Digital image processing paper*

*Note: Sub-titles are not captured in Xplore and should not be used

Tumi Jourdan 2180153
CSAM

Wits University

***Abstract*—This document presents implementations for the artistic stylization and cartoonization of static color images. The core methodology employed is an extended difference of Gaussians (XDoG) approach, which facilitates the extraction and manipulation of image edges to achieve various stylistic effects, including charcoal artworks, pencil sketches, and abstract ink drawings. The XDoG technique is further enhanced through the incorporation of edge flow-based filtering and line integral convolution, enabling denoising and edge smearing characteristics. Furthermore, color manipulation techniques, such as k-means clustering and color smearing inspired by the artistic style of Vincent van Gogh, are introduced to realize cartoonized and painterly aesthetics. The paper explains the implementation details, highlighting the utilization of GPU acceleration via OpenCL to address computational challenges arising from the processing of high-resolution images. The proposed methodologies offer a versatile framework for transforming conventional images into visually captivating artistic renditions.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

This project aims to implement a caroonization and artistic stylization of still color images. When first introduced to this topic a video from Acerloa **YouTube** covering methods of an Extended difference of gaussians developed by the paper [1] came to mind. This method in the paper generates several parameters that can be used to adjust and cosomize the results. These results can then be blended back with the original colors. This creates the oppertunity to also edit the color data before blending. The colors are manipulated using an edge flow tensor , or a more cartoony kmeans flattening.

II. Implementations

A. Difference of Gaussians (DoG)

The concept behind difference of gaussians is that an image is passed through a gaussian filter with some standard deviation σ (g), the original image is then passed through another gaussian filter but this time the standard deviation is scaled by scalar k, $k\sigma$ (gk). Taking the difference between these gaussians results in a binary image demarcating the edges. This result is an estimation of the Laplacian of Gaussians that is more computationally efficient. Throughout the paper the k will allways be set to 1.6 as Winmoller 2012[1] found it to work well.

Identify applicable funding agency here. If none, delete this.

The reason that the DoG finds the edges so well is that the Gaussian filter acts as a low pass filter, and taking the difference of these generates a band pass filter thereby detecting edges. To control how the DoG operates another scalar is introduced, p. Through reparameterization , [1] creates a new equation:

$$(1+p)\cdot G_\sigma(x) - p\cdot G_{k\sigma}(x) \tag{1}$$

The new equation scales the first gaussian and the second gaussian appropriately along with the scalar p which represents the sharpness of the DoG.

This method is further improved using a hyperbolic tangent thresholding function to accentuate the edges of the image.

$$T_{\epsilon,\phi}(u) = \begin{cases} 1 & \text{if } u \geq \epsilon \\ 1 + \tanh(\phi \cdot (u - \epsilon)) & \text{otherwise} \end{cases} \tag{2}$$

B. Flow based Difference of Gaussians (FDoG)

To further improve and denoise the DoG, we replace the gaussian blurring in the DoG with 1 dimensional Gaussian blurrings accross the vector flow fields of the image. The vector flow fields are estimated by generating the edge tangent flow map (etf). The following steps are how to make an etf:

- First apply a vertical and horizontal sobel filter on the image, this approximates the the differential (gradient) of the image.
- From this we can calculate the structure tensor of the image which describes the gradient around each pixel.
- From this we can calculate the eigen vectors and eigen values. These represent the directions of the gradient, and by selecting the smallest one (min) we can determine the direction of least change of each pixel. Resulting in the etf.

With the etf, we can then calculate the 1 dimensional gaussians perpendicular to the vector direction of each pixel. Figure 1 demonstrates how the gaussian filters are applied along the direction of borders.

The ouput from this new DoG is good but is still too noisy. To fix this we blur along the edge lines produced from the DoG result. This is achieved through the line intergral convolution.

1) Line Integral Convolution: Line integral convolution is the process of applying gaussian filters, but the centre of the filter is moved along the direction of the etf until it exceeds the

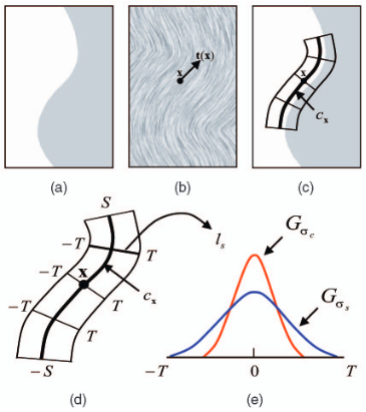

Fig. 1: (a) Input. (b) ETF. (c) Kernel at x. (d) Kernel (e) Gaussian components for DoG frome **Kang2009**

boundries of the filter instance. It then does the same process but in the opposite direction of the etf. Fugure 2 demonstrates the order in-which the center of the filter is moved along a line.

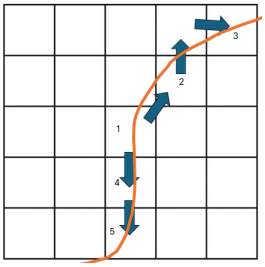

Fig. 2: Center moved from position 1 to 5

Not only does this effect remove noise , but it also smears the image along the edge lines. Adding a more lifelike artistic style to the edges.

The final FDoG has a few parameters that can be used to tweak the resulting image.

- σ_c : The initial bluring of the image during the construction .
- σ_e : The gaussian bluring of the 1 dimensional gaussians that are subtracted from one another.
- σ_m : First aligned linear convolution.
- p : The sharpness scalar from the DoG reparametrization 1.
- ϕ : The sensitivity of the continuous ramp in the hyperbolic tangent.
- ϵ : The threshold of hyperbolic tangent thresholding.
- σ_a : Second aligned linear convolution.

These parameters can make abstract ink drawings, charcoal artwarks, and penned artworks. Later it is also used for the edging of cartoon image simulation.

C. Colour adjustments

In the previous sections, all of the manipulations are to find and manipulate the edges of the images, to further the stylization of the images we can manipulate the colour data. There are 2 implemented methods, K-means flattening, and Vincent van Gogh-esque painting.

1) K-means flattening: Many cartoons are stylized by their lack in gradient shading, opting instead for hard transitions in shade. This style can be replicated by utilizing the k-means clustering algorithm to create regions from the color data of the image.

When the k-means cluster converges on k-clusters from the image, each k-means center is extracted and represents the dominant color of an area. Each pixel has its color reassigned to the associated k-means center, effectively quantizing the image.

By adjusting the number of clusters (k) in the k-means algorithm, you can control the level of color flattening and stylization. A smaller number of clusters will result in fewer distinct colors and a more pronounced cartoonized effect, while a larger number of clusters will preserve more color variations.

The k-means method alone is one style, but by blending it with the FDoG edges creates cartoons with dark edges.

2) Vincent van Gogh-esque painting: Whilst learning about the line integral convolution (lic), etf maps and its smearing properties, it became apparent that this could be fused with the colour data to smear the colors, resulting in a more painterly aesthetic.

By first splitting the colour data, applying the lic (with an extremely high sigma in the gaussian blurring of the etf) to each color channel, merging the colors again results in an image with simulated brush strokes.

III. Techincal difficulties

Much of this paper is exploring already trodden ground, mostly in the creating of the FDoG system, but a novel difficulty reared its head during development. Python can only run on a single core, and usually this is sufficient. But in the case of the edge aligned gaussian filters, it was a farcry from optimal. Using the tqdm python package it estimated that the 1d convolution of the gaussians along the vector fields would take 11 hours. The reason for this is that some images that are used have a resolution of $1920x1080$ pixels, which is $2,138,400$ pixels that need to be processed.

To fix this the pyopencl package is used in conjunction with .cl files to move the computational load to the gpu. This made the computation time near instant. Sadly the OpenCl language does not include many of the useful libraries that python contains, mainly GaussianBlur. So custom kernels were made in the stead. OpenCL also does not work with 2d arrays so most of the arrays had to be linearized to be operated on in the gpu.

References

References

[1] H. Winnemöller, J. E. Kyprianidis, and S. C. Olsen, "XDoG: An eXtended difference-of-Gaussians compendium including advanced image stylization," Computers & Graphics, vol. 36, no. 6, pp. 740-753, 2012.

[2] H. Kang, S. Lee, and C. K. Chui, "Flow-Based Image Abstraction," in *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 1, pp. 62-76, Jan.-Feb. 2009, doi: 10.1109/TVCG.2008.81.

[3] YouTube, "This is the Difference of Gaussians," YouTube, Dec. 24, 2022. [Online]. Available: https://www.youtube.com/watch?v=5EuYKEvugLU&t=32s. [Accessed: 02-Jun-2024].

Fig. 3: Charcoal

Fig. 4: Pen Style

Fig. 5: abstract

Fig. 6: Cartoony kMeans

Fig. 7: Van Gogh Style 1

Fig. 8: Van Gogh Style 2

Fig. 9: K means smeared and blended with edges 1

Fig. 10: K means smeared and blended with edges 1