Assigment 4 – Artistic Stylization

Department of Electrical and Computer Engineering
Ryerson University

ELE 882 – Fall 2017

1 Introduction

One application of image processing is to perform so-called "stylization" or "non-photorealistic" effects. Essentially it's manipulating the image to produce an effect that simulates a "natural" artistic effect, such as making an image look like an oil painting, or to produce an effect that is completely unnatural, such as changing the colour pallet of the image. This assignment will consist of two parts: the eXtended Difference of Gaussians (XDoG) effect and a cartoon drawing effect. The first effect will be explained in detail while you will only have a very brief overview of the second effect.

1.1 Colour Images

In the previous assignments, all of the processing was done only on greyscale images. However, many of the stylization effects work, and look, best on colour images. For our purposes, the only difference between a greyscale and colour image is that a colour image is actually *three* separate greyscale images.

Processing colour images in Matlab, and by extension MEX code, is simple. Matlab considers all matrices to be stored in **column-major** format with each dimension being a constant offset from the other. As you've already seen, a pixel with a coordinate (x, y) in a $H \times W$ image is accessed by

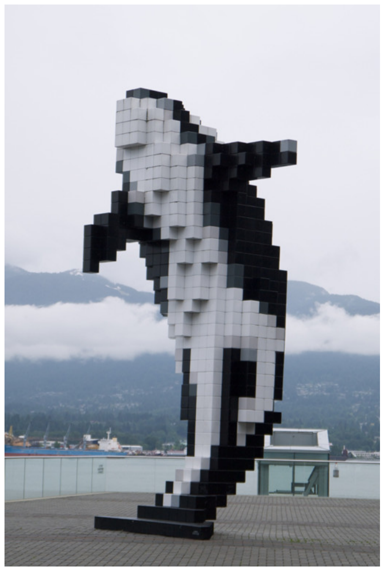
$$i = Hx + y, \tag{1}$$

where i is the pixel's *index* value. This is done with the assumption that the indexing is zero-based, like it is in C.

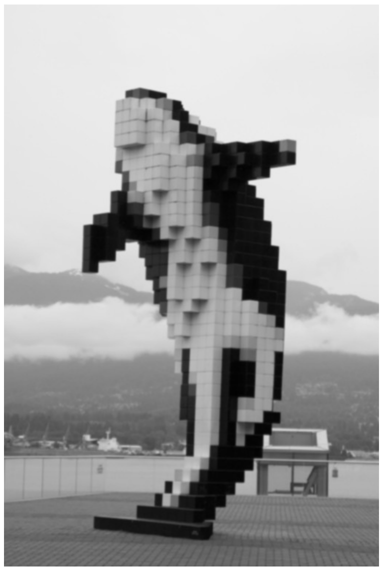
What this means is that for a pixel on column y, its neighbouring pixel (horizontally) at $y + 1$ is located H elements away in memory. The same is true for colour images except that the offset between colour channels is HW. This is from the fact that we have a 2D image and the colour channels represent a third dimension when addressing. With this in mind, the colour-based indexing is

$$i = (Hx + y) + HWc, \tag{2}$$

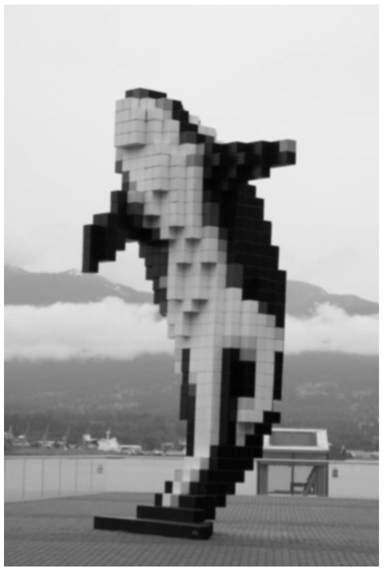
where c is the colour channel index. Matlab uses the convention where $c = 0$ is the red channel, $c = 1$ is green channel and $c = 2$ is blue channel.



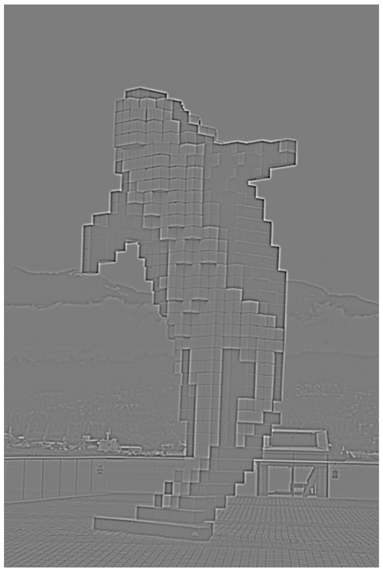
(a) Original Image



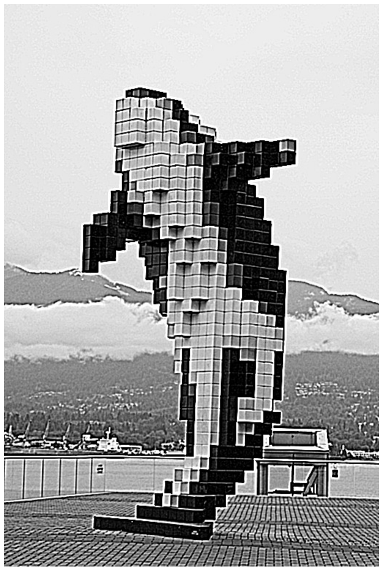
(b) $G_\sigma(x, y)$



(c) $G_{k\sigma}$



(d) $G_\sigma(x, y) - G_{k\sigma}(x, y)$



(e) $D_{\sigma,k,p}(x, y)$

Figure 1: The XDoG filter's processing steps using $\sigma = 0.9$, $k = 1.2$ and $p = 100$. The original image (a) is blurred twice: once with a Gaussian kernel with a variance σ (b) and then with a kernel of veriance $k\sigma$ (c). The XDoG image in (e) is then obtained by using the DoG image (d) to "sharpen" the less-blurred image.

2 Stylization Effects

2.1 eXtended Difference of Gaussians

The eXtended Difference of Gaussians (XDoG) filter [**?**] is a simple stylization technique based on second-order derivative filtering and thresholding. Let $G_\sigma(x, y)$ be the filtered version of the greyscale image $I(x, y)$, filtered with a Gaussian low-pass filter with a variance[1] of σ. The XDoG filter is then defined as

$$D_{\sigma,k,p}(x, y) = (1 + p)G_\sigma(x, y) - pG_{k\sigma}(x, y). \tag{3}$$

The k term in the second filtered image is a multiplier that controls how much larger the other blurring kernel is from the original one. A value of $k = 1.6$ will approximate a *true* Laplacian of Gaussian filter but it can be set to anything as long as $k > 0$ (if it's less than zero then it will make $k\sigma < 0$).

One thing to observe is that this doesn't preserve the range of values in the original image. There is no restriction on the value of p so it can be chosen to produce a $D_{\sigma,k,p}(x, y)$ with a very wide range of values (from -1000 to 1000, for instance). To produce the final output image, Winnemoller et al apply some simple point operations to $D_{\sigma,k,p}(x, y)$ to bring it to the expected image range of 0 to 1. In the paper, they define a simple hard threshold

$$T_\epsilon(u) = \begin{cases} 1 & u > \epsilon \\ 0 & \text{O.W.} \end{cases}, \tag{4}$$

where ϵ is the threshold value. They also define a "soft" threshold

$$T_{\epsilon,\varphi} = \begin{cases} 1 & u > \epsilon \\ 1 + \tanh(\varphi(u - \epsilon)) & \text{O.W.} \end{cases}, \tag{5}$$

where $\tanh(\cdot)$ is the **hyperbolic tangent** function and the φ term controls how sharp the transition is between 0 and 1 in the output image.

As presented, the XDoG filter is somewhat deceptive. However, by rearranging (3) it becomes easier to see what this filter is actually doing:

$$\begin{aligned} D_{\sigma,k,p}(x, y) &= (1 + p)G_\sigma(x, y) - pG_{k\sigma}(x, y) \\ &= G_\sigma(x, y) + pG_\sigma(x, y) - pG_{k\sigma}(x, y) \\ &= \underbrace{G_\sigma(x, y)}_{\text{Blurred Image}} + p\underbrace{\left(G_\sigma(x, y) - G_{k\sigma}(x, y)\right)}_{\text{Edge Image}}. \end{aligned}$$

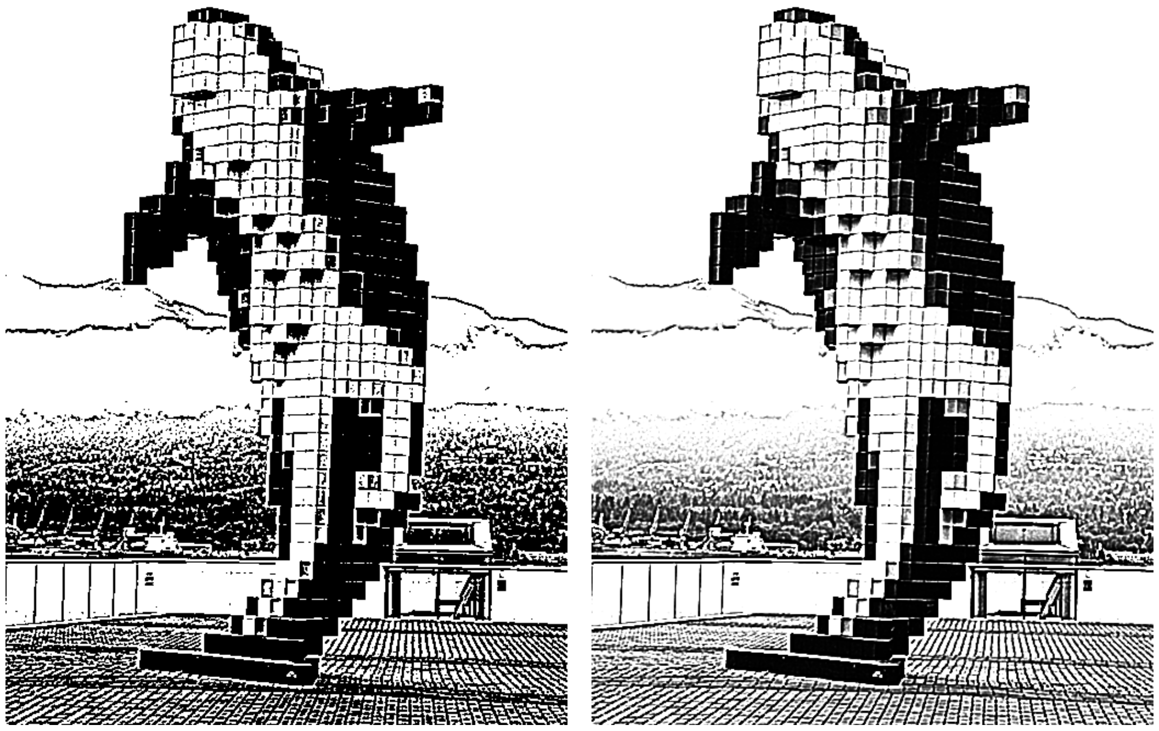
This is simply an unsharp masking. The only difference between a normal unsharp masking is that the image being processed is blurred and then that *blurred* image is "sharpened". The point of this is to force the "lobes" along the edges, the overshoot that occurs when the edges are added back onto the image, to be somewhat large. By choosing the threshold appropriately, a number of different effects can be achieved. More details are provided in the original paper and has been made available on Blackboard.

Figure 1 shows the different stages of the XDoG filter. The thresholding has been omitted as even though it's considered to be part of the filter, it is really a post-processing step. One thing to note is that the larger p gets, the closer the XDoG filter becomes to a normal DoG filter. The other is that

[1]Recall that the height and width of the Gaussian filtering kernel should be around 6σ.

Figure 1e is shown with the values *truncated* to be on 0 to 1. The actual image produced by the filter has values between -4.27 to 5.39.

Once $D_{\sigma,k,p}(x,y)$ is available, there are a number of different ways to threshold or process it. Figure 2 shows the two different thresholding approaches provided explicitly in [**?**]. The thresholding parameters are similar so the two images appear similar. The main difference is that hard thresholded image has distinct edges while the soft thresholded image gives the appearance of shading.

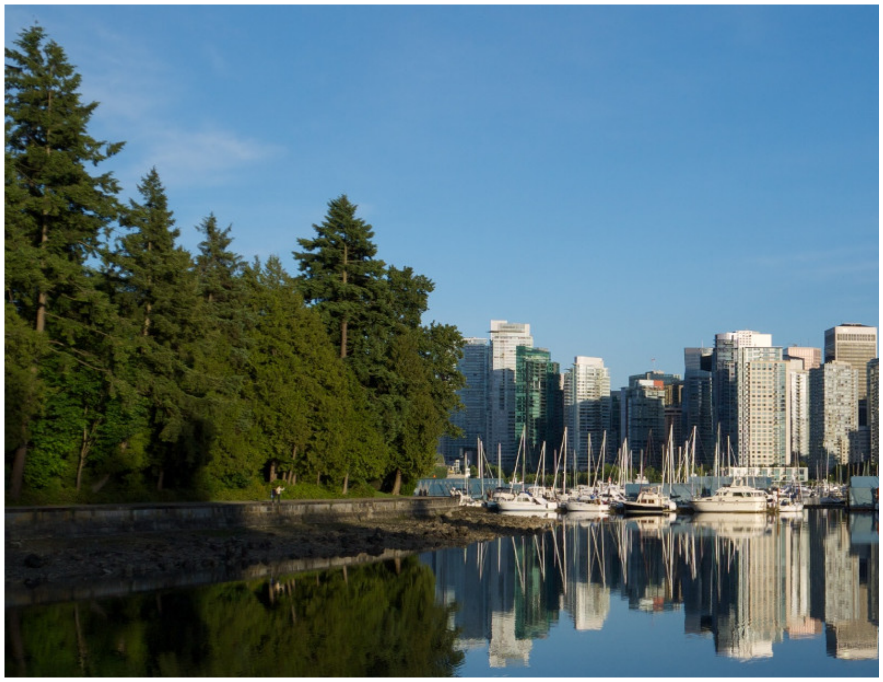


(a) Hard Thresholding (b) Soft Thresholding

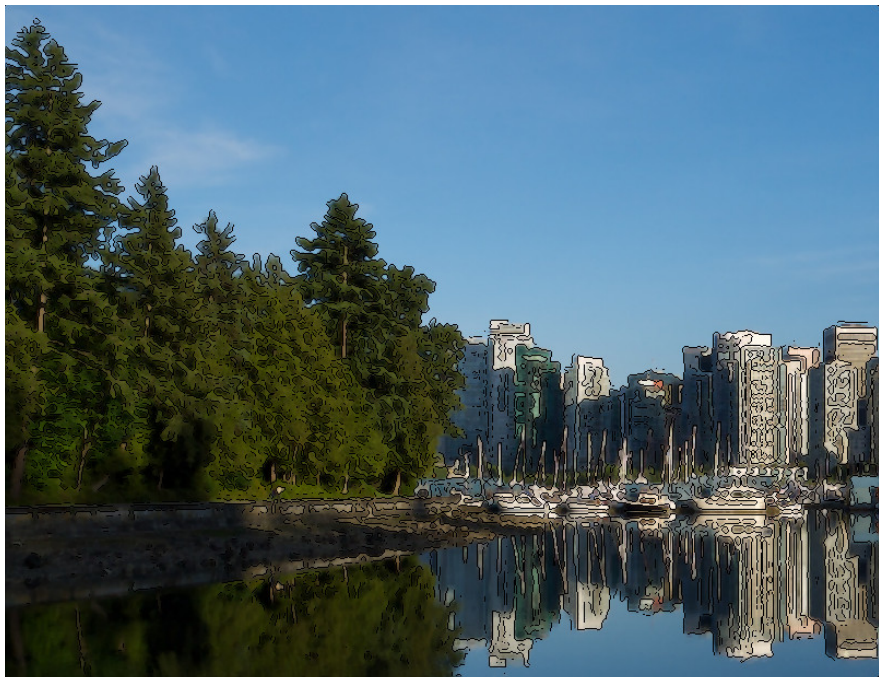
Figure 2: Thresholding the XDoG-filtered image from Figure 1. Both the hard and soft thresholds, equations (4) and (5) respectively, are shown. The thresholding parameters were $\epsilon = 0.5$ and $\varphi = 6$.

2.2 Cartoon Effect

The cartoon filter does two things. First it “smears” the colours in an image to give the appearance that the image has been painted. After that, it overlays the edges onto the “painted” version of the image. Figure 3 shows an example of what this looks like. Generating this effect requires figuring out: how to do the initial filtering, finding edges to overlay and then finally blending/merging the edges with the painted image.



(a) Original Image



(b) Cartoon-filtered Image

Figure 3: An image processed using a possible cartoon effect filter. This effect was accomplished using a combination of simple filters applied multiple times. All of the filters used were already presented in the lectures; only *how* they were applied is new.

3 Assignment

This assignment builds off of the work that you have done in the previous assignments. Most of what is being done is simply a new way of applying existing concepts. Even the XDoG filter is just a composition of operations that you have already encountered: point operations and spatial filtering. The one new concept is how to handle colour images. Even there, this is just a slight extension to what you have already been doing. As stated in Section 1.1, colour images are really just three greyscale images. Use the indexing introduced in Equation (2) when processing those images to access the different colour values.

Unlike the previous assignments, you now have much more leeway in how you go about *implementing* the different filters. That means that you will be assessed both on the quality of your report *and* documentation. In the report you must provide a full description of how the algorithms work along with how you went about implementing them. In your code you must provide clear instructions how that code is to be used. Failure to do so will be reflected in the final marks.

Important: The colour-related processing is completely **OPTIONAL**. However, you are encouraged to try colour processing as the results, particularly for the cartoon effect, are much more visually appealing. The effect filters are completely agnostic to whether or not the image is greyscale or colour. The only difference is that for colour images, the effect is applied three times (once per colour channel).

3.1 Problems

3.1.1 XDoG [20 marks]

1. **[10 marks]** Implement the eXtended Difference of Gaussians filter as described by equation (3). The XDoG filter is purely greyscale-based so you should convert your images prior to processing using `rgb2gray()`. It would also be advisable to convert the images into the `double` data type using `im2double()`.

2. **[5 marks]** Implement the thresholding functions described in equations (4) and (5). Unlike the thresholding from Assignment 1, these should be considered more as point operations than as true thresholding (thresholding produces binary images as this does not).

3. **[5 marks]** Implement a **three-tone** operator similar to the one shown in Figure 7c in of the original XDoG paper. Refer to the PDF provided on Blackboard.

3.1.2 Oilify Filter [15 marks (Bonus)]

The oilify filter produces an interesting “oil painting” effect. The filter uses a number of ideas that are somewhat outside of the scope of the course (though not so far outside that it would be impossible to understand). Therefore, it has been provided as supplementary material and implementing it is optional. Refer to the `oilify.pdf` file on Blackboard for more information about this effect.

3.2 Cartoon Effect [25 marks]

The key to obtaining the cartoon effect is to *experiment*. The hints provided below are just that, hints. You must explain, in your report, how your solutions actually end up generating the desired effect.

1. **[10 marks]** Design a filter that meets the following criteria: edges are preserved while larger regions are smoothed out.

   Hint: A median filter *almost* meets this definition. The problem is that the larger the filter window, the more filtering action you will have, i.e. you start losing prominent edges. One possible solution, and what was done to generate Figure 3b, is to run a median filter *multiple* times while *changing* the filter window size each iteration.

2. **[10 marks]** Design a filter that extracts prominent image edges.

   Hint: This is essentially an edge detector but not in the same way, for example, as in Assignment 2. Here the detector is more about finding edges in a visually appealing way then necessarily worrying about how accurate it is. Therefore, one option could be to just directly threshold the gradient magnitude. Another could be to post-process the edge maps.

3. **[5 marks]** Define an operator that will merge the edge map with the smoothed image to produce the final cartoon effect. There are several ways to accomplish this, ranging from just setting the pixel values to zero wherever there is an edge (results in black lines) to something more complicated, like choosing the edge colour based on the surrounding pixel values.

   Hint: If you are designing a purely greyscale effect filter, you may also want to add a step that will change the image brightness and/or contrast to make the lines more visible.

4. **[10 marks (Bonus)]** Incorporate the XDoG and/or oilify filters into the cartoon effect. This could be either to augment the existing effect (use the XDoG results for the edges) or to use a combination of the XDoG and oilify filters to produce the actual cartoon effect.