp\left(-\Lambda(f)(x,y)\right)$$

Where Λ will provide some measure for strength of edges.

Consider derivative of the image at the edges. It will have higher values compared to non edge vicinity. So we can define our function as,

$$c(x,y) = \exp\left(-\frac{f_x^2(x,y) + f_y^2(x,y)}{\tau}\right)$$

where τ is the scaling factor.
At edges, the numerator of this function becomes large, making the c value near zero. In rest of the image area, the numerator is near zero giving c a very high value. This function satisfies all of our requirements.

2.3. Single Pixel Analogy:

Because of two dimensional quantities and their large sizes the heat equation is still difficult to understand. Here presenting a Single Pixel Image Analogy.

Consider the image with single pixel in it. Let it be f. The size of this image is $[1 \times 1]$. We will operate ∇ on this image. The result obtained is $\begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$. The size of this matrix is $[2 \times 1]$.
Since $c(x,y)$ is a position based function and our image has a single pixel, $c(x,y)$ will just have one value. Let's consider it to be c.
After the multiplication with c, we will get $\begin{bmatrix} c\frac{\partial f}{\partial x} \\ c\frac{\partial f}{\partial y} \end{bmatrix}$.
Then we operate ∇^{T} on this result. $\nabla^{\mathrm{T}} = \begin{bmatrix} \frac{\partial}{\partial x} & \frac{\partial}{\partial y} \end{bmatrix}$ has a size $[1 \times 2]$.

Ultimately we will get a single pixel image which will have a value

$$f_{new} = \frac{\partial}{\partial x}\left(c\frac{\partial f}{\partial x}\right) + \frac{\partial}{\partial y}\left(c\frac{\partial f}{\partial y}\right)$$

This value then can be multiplied with α and can be subtracted from the original f. We will reciprocate this analogy to larger sized images and define our algorithm as follows.

2.4. Algorithm:

```
$Iterate$
{
    1. $f_{input} \leftarrow input\ image$
    2. $Operate\ \nabla. \quad \rightarrow \quad f_x\ and\ f_y$
    3. $Calculate\ c\ matrix\ according\ to\ f_x\ and\ f_y\ values.$
    4. $Multiply\ c\ with\ f_x\ and\ f_y\ individually.$
    5. $Operate\ \nabla^{\mathrm{T}}. \rightarrow \frac{\partial}{\partial \mathrm{x}}(cf_x) + \frac{\partial}{\partial \mathrm{y}}(cf_y) \rightarrow f_1$
    6. $Multiply\ by\ the\ rate\ factor\ which\ is\ \alpha.\ //usu.taken\ as\ 0.01 \rightarrow \alpha f_1 \rightarrow f_2$
    7. $Add\ this\ image\ to\ the\ original. \rightarrow f_{input} + f_2 \rightarrow f_{new}$
    8. $Assign\ input\ image\ as\ newly\ obtained\ image.\ f_{input} \leftarrow f_{new}$
}
```

2.5. Importance of α and τ:

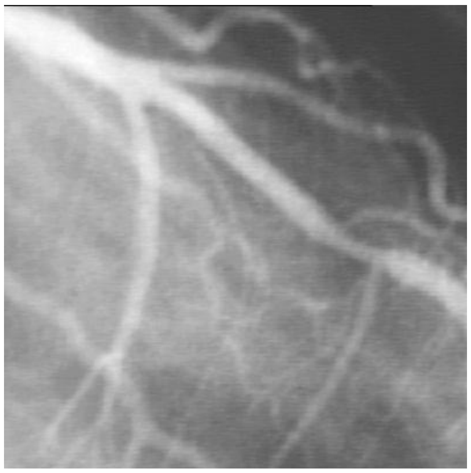
α basically decides the rate of change between the smoothened image and original image. As the value of α increases, the amount of blurring increases and vice versa. It can be intuitively seen that, we are multiplying the obtained image by α and then adding to the original image. More the quantity we add, more the image will deviate from the original one. α is hence used for controlling the rate of blurring by achieving precision.

τ is present in the denominator of the function with the help of which we calculate conductance. It's a scaling factor which is used for scaling the exponential value.

We will see the effects of changes made to these values on VCD operation in section 4.

3. Results

We will see the process outputs one by one. Let’s run this algorithm just for one iteration.
$\alpha = 0.01 \ and \ \tau = 128$



Input Image

We now take the derivative of this image in both directions.

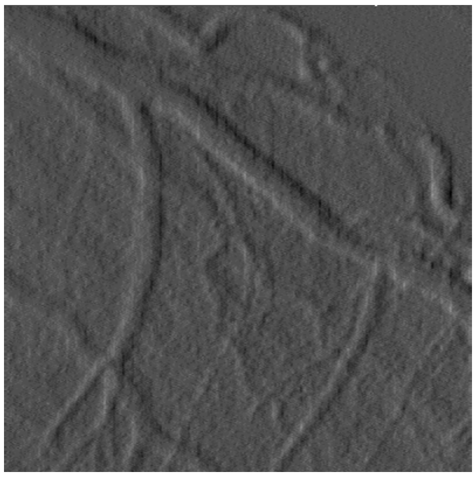


Image Derivative in x-direction

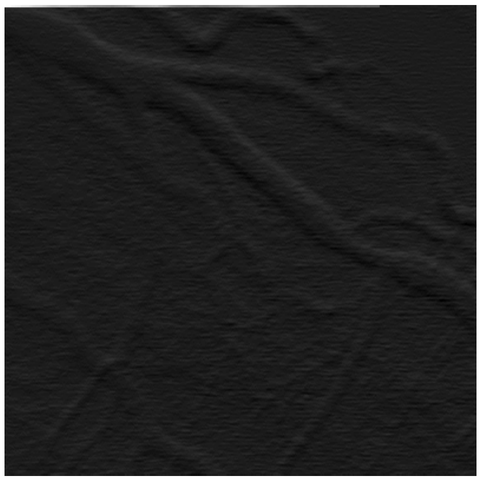
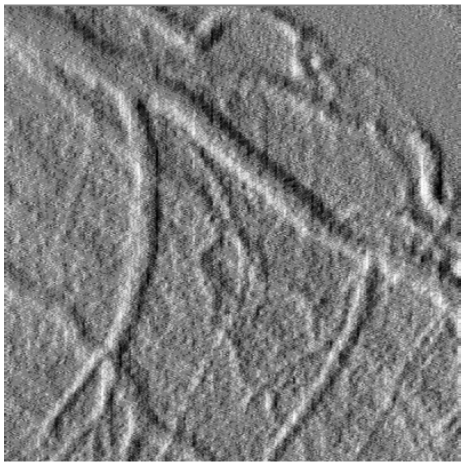
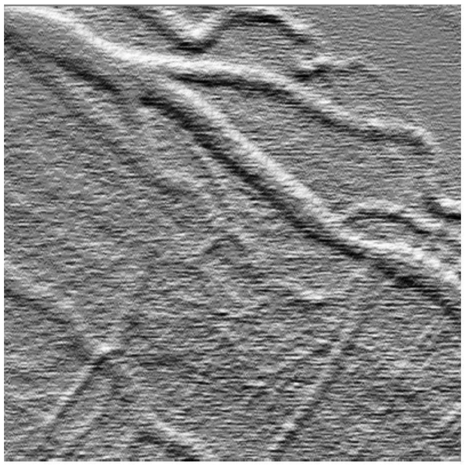


Image Derivative in y-direction

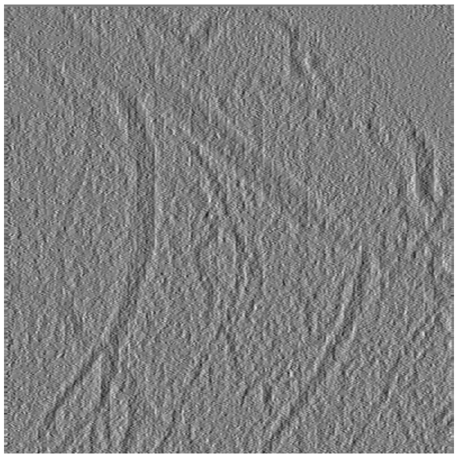
You can clearly see the vertical edges in the x-direction derivative. And the y-direction derivative is mostly black as there are almost no edges in horizontal direction.

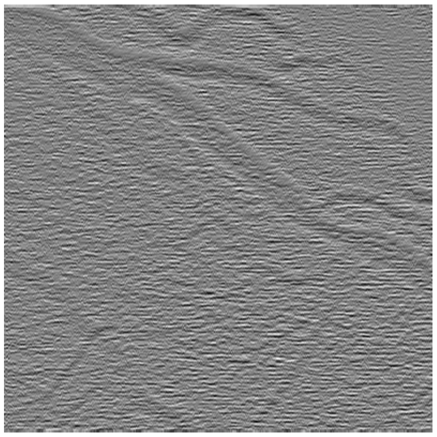
We will calculate the conductance matrix from these images. And multiply that matrix with these images to obtain two new images.


x-derivative multiplied by conductance

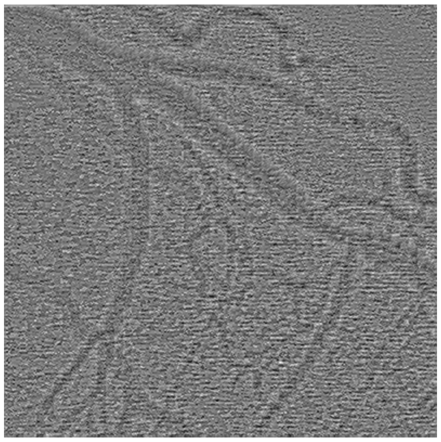

y-derivative multiplied by conductance

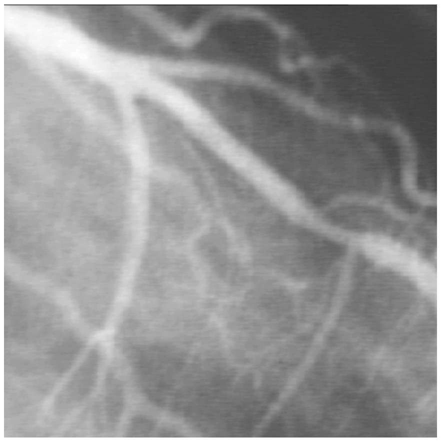
You can clearly distinguish edges from these images. Then we will again take the derivative of these images.


x-direction derivative

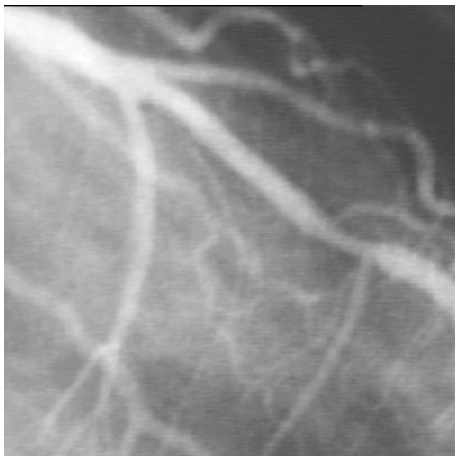

y-direction derivative

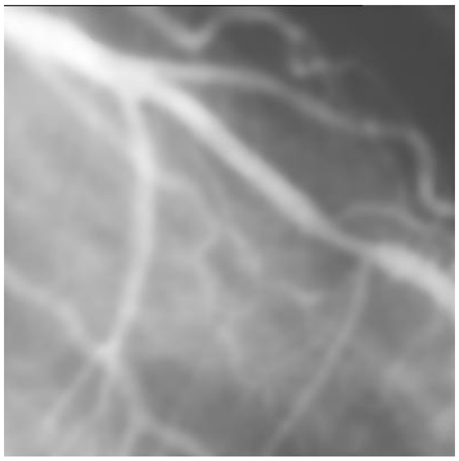
We will add these two images now. Multiply it by α. Then we will add the obtained image to the original image to obtain our blurred image. In the above two images, you can see the edges.


Addition and Multiplication by α


Smoothened Image 1 iteration

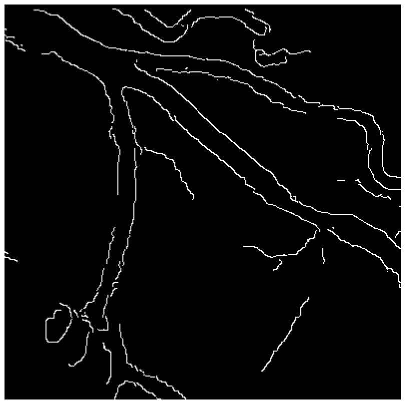
We cannot see much of change in input and output currently as we have run the algorithm only once. Let's us run this algorithm for 700 runs.


Input Image


Output Image

Now we can see the smooth image. The edges are also pretty much distinguishable. We will perform the edge detection just to see whether we have preserved the edges or not. We will use the canny edge detection method. I am using MATLAB edge detection function.

```
Image = imread('angio.jpg');
Image = rgb2gray('Image');
Output = edge(Image,'Canny');
imshow(Output);
```



Canny Edge Detection

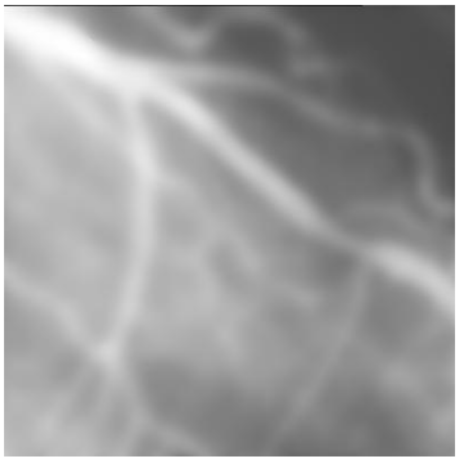
Edge detection works fine in this case. So 700 number of iterations for particular values of α and τ are optimum.

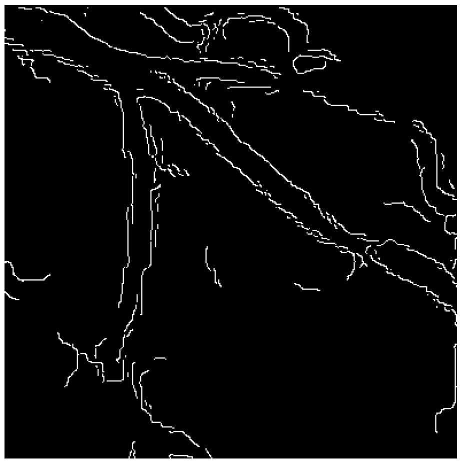
4. Extra Work

Now let us vary some parameters and analyze the outputs obtained after each variation. We will focus on change in number of iterations, changes in the values of α and τ and their corresponding outputs.

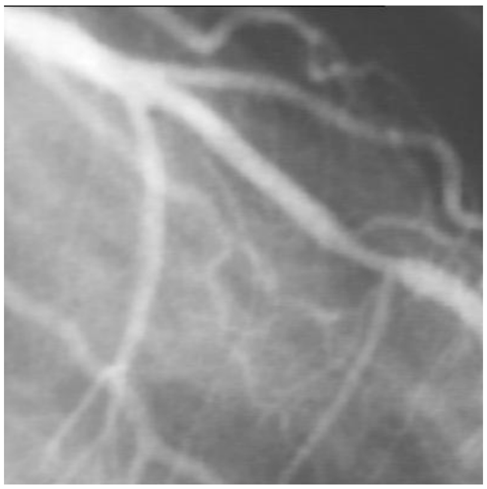
4.1. Changing the number of iterations:

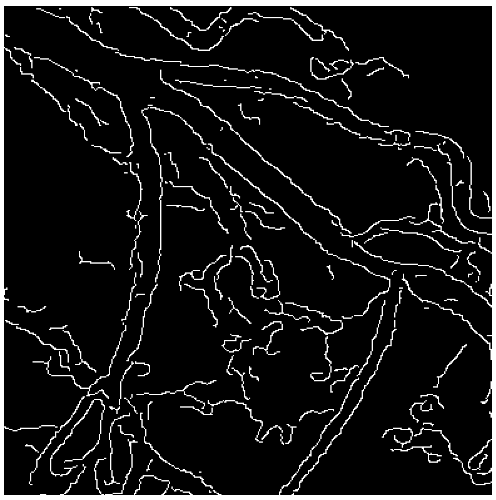
$(\alpha = 0.01 \; and \; \tau = 128)$





No of iterations = 2000



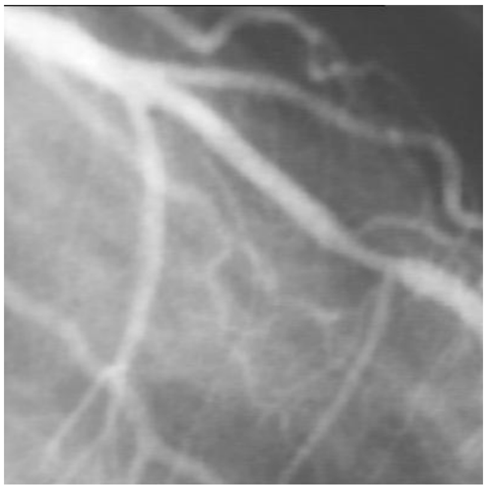


No of iterations = 100

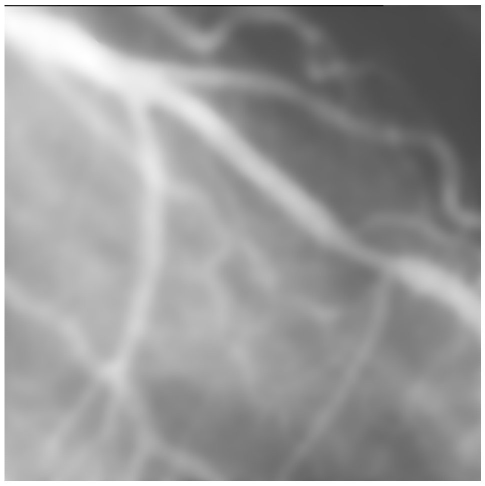
We can see that with same $\alpha \; and \; \tau$, as we increase the number of iterations, blurring increases and edge contents starts reducing and vice versa. We have to choose optimum number of iterations.

4.2. Changing α:

$(Number\ of\ iterations = 100\ and\ \tau = 128)$

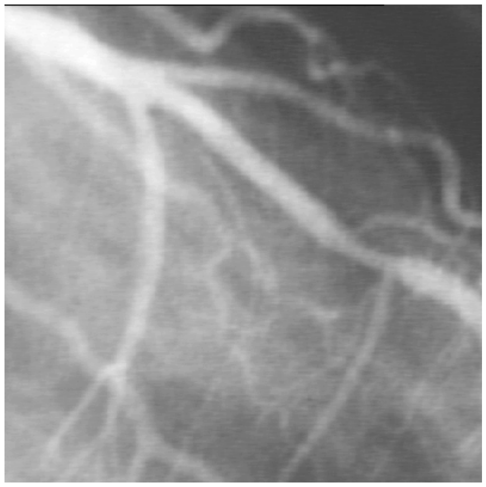


$\alpha = 0.01$



$\alpha = 0.1$

$\alpha = 1$

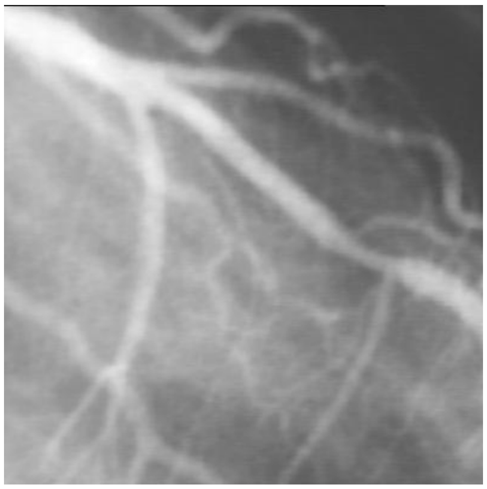


$\alpha = 0.001$

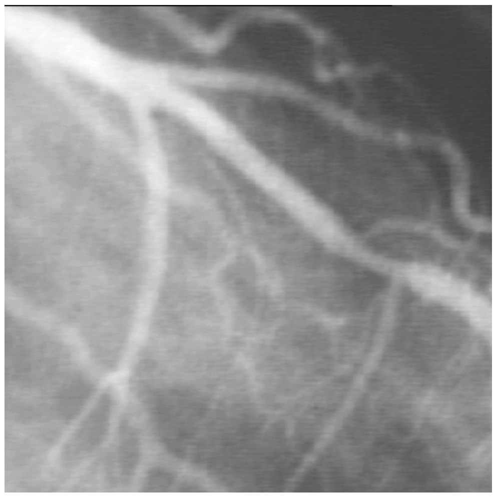
From the results above, we can observe that, as we go on increasing α, the amount of smoothness increases in the image and if we reduce α, the amount decreases. If we keep α large, we can achieve smoothing in less number of iterations but the precision and control over the amount will be difficult. Hence it is advised to keep α as small as possible usually taken as 0.01.

4.3. Changing τ:

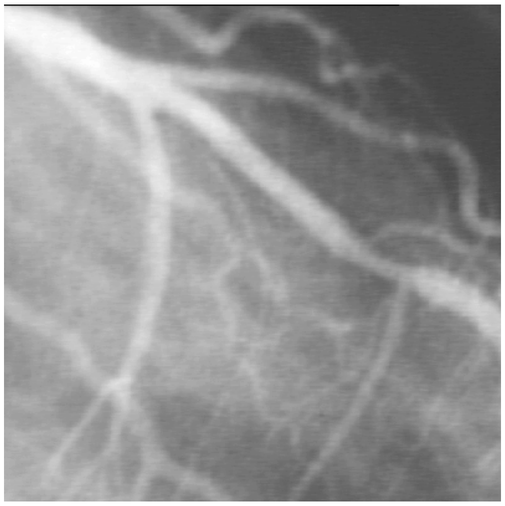
$(Number\ of\ iterations = 100\ and\ \alpha = 0.01)$



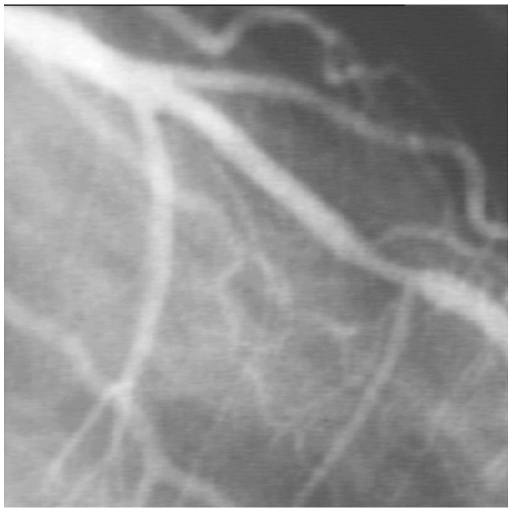
$\tau = 128$



$\tau = 1$



$\tau = 128 * 128$



$\tau = 128 * 128 * 128$

The change in τ does not affect the image as rigorously as the change in α. As we increase the value of τ, the amount of blur increases but very slightly. Since τ is used in a conductance calculation which is a **negative exponential function**, large changes in the power of τ, will also contribute very less. So we can keep τ constant. $\tau = 128$ is taken as it is a mean of 8-bit grayscale levels.

5. Conclusion

Noise can be removed using blurring or smoothing. The main drawback of smoothing, the edge loss, can be overcome using the edge preserving smoothing algorithms such as Variable Conductance Diffusion.

It is modelled after the Heat or Diffusion Equation where conductance value decides the amount of blur given at particular point in the image; since conductance is a function of space.

The blurring is done gradually in steps, so that we can have precision and control on amount of blurring.

Directional derivatives of the original image are used for the calculation of new image since derivatives give the position of edges in the image. Even conductance value is also based on the position of edges. Hence we can preserve the edges and still blur the image through this algorithm.

We performed VCD and got smooth image with distinguishable edges. The effect of changing number of iterations and values of α and τ are also observed. More number of iterations, more is the blurring keeping α and τ constant. Keeping number of iterations and τ constant, as we increase α, the smoothness increases and vice versa. Change in τ, changes the smoothness level but very gradually.

Optimum number of iterations can be found by analyzing various inputs and various outputs for different values of α and τ.

6. References

[1] Wesley Snyder, and Hairong Qi, “Fundamental Principles of Computer Vision”, Jan 2016

[2] Wesley Snyder, “Image File System Reference Manual”, Version 8.2.6, Aug 2015

[3] Dark Current, http://www.photometrics.com/resources/learningzone/darkcurrent.php

Appendix (C++ Code for VCD)

```
#include <iostream>
#include <math.h>
#include "ifs.h"
#include "flip.h"

using namespace std;


void genCondMat(float **,IFSHDR *,IFSHDR *,int,int);

IFSHDR * varConDiff(IFSHDR * image,int * len,int maxrow,int
maxcol,float **);

int main()
{
     cout << "Variable Conductance Diffusion\n";

     // Input Image and its dimensions
     IFSIMG image,image2,temp;
     image2 = ifspin((char *) "angio.ifs");
     int * len;
     int dimension,maxrow,maxcol,row,col,iteration = 0;
     float value;
     len = ifssiz(image2);
     dimension = len[0];
     maxrow = len[1];
     maxcol = len[2];

     // Converting image to float

     image = ifscreate((char *)"float",len,IFS_CR_ALL,0);
     for(row=0;row<maxrow;row++)
     {
          for(col=0;col<maxcol;col++)
          {
               value = ifsfgp(image2,row,col);
               ifsfpp(image,row,col,value);
          }
     }
     ifspot(image,(char *)"image_float.ifs");

     temp = image;

     // Initialize C Matrix

     float ** condMat = new float*[maxrow];
     for(int i = 0; i < maxrow; i++)
       condMat[i] = new float[maxcol];

     while(iteration < 700)
     {
          temp = varConDiff(temp,len,maxrow,maxcol,condMat);
          cout << iteration <<endl;
          iteration ++;
     }

     ifspot(temp,(char *)"smooth.ifs");


     return(1);
}


void genCondMat(float ** condMat,IFSHDR * im_dx,IFSHDR *
im_dy,int maxrow,int maxcol)
{
  int row,col;
  float value,tou=128;

  for(row=0;row<maxrow;row++)
    {
      for(col=0;col<maxcol;col++)
     {
       value = exp(-
((ifsfgp(im_dx,row,col)*ifsfgp(im_dx,row,col)) +
(ifsfgp(im_dy,row,col)*ifsfgp(im_dy,row,col)))/tou);
       condMat[row][col] = value;
     }

    }
}

IFSHDR * varConDiff(IFSHDR * image,int * len,int maxrow,int
maxcol,float ** condMat)
{

     IFSIMG im_dx,im_dy,cimdx,cimdy,fdx,fdy,fnew;
     int devout,row,col;
     float value,alpha=0.01;

     im_dx  = ifscreate((char *)"float",len,IFS_CR_ALL,0);
     fldx(image,im_dx);

     im_dy  = ifscreate((char *)"float",len,IFS_CR_ALL,0);
```

```
	fldy(image,im_dy);

	genCondMat(condMat,im_dx,im_dy,maxrow,maxcol);

	// Multiplication by C

	cimdx  = ifscreate((char *)"float",len,IFS_CR_ALL,0);
	cimdy  = ifscreate((char *)"float",len,IFS_CR_ALL,0);

	for(row=0;row<maxrow;row++)
	{
		for(col=0;col<maxcol;col++)
		{
			value = ifsfgp(im_dx,row,col)*condMat[row][col];
			ifsfpp(cimdx,row,col,value);
			value = ifsfgp(im_dy,row,col)*condMat[row][col];
			ifsfpp(cimdy,row,col,value);
		}
	}

	ifsfree(im_dx,IFS_FR_ALL);
	ifsfree(im_dy,IFS_FR_ALL);


	// Next Del Operator

	fdx  = ifscreate((char *)"float",len,IFS_CR_ALL,0);
	fldx(cimdx,fdx);

	fdy  = ifscreate((char *)"float",len,IFS_CR_ALL,0);
	fldy(cimdy,fdy);

	ifsfree(cimdx,IFS_FR_ALL);
	ifsfree(cimdy,IFS_FR_ALL);

	fnew = ifscreate((char *)"float",len,IFS_CR_ALL,0);
	for(row=0;row<maxrow;row++)
	{
		for(col=0;col<maxcol;col++)
		{
			value = ifsfgp(fdx,row,col) +
ifsfgp(fdy,row,col);
			value = alpha*value;
			value = value + ifsfgp(image,row,col);
			ifsfpp(fnew,row,col,value);
		}
	}
```

```
     ifsfree(fdx,IFS_FR_ALL);
     ifsfree(fdy,IFS_FR_ALL);

     ifsfree(image,IFS_FR_ALL);

     return(fnew);
}
```