Project 3 Part B Report

Dian Chen, Yu Sun

November 2, 2017

1 Abstract

We're proposing a mini project to generate non-photorealistic pictures from real photos. The motivation comes from the fact that there are tons of Apps for mobile phones these days that can render our daily life photos in multiple styles, such as cartoon, oil painting and pencil sketches, in a mere tap. It would be fun to explore and implement the postprocessing of real photos in reasonable, yet aesthetic ways. We're approaching this in a way of filter design and utilizing random noise functions. Part of our implementation would be based on a paper written by Eduardo and Manuel [1], and the rest of the designs would just come from pure brainstorm.

2 Related Work

Mobile phone Apps such as Prisma and Meitu have been quite common and popular these days with which people can easily produce stylish pictures from their own daily shots, and built-in photo editors are even getting as powerful ever. Common render styles include: cartoon, oil painting, pencil sketch, mosaic, or simply adjusting the color tone of the image. These kinds of post-processing are called "Non-Photorealistic Rendering" (NPR), which is to modify a real photo to give it a more aesthetic (or just for fun) without being natural.

The idea of non-photorealistic rendering has been out for a while, and there are primarily two classes of ways of achieving the effect. The more classic and traditional class is through filter designing, which is to intentionally design filters to catch desired patterns underlying the image and utilize the patterns coming out after the filtering. Examples include *Domain Transform for Edge-aware Image and Video Processing* by Eduardo S. L. Gastal, Manuel M. Oliveira [1], *Richness-Preserving Manga Screening* by Qu, Y., Pang, W., Wong, T., Heng, P. [2], and *Simple Art as Abstractions of Photographs* by Peter Hall and Yi-Zhe Song [3], all of which come up with distinct filters used to manipulate the raw image. Some of the work pursue real time performance on top of desirable visual effects.

The other class of methods is to use convolutional neural networks (CNN), which has grown increasingly popular in commercial Apps like Prisma. Given a image to be rendered, and given a reference style, it can render the image into the style in an amazingly compatible way. *A Neural Algorithm of Artistic Style* by Leon A. Gatys, Alexander S. Ecker and Matthias Bethge is the first paper proposing this idea, and Justin Johnson of Stanford University has an implementation in Torch. The most striking examples are rendering daily photos into styles of Vincent van Gogh's Starry Night, and of other paintings from Pablo Picasso, Monet, etc.

3 Methodology

We're describing all of the filters we've implemented here. The implementation is all in MATLAB.

3.1 Domain Tranform Filter Based Processing

In this project we're approaching the NPR in a way of filter design, with some ideas of shader programming borrowed from computer graphics such as utilizing random noise functions. We will try to implement some filters suggested in [1] and achieve the expected effects, as well as go completely wild with our own ideas of creating filters and manipulating the pixels.

In the paper they described a technique that can preserve edges while smoothing the images. The main idea behind the methodology is to find a domain transformation from R^5 (R,G,B,X,Y) into lower dimensions while preserving the distances. Filtering is then applied to the transformed image. The details steps are as followed:

1. Compute x and y gradient, L1 norm of neighboring pixels
   The nearest-neighbot L1 norm serves as a way to perserve the original distance between points on a curve C in R^2 in the graph $(x, I(x))$, such that $|t(x_i, I(x_i)) - t(x_j, I(x_j))| = ||(x_i, I(x_i)) - (x_j, I(x_j))||$ with t as the transformation

2. Compute x and y domain transforms
   In order to satisfy the equation listed above, the transform t must satisfy $ct(x+h) - ct(x) = h + |I(x+h) - I(x)|$, leading to the equation $ct(u) = \int_0{}^u 1 + |I^{'}(x)|dx$.
   For multichannel signals, the following holds:
   $ct(u) = \int_0^u 1 + \sum_{k=1}^c |I^{'}(x)|dx$

3. Perform edge-preserving filtering
   The ability to control the support over signal's space σ_s and range σ_r are encoded in the transformation. For each channel k, $\sigma_{rk}^2 = Var(H_{1/\alpha}) = Var(H)/\alpha^2 = \sigma_H^2/\alpha^2, k$ holds.
   The domain transform is thus $ct(u) = \int_0^u \frac{\sigma_H}{\sigma_s} + \sum_{k=1}^c \frac{\sigma_H}{\sigma_{rk}}|I^{'}(x)|dx$

4. Perform recursive filtering in the transformed domain
   For a discrete signal $I[n] = I(x_n)$, a recursive edge-preserving filter can be defined as :
   $J[n] = (1-a^d)I[n] + a^d J[n-1]$
   $d = ct(x_n) - ct(x_{n-1})$ is the distance between neigboring samples in the transformed domain.

This technique is very versatile and can then utilized in different applications, and we'll explore some of them.

3.1.1 Detail Enhancement

One example here is a straightforward implementation for image detail enhancement.
The process is very straight forward, we take the difference of the filtered image and the original image, multiply that difference with a factor, and then add it back to the original image.

3.1.2 Stylization

Another example here is a stylization technique. We take the gradient G_x and G_y of the image using a sobel filter, computing the magnitude of the edge with $G = \sqrt{G_x^2 + G_y^2}$ for each channel. We boost the edge with a amplication factor and the final output image would be the original image substracting the amplified edge.

3.2 Brainstorm

3.2.1 Electric Effect

This is a filter design work that comes with no direct intuition. We just manipulated the pixels and waited to see the results. The manipulation is as follow:

1. use two 3 by 3 sobel filters (horizontal and vertical) to filter the image, and get two gradient maps G_x, G_ycorresponding to the horizontal and vertical direction, respectively. The filtering is done in three separate R, G, B channels, so we should get in total 6 gradient maps.

$$S_x = \begin{bmatrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{bmatrix}, \quad S_y = \begin{bmatrix} 3 & 10 & 3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix}$$

2. For each channel, compute the length of the gradient, i.e., for each pixel compute the square root of the sum of squares of the gradient.

$$\sqrt{G_x(i,j)^2 + G_y(i,j)^2}$$

3. For each pixel in each channel, set its intensity value to this length of the gradient.

$$I(i,j,r) = \sqrt{G_{x,r}(i,j)^2 + G_{y,r}(i,j)^2}$$

$$I(i,j,g) = \sqrt{G_{x,g}(i,j)^2 + G_{y,r}(i,g)^2}$$

$$I(i,j,b) = \sqrt{G_{x,b}(i,j)^2 + G_{y,r}(i,b)^2}$$

This created a geeky looking electric effect on the original image.

3.2.2 Blooming Effect

This idea is borrowed from computer graphics, which aims to fake the effect in which background light would leak through the boundary of an object (see the picture below).

Normally, this involves shader programming with several steps. We implemented this in a simplified way as follow:

1. Take a Gaussian filter, say 7 by 7. And set a intensity threshold t.
2. Use the filter to filter the image with a special care: for each pixel beneath the filter, only take it into account if its intensity is above the threshold. For example, if a pixel has intensity 30 in R channel and the threshold is 45, then the pixel has 0 contribution in the convolution.
3. Operate in three channels. Add the filtered result to the original image and get the final result.

In this operation we have many knobs to tweek, such as the threshold in each channel, and the blurring kernel to use in step 1.

3.2.3 Mosaic Effect

This idea utilized Worley Noise which is widely used in computer graphics. Worley Noise is some random pattern that correponds to a Voronoi diagram, and can be generated as follow:

1. Generate many random points in the whole picture as control points. In order to make them distribute evenly, we can divide the picture into N by N cells and put a random point in each cell.
2. Use the control points to divide the picture into a Voronoi diagram, so that each pixel lies inside a certain Voronoi cell and has a control point that corresponds to that cell.
3. For each pixel, find its control point and color it with the same color as the control point

In this way we can generate mosaic effect on the original image, in which each facet has a different, aesthetic shape. The implementation simulates GPU shader programming except that the computation is not parallel.

3.2.4 Bubble Effect

Worley Noise is commonly used in computer graphics to produce fun effects during post-processing. The same idea can be used here to add realistic-looking bubbles on top of the image. Based on the process described in Mosaic Effect, we can make some modification to create bubbles on top of the original image:

1. For each pixel, find its control point and compute the distance d to that point

2. Use the distance d to compute the height of the bubble surface

$$z = \sqrt{R^2 - d^2}$$

in which R is a tunable parameter representing the diameter of a bubble sphere. Now we have a height map $h(x, y)$.

3. Take the gradient of the height map in x and y direction, and use that to form a normal vector for each pixel location

$$n(x, y) = (g_x, g_y, -1)$$

4. Set a light source vector. Compute for each pixel the Lambertian light coefficient using the dot product between the surface normal and the light source vector.

5. Set a view vector a compute the specular reflection coefficient. Also add some constant ambient light coefficient.

6. For each pixel, retrieve the color of the original image with displacement to simulate the light refraction through bubbles.

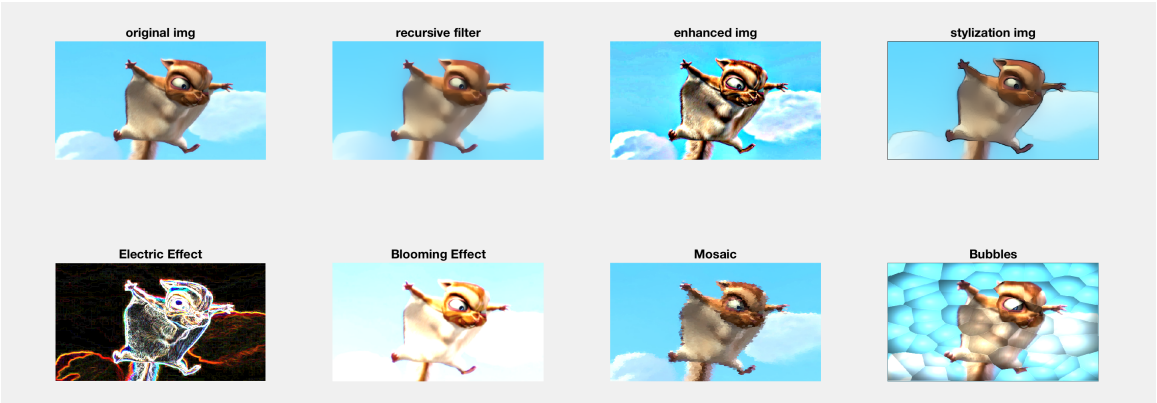
7. Multiply the color with the sum of Lambertian coefficient, specular coefficient and ambient coefficient. Set the final color of that pixel to the computed result.

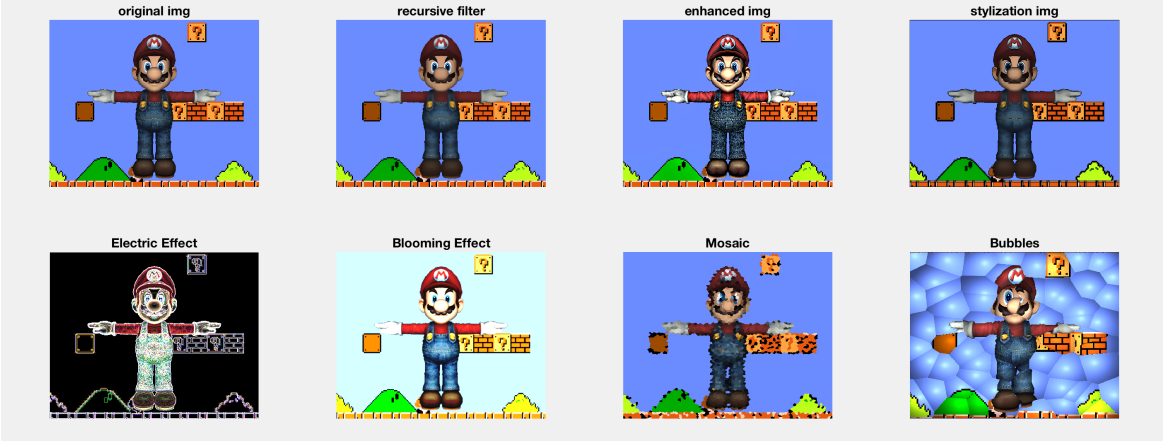
This process simulate the light condition of a real bubble surface, and the implementation simulates GPU shader programming except that the computation is not parallel. And this comes mostly from computer graphics and a course which Dian is taking.

4 Comparison with your proposed methodology

In the proposal, we mentioned that we would implement the Normalized Convolution. However, we implemented the Recursive Filtering Kernel. This is a little bit conterintuitive because the recursive filter actually runs faster in Matlab because of the nature of the code, and both methods produce similar results.

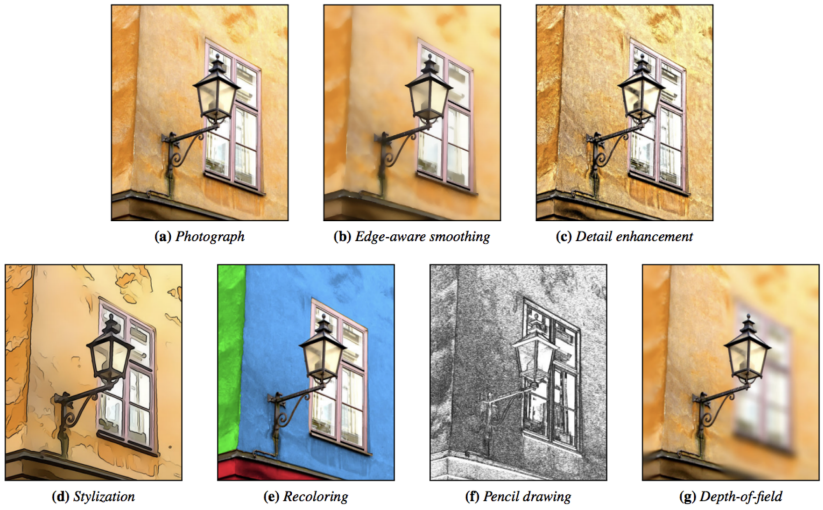
5 Achieved results





6 Analysis of comparison with expected results

Section 3.1 relies on [1] and have predictable results. The output from edge-aware smoothing filter should be a blurred version of the original image, with the edges being preserved. We can see that we've produced similar effects as suggested in the paper:



The results of 3.2 also looks amusing and reasonable. The blooming effect resembles the real blooming in CG industry at some level.

7 Reference

1. Eduardo S. L. Gastal, Manuel M. Oliveira, "Domain transform for edge-aware image and video processing", ACM Trans. Graph. 30(4): 69, 2011.

2. Qu, Y., Pang, W., Wong, T., Heng, P. (2008). Richness-preserving manga screening. ACM SIGGRAPH Asia 2008 papers on - SIGGRAPH Asia 08. doi: 10.1145/1457515.1409108

3. Peter Hall and Yi-Zhe Song. 2013. Simple art as abstractions of photographs. In Proceedings of the Symposium on Computational Aesthetics (CAE '13), Stephen N. Spencer (Ed.). ACM, New York, NY, USA, 77-85. DOI: https://doi.org/10.1145/2487276.2487288

4. http://www.cis.upenn.edu/ cis460/current/