Adaptive Difference of Gaussian Algorithm for Coherent Line Drawing

Abadi Kurniawan

Abstract

Non-photorealistic rendering is a method of imitating hand-drawn images using computer as the tool. One of hand-drawn techniques used by artists is line drawing. This goal of this thesis is to produce a technique to create line drawing images from photographs. Based on a previous technique called Flow-based Difference of Gaussian (FDoG), we try to improve the output image so that it will create more believable pictures, as it if were hand-drawn by an artist.

FDoG has proven to be able produce images with coherent and continues lines, but it fails to capture coarse details on isotropic areas in the image. Another technique in line drawing called Difference of Gaussian (DoG), which FDoG was based on, can produce better detail on isotropic areas. Combining these two techniques can create better results for both isotropic and anisotropic areas. We create an image segmentation technique using polarity to divide isotropic and anisotropic areas in the image. Using this segment, we then adaptively apply FDoG and DoG filters to each segment.

1. Introduction

Over the last 30 years, computer graphics researchers have tried to achieve photorealistic images by using computer programs [17]. Ivan Sutherland was one of the pioneers in creating computer generated photorealistic images, working in the early 1960s. Photorealistic computer graphics is essentially an attempt to imitate real life pictures that would be normally taken by using another tools such as camera. Such attempts usually occurs in 3 dimensional (3D) computer graphics, which in most cases, try to render the 3D model as realistically as possible. At the early stage of computer graphics research, these attempts has been failed in many cases, especially on the attempts to imitate human appearance. Recent research, however, has started to show more believable lifelike human facial animation [11]. More recently, instead of producing results that look like real life objects, researchers have tried to produce more cartoon-looking or hand-drawn images; this technique is known as Non-photorealistic Rendering.

Non-Photorealistic Rendering (NPR) is a process which uses a computer program to produce non-photorealistic renditions [17], such as sketch illustration, oil painting, watercolor-painting, etc. While using photorealistic images or even real photographs may deliver accurate information to the viewers, they also possess some weaknesses, which can be surpassed by hand-drawn images. In some cases, hand-drawn images can actually delivers more information because they can allocate more focus on a certain aspect of a picture while ignoring unimportant ones. This way, the viewers will focus their attention on what the picture tried to convey. These kinds of images can be found in technical illustrations, for example in user manual of some electronic devices, or can also be found in medical illustrations.

Hand-drawn images can also be used to show the preliminary stage of a design in design phase period. For example, in architectural design [13], architects usually use pencil sketch drawings to show a draft of a design. The viewer can get the feeling of this preliminary stage better when seeing a hand-drawn pencil sketch image compared to a photorealistic image.

Another benefit of using hand-drawn images is to give aesthetic feeling to an image. Images, such as paintings created by an artist, usually have aesthetic values that can stimulate certain emotions in humans. While the paintings themselves might not look as realistic as a photograph, they usually can stimulate senses and emotions in the people who see it.

The goal of NPR is to try to achieve the same benefits of using hand-drawn images in cases when an actual hand-drawn image is very difficult or impossible to create. The source of image generation on NPR can come from either 3 dimensional model or photographs taken by camera.

Since the first time NPR was introduced, there have been many researches to work on the topic. Many of them have created techniques of imitating different styles of hand-drawing period. For example: sketch rendering [16], pen-and-ink illustration [18],[12], stipple drawing [14],[4], painterly effect [9],[7],[6], and line drawing [3], [15],[8].

2. Line Drawing

Line drawing is a drawing style that is commonly used to create illustration, such as technical illustrations and medical illustrations, caricature drawings, sketch drawings, etc. Line drawing uses straight or curved lines to form an image. Some techniques, such as stippling or hatching, can be used to show different levels of shade. Line drawing has become one of the primary branches of NPR. Techniques have been improving in creating more believable results compared to line drawing illustration made by an artist. Line drawing in NPR can be divided into two categories based on the source of the image, which can be a 3D object or a 2D object. Many works have been introduced on line drawing based on 3D object [3], [15]. Creating line drawing images from 3D models is relatively easier compared to 2D objects. In 3D models, the edge can be easily identified because all the information is stored in a straightforward manner, while in 2D objects, such as in photographs, information about edges are more difficult to extract and require extra steps and complicated methods to achieve a good result.

Most work in line drawing in NPR is based on edge detection or image segmentation techniques. The difference between edge detection and line drawing is that edge detection is more focused on how to extract edges from a picture without considering the aesthetic aspect of how the lines (edges) are compounded. While line drawing in NPR is also trying to extract edges from pictures and form lines, it attempts to create a more artistic result, as if the image were created by human artist. Some aspects that need to be considered to produce artistic results are: the structures of the lines and the use of stylistic edges. There are many methods to do edge detection, two examples of them are Canny's method [2], and Marr-Hildreth's method [10],

which became the base for Difference-of-Gaussian (DoG) filter and used by Gooch et al. in their facial illustration system to create some artistic line drawing.

3. Difference of Gaussian (DoG)

Gooch et al. in their facial illustration system [5] were able to create some artistic results. They were able generate a caricature of a human face based on a photograph. Their method uses DoG filter, which was based on Marr-Hildreth's edge detection.

The basic idea of DoG is to apply 2 Gaussian filters with different blur levels (σ). The second Gaussian filter uses a bigger blur radius to create blurrier image compared to the image produced by the first Gaussian filter. The Area close to edges will have more changes compared to area far from edges if applied to Gaussian filter with different blur level. The difference of intensity of each pixel of these two blurred images will be use by DoG to obtain edges, as shown in Figure 1.

(a) Input image

(b) Gaussian with $\sigma = 1.0$

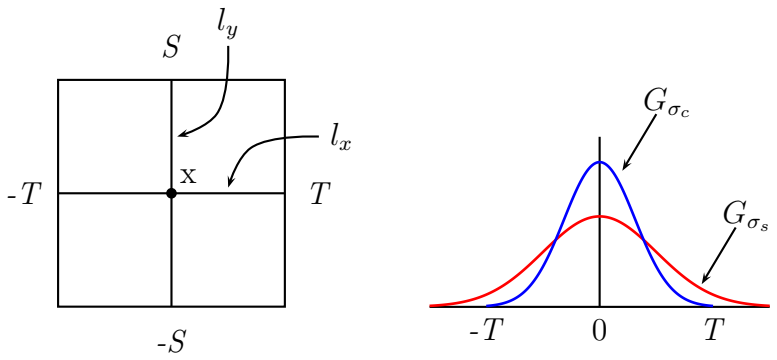
(c) Gaussian with $\sigma = 1.6$

Figure 1: Sample of gaussian filter

The Gaussian function used in DoG is formulated as the following 1-dimensional Gaussian function:

$$G(i, \sigma) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{i^2}{2\sigma^2}} \tag{1}$$

where σ is the blur level. $G(i, \sigma)$ is used as weight factor on how intensity of each pixel and its neighbor will influence intensity of the output, thus creating a blur image.



(a) DoG Kernel (b) Gaussian components for DoG

Figure 2: Difference of Gaussian filtering

This weight factor applied with σ_c for each intensity of pixels along line l_x started from $-T$ to T:

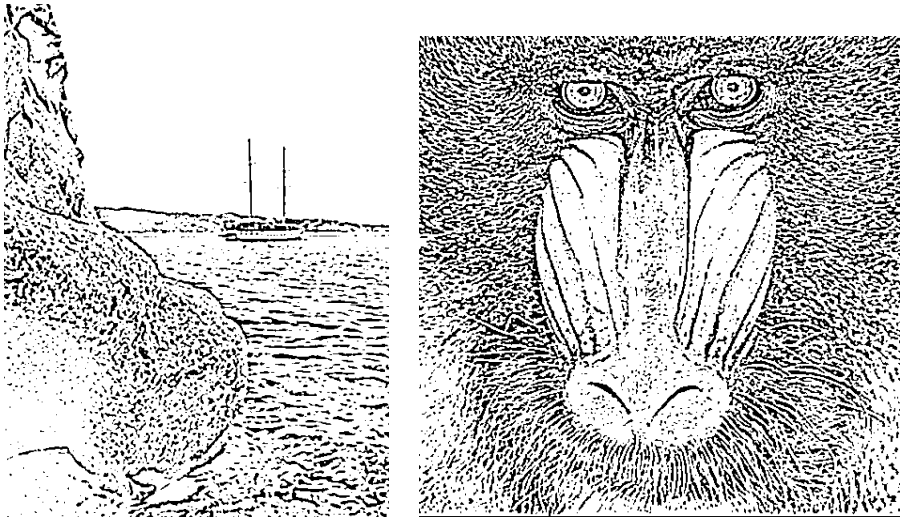
$$f(t, \sigma) = \int_{-T}^{T} G(t, \sigma) I(l_x(t))\, dt \tag{2}$$

and using the value from $f(t, \sigma)$, we use We then apply the same function along line l_y starting from -S to S, and repeat this steps for every pixel on input image to produce a blurred image.

$$F(\mathrm{x}, \sigma) = \int_{-S}^{S} G(s, \sigma) f(l_y(s), \sigma)\, ds \tag{3}$$

The same procedure with $\sigma_s = 1.6\sigma_c$ is then applied to the same input image, producing another blurred image. From these 2 output images, we then able to calculate differences of each pixel's intensity, and capture edges of the input image (Figure 3).

$$H_d(\mathrm{x}) = F(\mathrm{x}, \sigma_c) - \rho \cdot F(\mathrm{x}, \sigma_s) \tag{4}$$



(a) Rock and Boat (b) Baboon

Figure 3: Difference-of-Gaussian

4. Flow-based Difference of Gaussian

Difference-of-gaussian (DoG) method has the nature of isotropic because the filter moves to every direction in order to calculate the differences. This nature, when applied to images with anisotropic characteristic, will create a lot of discontinuity lines because it failed to follow the direction alongside the edges. So, instead of moving to every direction, Kang et al. are suggesting to incorporate direction guide to the filter that follows the direction of edges [8].

To create the direction guidance for DoG filter, Kang et al., use the information from edge flow, which is where an area of image has high contrast of intensity level, and run the DoG filter perpendicular to the edge flow direction. By providing guide for the filter direction, Kang et al., were able to produced more coherent and continues line drawing [8].

4.1. Edge Tangent Flow

Kang et al., technique in creating line drawing required two steps. First step is to generate edge tangent flow that will provide direction to run DoG filter. To create edge tangent flow from input image $I(\mathrm{x})$, where $\mathrm{x} = (x, y)$

denotes a pixel at coordinate (x, y), Kang et al., use edge tangent t(x) as a vector perpendicular to the image gradient $g(\mathrm{x}) = \nabla I(\mathrm{x})$, and called it Edge Tangent Flow (ETF).

They presented a technique to construct ETF by using kernel-based nonlinear smoothing of vector field. The ETF construction filter is as follows:

$$\mathrm{t}^{new}(\mathrm{x}) = \frac{1}{k} \sum_{\mathrm{x}' \in \Omega(\mathrm{x})} \phi(\mathrm{x}, \mathrm{x}')\mathrm{t}^{cur}(\mathrm{y})w_s(\mathrm{x}, \mathrm{x}')w_m(\mathrm{x}, \mathrm{x}')w_d(\mathrm{x}, \mathrm{x}') \tag{5}$$

where k is the vector normalizing term and $\Omega(\mathrm{x})$ is the neighbor of x. ETF function uses 3 weight function, the first one is w_s, a *spatial weight function*, which uses radially-symmetrical box filter with radius of r, where r is the radius of kernel Ω. The *spatial weight function* w_s is defined as following:

$$w_s(\mathrm{x}, \mathrm{x}') = \begin{cases} 1 & \text{if } \|\mathrm{x} - \mathrm{x}'\| < r \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

The second weight function is w_m, a *magnitude weight function*, which is defined as following:

$$w_m(\mathrm{x}, \mathrm{x}') = \frac{1}{2}(1 + \tanh[\eta \cdot (\hat{g}(\mathrm{x}') - \hat{g}(\mathrm{x}))]) \tag{7}$$

where $\hat{g}(\mathrm{x})$ and $\hat{g}(\mathrm{x}')$ denotes normalized gradient magnitude at x and x′ respectively. Variable η is the weight variable that determine how the neighboring pixel x′ with gradient value higher than gradient value of x will influence the edge tangent t(x).

The third weight function is the direction weight function w_d, which is defined as following:

$$w_d(\mathrm{x}, \mathrm{x}') = |\mathrm{t}^{cur}(\mathrm{x}) \cdot \mathrm{t}^{cur}(\mathrm{x}')| \tag{8}$$

where $\mathrm{t}^{cur}(\mathrm{x})$ and $\mathrm{t}^{cur}(\mathrm{x})$ denote the current normalized tangent vector at x and x′ respectively. This function is using a dot product between tangent vectors at a particular location with its neighbors'. The cross product produce a high weight value for vectors that are closely aligned (where angle between two vectors (θ) is close to 0° or 180°) and produce a low weight value for vectors that are perpendicular to each other (where angle between two vector (θ) is close to 90°).

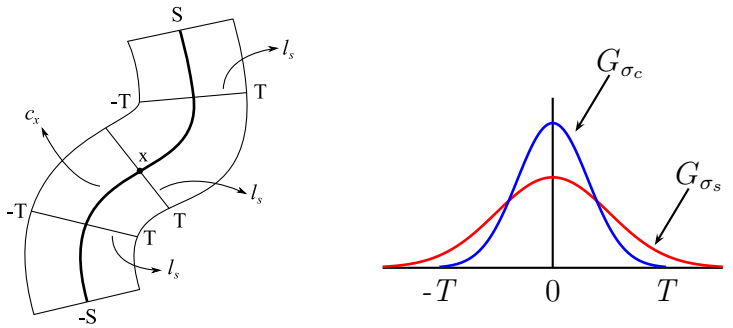
The last function is used to reverse the direction of $\mathrm{t}^{cur}(\mathrm{x}')$ before smoothing when angle between vectors (θ) is bigger than 90° (dot product between two vectors is ¡ 0):

$$\phi(\mathrm{x}, \mathrm{x}') = \begin{cases} 1 & \text{if } \mathrm{t}^{cur}(\mathrm{x}) \cdot \mathrm{t}^{cur}(\mathrm{x}') > 0 \\ -1 & \text{otherwise} \end{cases} \tag{9}$$

As shown in the function, $\mathrm{t}^{new}(\mathrm{x})$ is achieved from applying all functions to $t^{cur}(\mathrm{x})$, and initially, $\mathrm{t}^{\circ}(\mathrm{x})$ is achieved from normalized vector perpendicular to gradient $g(\mathrm{x})$ in gradient map of the input image I. ETF is constructed by iteratively apply $\mathrm{t}(\mathrm{x})$ function several times (in this experiment, we applied 2 or 3 times).

4.2. Flow-based DoG

After constructing ETF, the next step will be applying anisotropic DoG with edge flow information from ETF to guide its direction. We build kernel whose shape is based on local flow from ETF, and apply DoG on this kernel. Examples of kernel is shown in Figure 4. Line c_x is a line that follows direction of flow from ETF, and line l_s is a line tangent to line c_x.



(a) FDoG Kernel (b) Gaussian components for DoG

Figure 4: Flow-based DoG Kernel

We first apply 1 dimensional filter $f(t)$ along the line l_s:

$$F(s) = \int_{-T}^{T} I(l_s(t)) f(t)\, dt \tag{10}$$

where $l_s(t)$ denotes a pixel on the line l_s at parameter t, and $I(l_s(t))$ is intensity level of the pixel. Function $f(t)$ is the same DoG function from previous chapter that use 2 gaussian filters with difference σ and then calculate differences from 2 output values.

$$f(t) = G_{\sigma_c}(t) - \rho \cdot G_{\sigma_s}(t) \tag{11}$$

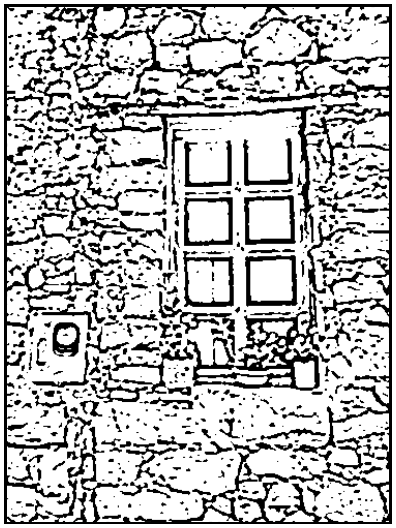
ρ value control the level of noise detected, and in this experiment, we use value of 0.99.

After applying filter $f(t)$ to each pixel, we then apply filter $F(t)$ along line c_x from $-S$ to S on FDoG kernel.

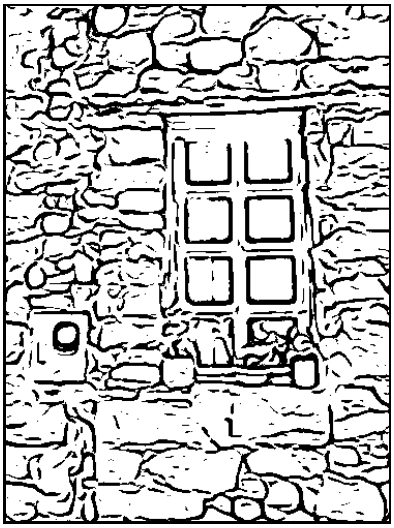
$$H_f(\mathrm{x}) = \int_{-S}^{S} G_{\sigma_m}(s) F(s)\, ds \tag{12}$$

where G_{σ_m} is another gaussian function that will become weight factor for each responses according to s. As shown in Figure 5, image from FDoG () has continuous lines and less noise compared to DoG.

(a) Input image



(b) DoG

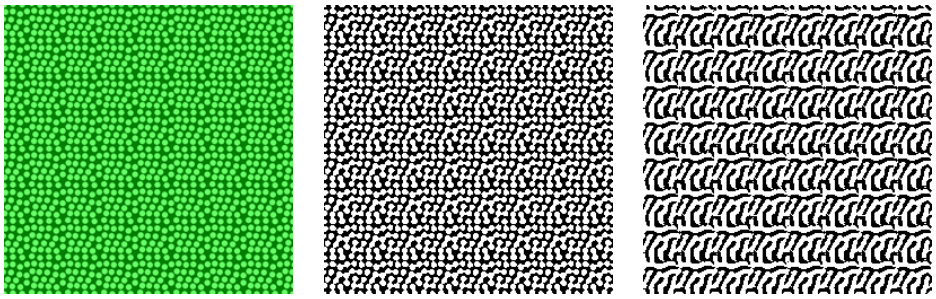


(c) FDoG

Figure 5: Difference-of-Gaussian

Hybrid Base on the output, images created by FDoG contained more coherent and continuous lines compared to images created by DoG. The directional kernel used in FDoG was proven to be able to create lines on anisotropic images, and created good line drawing images. One major draw-

back of FDoG is it cannot capture all the detail on images with isotropic characteristic. Example on Figure 6 shows that FDoG failed to capture the details of grained textures, furthermore creating lines in area where it is not supposed to be. While DoG can easily cope with this flaw, it still cannot outmatch FDoG in cases of anisotropic images. These two different characteristics of FDoG and DoG on handling isotropic and anisotropic images bring us to the idea of combining to method in order to achieve optimal results in both cases.



(a) Input Image (b) DoG (c) FDoG

Figure 6: Weakness of FDoG

5. Combining FDoG with DoG

The simple method to combine DoG and FDoG is by applying linear combination of FDoG and DoG functions, and called it Hybrid DoG:

$$H_h(\mathrm{x}) = \lambda H_f(\mathrm{x}) + (1 - \lambda) H_d(\mathrm{x}) \tag{13}$$

Value of Λ will determine the closeness of the output to FDoG or DoG, when the value of Λ approaches 0, it will produce output that similar to DoG, and will produce output similar to FDoG when it approaches 1.

With this variable λ, we can control the behavior of the filter according to the character of input images. Use value of λ close to 0 for input images with majority of isotropic images, and use value of λ close to 1 for anisotropic images. For images with both isotropic and anisotropic characteristic, based on our experiment, values of 0.5 for λ will produce good results for most of the images.

Adaptive-DoG Technique Another method to combine DoG and FDoG is to adaptively apply the right filter for different area depending on their characteristic. Area with isotropic textures will use DoG to retain the texture, while area with anisotropic detail will use FDoG to create more coherent lines and low noise details. To be able to adaptively apply the right filter, we first need to be able to distinguish between isotropic and anisotropic segment. Belongie et al., [1] in their paper use polarity to segment the image based on texture type. We simplify the technique used by Belongie et al.

6. Image Segmentation

Polarity is calculated by comparing the gradient's direction of a pixel to its neighbor's gradients. Area with similar gradient vector direction considered as high polarity area, while area with non-uniform gradient vector direction is considered as low polarity area. Non-uniform gradient vector direction is can be found in isotropic area the gradients in this area scattered. Anisotropic area will have more uniform and similar direction, although it may also has have gradients facing opposite to each other as we can see on area with stripe texture. Using this behavior, we can distinguish between isotropic and anisotropic area by determining polarity of each pixel. One exception that need to be considered is isotropic area with low gradient magnitude should be considered as isotropic segment since this area should be applied with FDoG filter to minimize noise.

To calculate polarity, we use a function S which will be applied to pixel location x:

$$S(\mathrm{x}) = \frac{\sum_{\mathrm{x}' \in \Omega(\mathrm{x})} |\, g(\mathrm{x}) \cdot g(\mathrm{x})' \,|}{|\, \Omega(\mathrm{x}) \,|} \tag{14}$$

where $\Omega(\mathrm{x})$ is the neighborhood of pixel x and pixel x' is element of $\Omega(\mathrm{c})$. We use absolute value for dot product between two gradient vector because we consider vectors with opposite direction in the same segment. The sum of dot product is then divided by $|\, \Omega(\mathrm{x}) \,|$, the total number of pixels in the neighborhood, to get the average.

Next step will be segmenting the image using the polarity measured on previous step:

$$P(\mathrm{x}) = \begin{cases} 0 & \text{if } S(\mathrm{x}) >= \alpha \text{ or } |g(\mathrm{x})| < \beta \\ 1 & \text{otherwise} \end{cases} \tag{15}$$

We use a threshold α as a cut point for polarity level, and in our experiment, we use $\alpha = 0.875$. Area with polarity above the threshold will be considered as anisotropic segment (labeled with 1). For isotropic segment (labeled with 0), since we also consider isotropic area with low gradient magnitude as anisotropic segment, we also need to consider the gradient magnitude. Variable β is threshold for gradient magnitude at pixel x and we use $\beta = 0.1$ in our experiment.

(a) Input image

(b) Polarity-based image segment; white indicates isotropic area, black indicates anisotropic area, and gray indicates area with low gradient magnitude

Figure 7: Image Segmentation

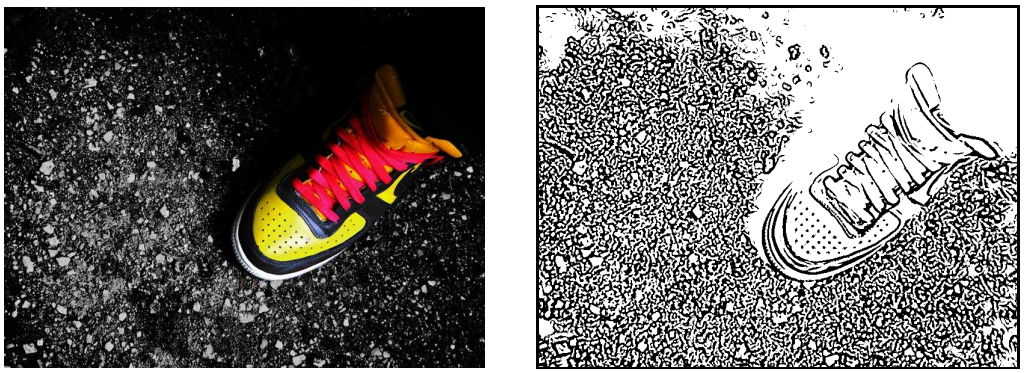
7. Adaptive-DoG

Once we obtain the image segment, the next step to adaptively apply hybrid DoG will be straightforward. Provided the characteristic of each area, we are now able to determine the proper filter for different area just by using label from the image segment.

$$H_a(\mathrm{x}) = \begin{cases} H_d(\mathrm{x}) & \text{if } P(\mathrm{x}) = 0 \\ H_d(\mathrm{x}) & \text{otherwise} \end{cases} \tag{16}$$

Segment labeled with 0 is isotropic area and use DoG filter H_d, while anisotropic area labeled with 1 is applied with FDoG filter H_f.

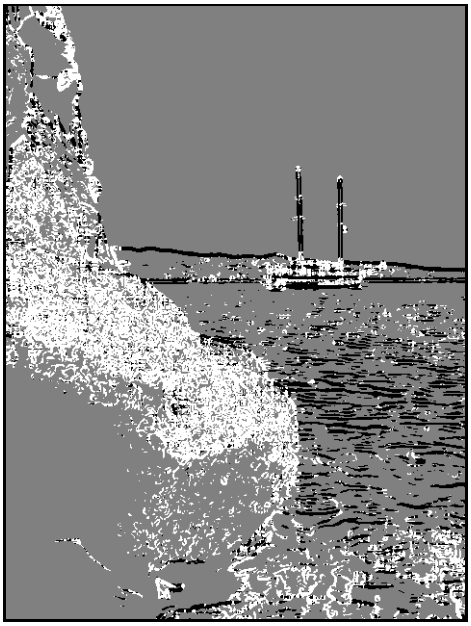
Results The results of this adaptive technique have coherent lines similar to lines in FDoG, while still maintaining the details in coarser area. The lines as shown in figure 8(b) around the shoe area have similar shape with lines from FDoG, while the surrounding textures still maintain its original shape.



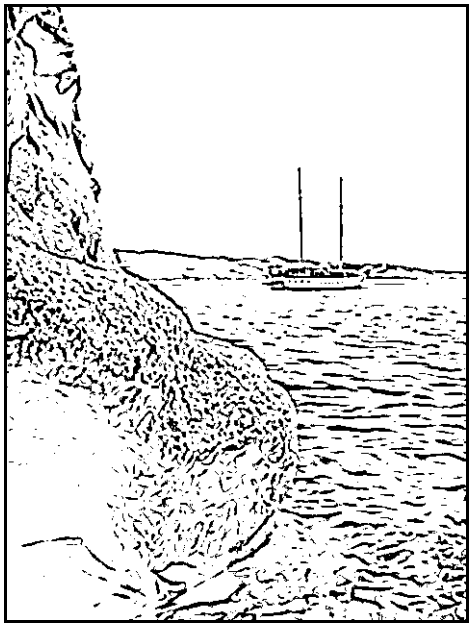
(a) Input Image (b) Adaptive-DoG

Figure 8: Adaptive-DoG

(a) Input Image

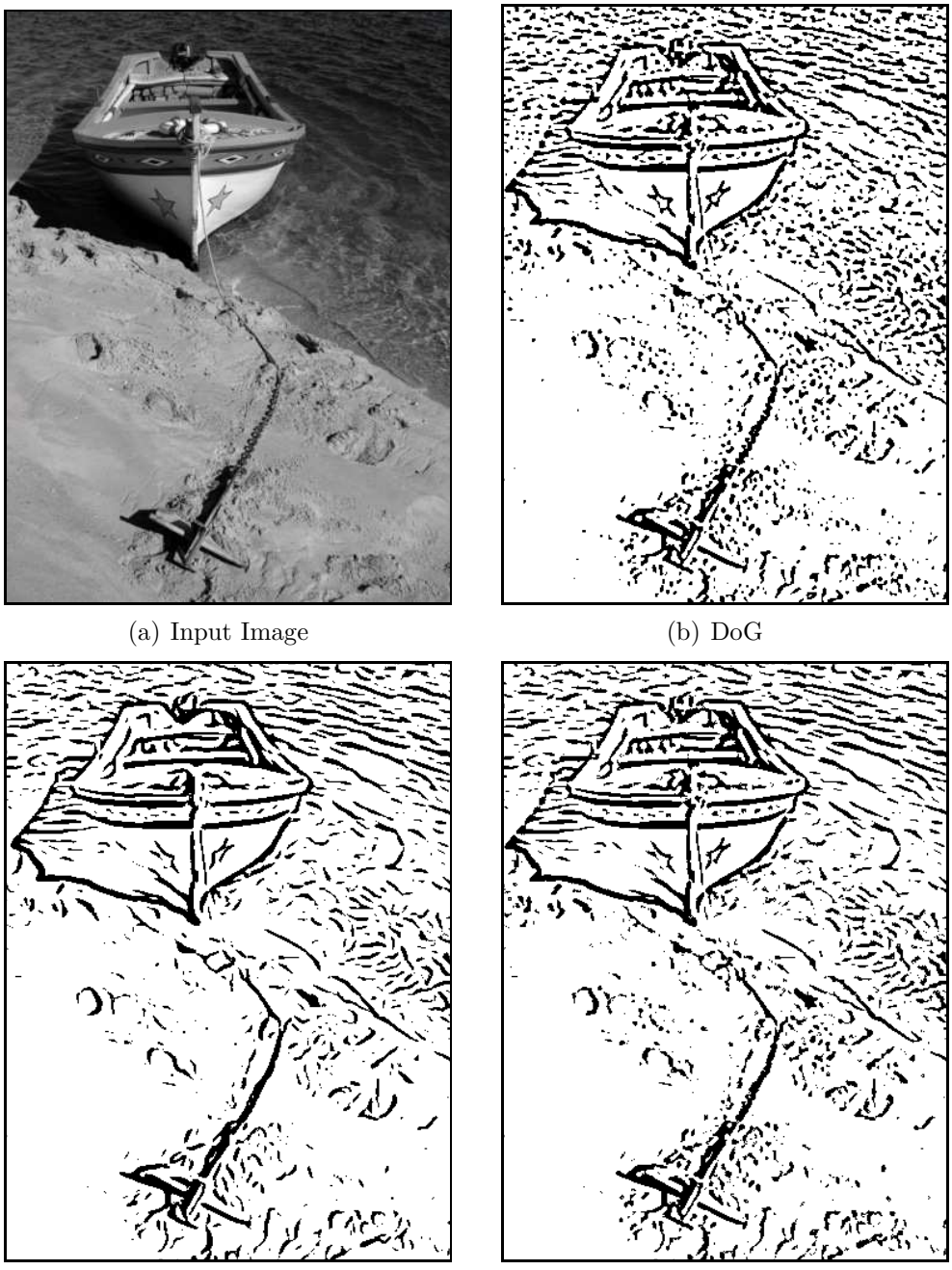


(b) Polarity Segment



(c) Adaptive-DoG

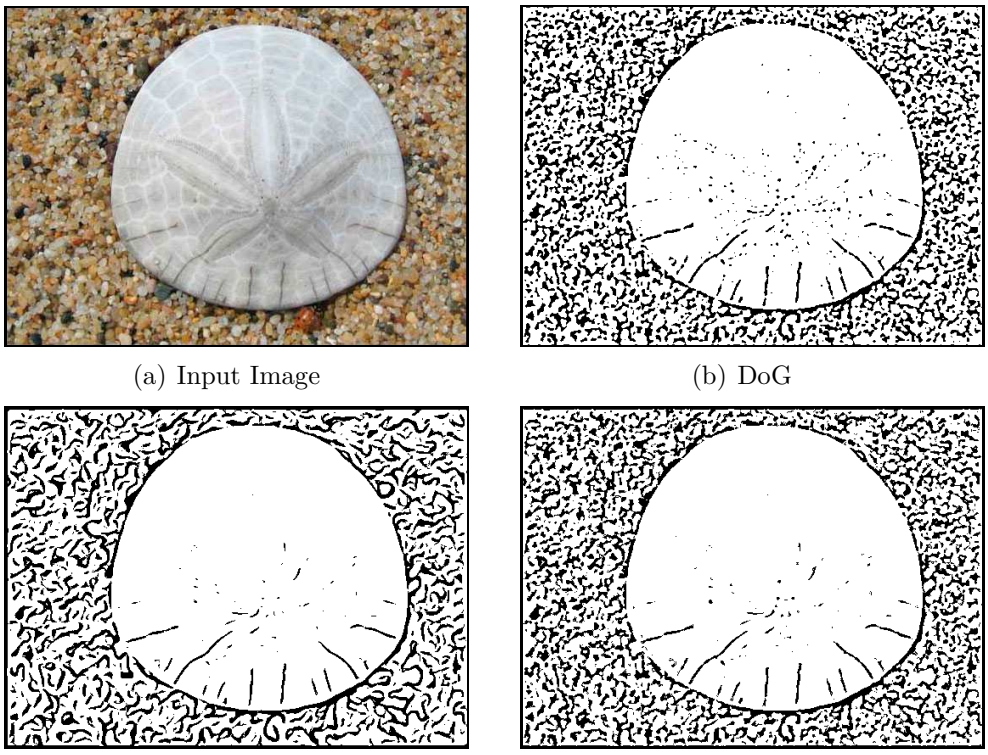
Figure 9: Boat and Rock



(a) Input Image (b) DoG

(c) FDoG (d) Adaptive-DoG

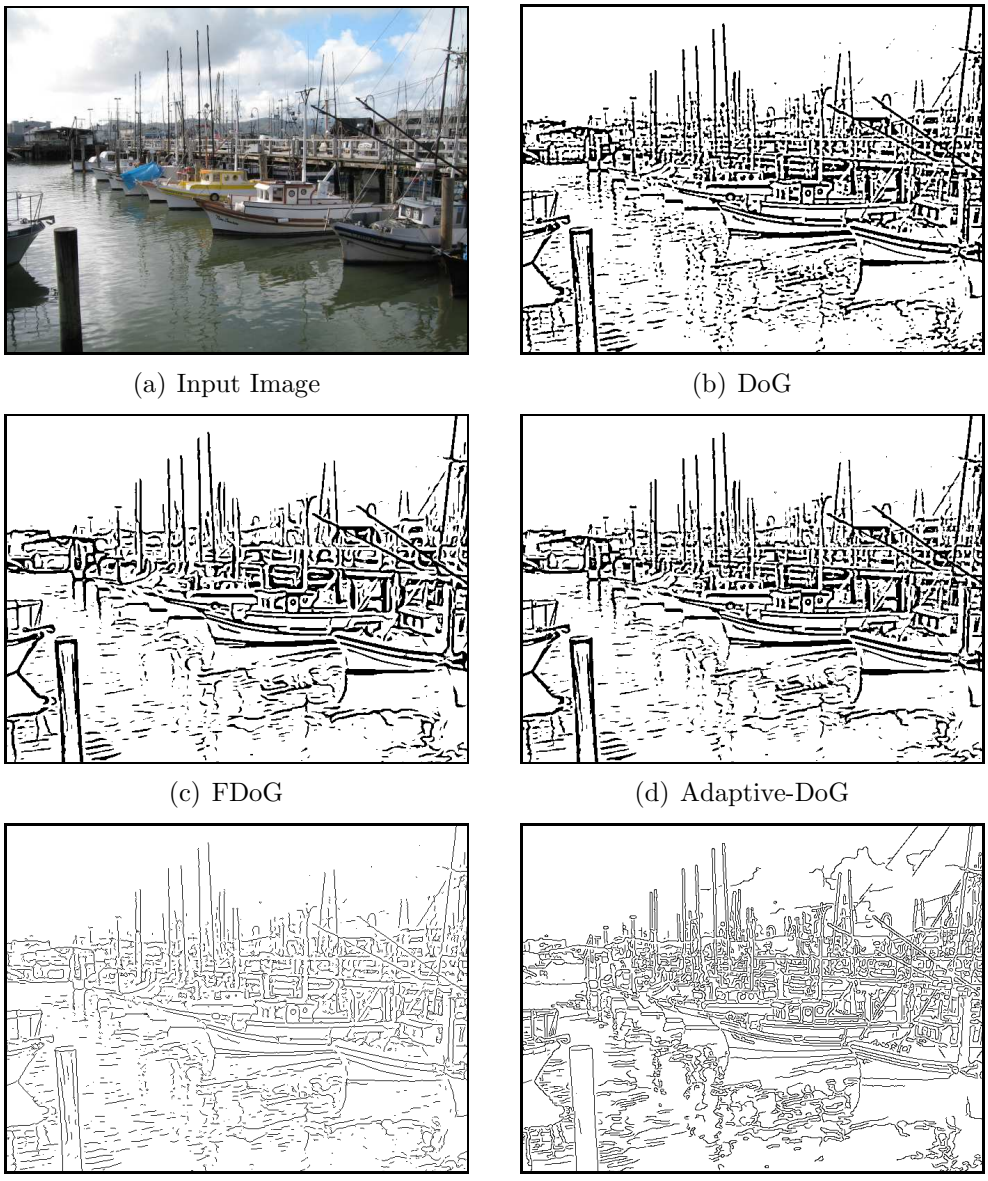
Figure 10: A Boat in Beach



(a) Input Image (b) DoG

(c) FDoG (d) Adaptive-DoG

Figure 11: Sand Dollar



(a) Input Image (b) DoG

(c) FDoG (d) Adaptive-DoG

(e) Adaptive-DoG Thin (f) Canny's Edge Detection

Figure 12: Boats at the Bay

The current implementation of our Adaptive-DoG filter is not optimized for performance yet. Currently, it runs significantly slower than DoG and

FDoG, due to the image segmentation step. Our implementation of polarity calculation requires $1344ms$ to perfoms on image with $750 * 500$ dimension with size of neigborhood at $20 * 20$ pixels. The Adaptive-DoG filter itself runs for $3390ms$ on the same image. The total run time for this technique is $7797ms$. This number is significantly higher DoG, which only requires $3859ms$, and also FDoG, which only requires $4578ms$. The main problem from our implementation is that we still apply both DoG and FDoG filters, in which both output will be selectively chosen based on image segment. Further improvement of this implementation is required to make the program run faster and more efficient.

Conclusion Our Adaptive-DoG technique, regardless the performance in term of speed, is able to produce better results when compared to DoG and FDoG in many different variaty of images in our experiment. The lines appear to be more continuous than lines in DoG images, and the details in coarse texture is better maintain than FDoG images.

Although the output is relatively better in some cases, the results of Adaptive-DoG may never exceed the results of neither DoG nor FDoG in some specific cases. For input image that only contains lines without any coarse texture, the results of Adaptive-DoG will, at best, be the same with FDoG, while for input image with coarse texture only, the results will never exceed DoG.

Another note from experiment experiment that we need mention is the performance in term of runtime. Our experiment shows that the performance of the filter is relatively lower than both preceding filters. Nevertheles, with more code optimization and tweaking, some improvements are possible to be achieved.

Possible future improvement for Adaptive-DoG is by iterating the filters to prouduce more solid shape for both lines and textures. The same polarity map may be use on the next iteration, but generating new polarity map might also produce better results.

References

[1] S. Belongie, C. Carson, H. Greenspan, and J. Malik. Color- and texture-based image segmentation using em and its application to content-based image retrieval. *Computer Vision, 1998. Sixth International Conference on*, pages 675–682, Jan 1998.

[2] John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-8(6):679–698, Nov. 1986.

[3] Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. Suggestive contours for conveying shape. *ACM Trans. Graph.*, 22(3):848–855, 2003.

[4] Oliver Deussen, Stefan Hiller, Cornelius Van Overveld, and Thomas Strothotte. Floating points: A method for computing stipple drawings. *Computer Graphics Forum*, 19:40–51, 2000.

[5] Bruce Gooch, Erik Reinhard, and Amy Gooch. Human facial illustrations: Creation and psychophysical evaluation. *ACM Trans. Graph.*, 23(1):27–44, 2004.

[6] James Hays and Irfan Essa. Image and video based painterly animation. pages 113–120, 2004.

[7] Aaron Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 453–460, New York, NY, USA, 1998. ACM.

[8] Henry Kang, Seungyong Lee, and Charles K. Chui. Coherent line drawing. *NPAR '07: Proceedings of the 5th international symposium on Non-photorealistic animation and rendering*, pages 43–50, 2007.

[9] Peter Litwinowicz. Processing images and video for an impressionist effect. pages 407–414, 1997.

[10] D. Marr and E. Hildreth. Theory of edge detection. *Proceedings of the Royal Society of London. Series B, Biological Sciences,*, 207(1167):187–217, February 1980.

[11] Image Metrics and USC Institute for Creative Technologies Graphics Lab. High resolution face scanning for "digital emily". http://http://gl.ict.usc.edu/Research/DigitalEmily/, 2008.

[12] Michael P. Salisbury, Sean E. Anderson, Ronen Barzel, and David H. Salesin. Interactive pen-and-ink illustration. pages 101–108, 1994.

[13] Jutta Schumann, Thomas Strothotte, Stefan Laser, and Andreas Raab. Assessing the effect of non-photorealistic rendered images in cad. pages 35–41, 1996.

[14] Adrian Secord. Weighted voronoi stippling. pages 37–43, 2002.

[15] Mario Costa Sousa and Przemyslaw Prusinkiewicz. A few good lines: Suggestive drawing of 3d models. *Computer Graphics Forum*, 22(3):381–390, September 2003.

[16] T. Strothotte, B. Preim, A. Raab, J. Schumann, and D. R. Forsey. How to render frames and influence people. 13(3):455–466, 1994. Special issue on Eurographics '94.

[17] Thomas Strothotte and Stefan Schlechtweg. *Non-Photorealistic Computer Graphics: Modeling, Rendering and Animation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, June 2002.

[18] Georges Winkenbach and David H. Salesin. Computer-generated pen-and-ink illustration. pages 91–100, 1994.