FLOW BASED PICTURE STYLISATION

LOHOU *Anthony,* LAURENT *Tristan*

ESIR
Imaging Processing
35042 Rennes

ABSTRACT

In this document we present an automated non-photorealistic image processing techniques that delivers a stylized abstraction of a photograph. We tried different techniques to reduce the color number inside the picture and increase the edges size to obtain a "comics" result. The first approach used were not strong enough to take only significant edges and preserve edges colors. The second is an approach based on a filtering guided by a vector field that describes the flow of the salient features in the input image. Using the flow based approach allows us to find the major edges of the shapes in the image. It also, create an abstraction of the image without suffering of the creation of artefacts.

Index Terms— Non-photorealistic rendering, Flow-based filtering, bilateral filter, line extractor

1. INTRODUCTION

Non-photorealistic rendering (NPR) is a form of computer graphics. It is inspired by drawing, painting and technical illustration. In this document we used NPR to create a cartoon effect on photo-realistic pictures. The cartoon effect is based on two parts. The first one is to extract the main edges int the raw picture, several algorithms can be used for this. The selection of the majors edges depends on the algorithm. The second part is the region smoothing. As the first part, several algorithms exist but most of them erase boundaries between the shpaes in the raw picture. to smooth the regions we apply iteratively a filter on the picture to smooth the difference between the pixels. Then we apply a quantization algorithm to reduce the numbers of colors or gray levels.

2. EXISTING ALGORITHMS

Before the implementation of the flow-based method, we implement common methods used to extract lines and smooth regions.

2.1. Line extractors

To extract lines, we first basically computed the gradient magnitude. The results were to noisy, thus most significant edges can not be determined.
As a better solution we implement a difference of Gaussian. The results were better but still noisy.
The best solution that we found was the Difference of Gaussian with a filtered zero-crossing. The noise is reduced when we filter the zero-crossing values and most significants edges appear. But the result is still unacceptable, edges are not preserved as you can see in Fig. 1.

2.2. Smoothing regions

Different methods can be used to smooth regions. First method could be a simple gaussian blur, in order to reduce the difference between pixels of a picture. Pixel values will be closer to its neighbour, so after quantization we obtain a smoother region than a quantization without pre-process. But the result is not smooth and a lot of details appears around boundaries.
A bilateral filter can also smooth regions, after some iteration the result is correct but the regions exceed boundaries. Edges details are not preserved.

3. FLOW-BASED FILTERING

In this document we choose to apply the algorithm of "Flow based filtering from [1]. Another document from [2] provides a different algorithm approaching the same results. We choose to implement [1] instead of [2] because the results looked better and the approach easier to implement.

3.1. How does it work ?

Flow-based filtering follows the same rules than the common methods of stylisation. The process is divided on two parts, one for the region smoothing and the other for line extraction as you can see in Fig. 2. With this method we firstly need to compute a edge tangent flow (ETF) map from a raw picture. The ETF map is used in both parts of process.

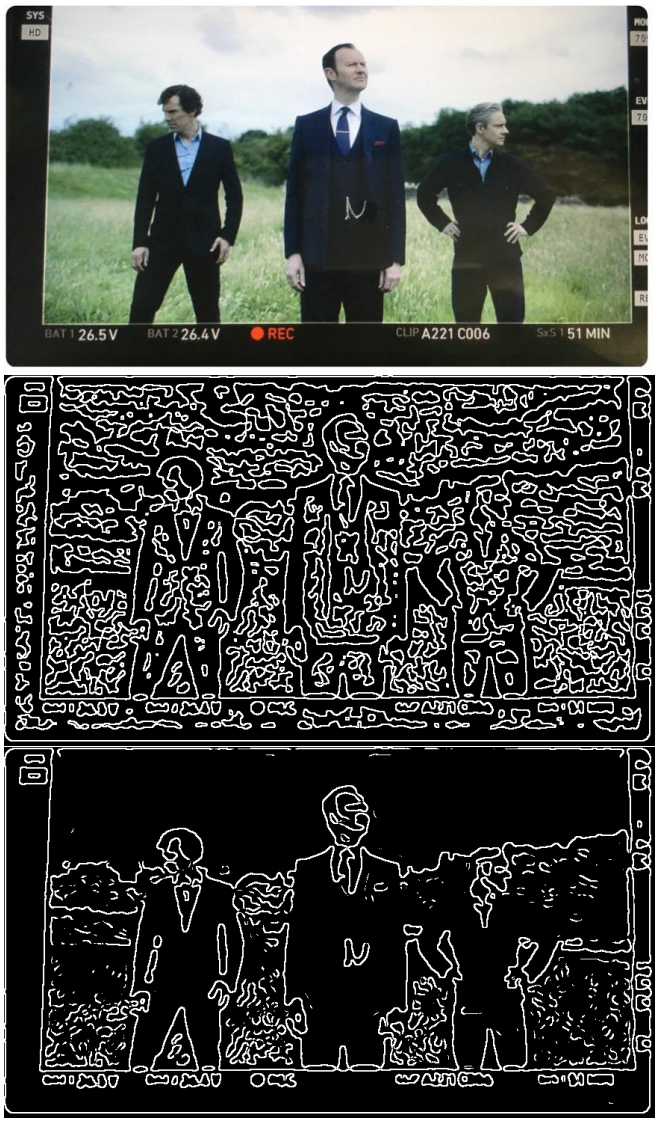

Fig. 1. From top to bottom: original, DoG with zero crossing, DoG with filtered zero crossing

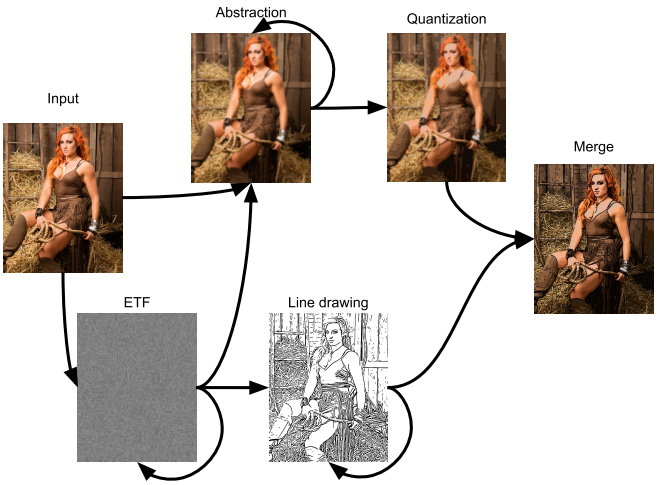

Fig. 2. Flow-based filtering schema

Once the ETF is computed, we use it for the abstraction part and the line extraction. To smooth regions, we apply a separated bilateral filter on each pixel of the picture in an oriented way. First in the isophote direction and in the gradient direction. We can apply the filter several times for a smoother result.
The ETF map is used another time for the line extraction process. Because we use a difference of Gaussian in the isophote flow way, we name this process "Flow-based difference of Gaussian". We use the ETF computed to compute a DoG along the edges, determining the most important edges. The algorithm is detailed in sec. 3.4.

Once we have the major lines extracted and the smoothed picture (with edges preserved), we apply a quantization on smoothed picture and both pictures are merged to obtain the final stylized image.

3.2. Edge Tanget Flow

To obtain feature preserving edge flow we use an algorithm called "Edge Tangent Flow". The algorithm gives us for each pixel a vector tangent to the edges. The tangent vector obtained *for one pixel* reflects *the most significant isophote direction* over an area defined by a radius.
The edge tangent flow for each pixel x corresponds to:

$$\hat{t}(x) = \frac{1}{K} \int \int_{\Omega_\mu} \phi(x,y) w_s(x,y) w_m(x,y) w_d(x,y) t(y) dy \tag{1}$$

With:

- Ω_μ a circle of radius μ
- x the pixel corresponding to the center of Ω_μ and y the current pixel
- $\hat{g}$ the normalized gradient magnitude.

- $t(x)$ the tangent vector corresponding to the isophote (which is orthogonal to the gradient).

The spacial weight function. In the computation, we ignore the pixel outside the circle with the following equation:

$$w_s(x,y) = \begin{cases} 1 & \text{if} \quad ||x-y|| < \mu, \\ 0 & \text{otherwise} \end{cases}$$

The magnitude weight function allows us to give more weight to the pixel whose gradient magnitude are higher than that of the center x.

$$w_m(x,y) = [\hat{g}(y) - \hat{g}(x) + 1]/2$$

The direction weight function allows giving more importance to pixels which have similar orientations:

$$w_d(x,y) = |t(x) \cdot t(y)|$$

To take into account correct orientation, we use the sign of vectors:

$$\phi(x,y) = \begin{cases} 1 & \text{if} \quad t(x) \cdot t(y) > 0, \\ -1 & \text{otherwise} \end{cases}$$

ETF can be applied iteratively to give a smoother vector edge map. You can notice in the computation of (1), that we are using the tangent corresponding to the isophote. For the first iteration this tangent is the orthogonal vector to the gradient.

3.3. Flow-based Bilateral Filter

Flow based bilateral filtering is basically a separable filter following two directions. One linear filter is following the edge and a second linear filter is following the gradient direction. The first linear filter is defined by the following equation.

$$C_e(x) = \frac{1}{K_e} \int_{-S}^{S} I(c_x(s)) G_{\sigma_e}(s) h(x, c_x(s), r_e) ds$$

where c_x denotes the flow curve of ETF. Two weights are used in this filter. The first one, denoted $G_{\sigma_e}(s)$ is the spatial weight. It is a Gaussian function following the direction of the edge c_x. The parameter σ_e determines the size of the kernel S. The second weight is a weight of similarity denoted $h(x, c_x(s), r_e)$. As a bilateral filter it compares the similarity of the center pixel with other pixels, but in our case it compares with the pixels along the axis c_x. The similarity weight can be used on colored pictures by using a distance metric in the color space.

The second linear filter is quite similar but in the gradient direction denoted $l_x(t)$.

$$C_g(x) = \frac{1}{K_g} \int_{-T}^{T} I(l_x(t)) G_{\sigma_g}(t) h(x, l_x(t), r_g) dt$$

For each equation the terms K_e and K_g is a normalization term defined by:

$$K_e = \int_{-S}^{S} G_{\sigma_e}(s) h(x, c_x(s), r_e) ds$$

$$K_g = \int_{-T}^{T} G_{\sigma_g}(t) h(x, l_x(t), r_g) dt$$

These two linear filters have different goals. First, C_e will preserves the edges and clean the shapes. C_g operates in the regions interior and suppress the differences between the colors. We apply this two filters several times one after the other. After several iteration we can obtain a smoothed picture like the one in Fig. 3.

Fig. 3. Fbl, $\sigma_e = 2$, $\sigma_g = 2$, $r_e = 50$, $r_g = 10$, 4 iterations

The ouput image after the FBL part is smoother but the amount of colors or gray levels is too high for a good non-photorealistic render. The regions need to be flattened. So we apply a quantization on the luminance. We use Kmeans to quantize the luminance. Using Kmeans allow us to preserve the main class of luminance and respect the luminance of the original picture. The Fig. 4 is the quantization step after the Fig. 3.

3.4. Flow-based Difference of Gaussians filter

The Flow-based difference of Gaussian is a filter which allows line extraction corresponding to the most significant edges inside a picture. As you can notice in the section title, we use a difference of Gaussian along a flow. This flow is

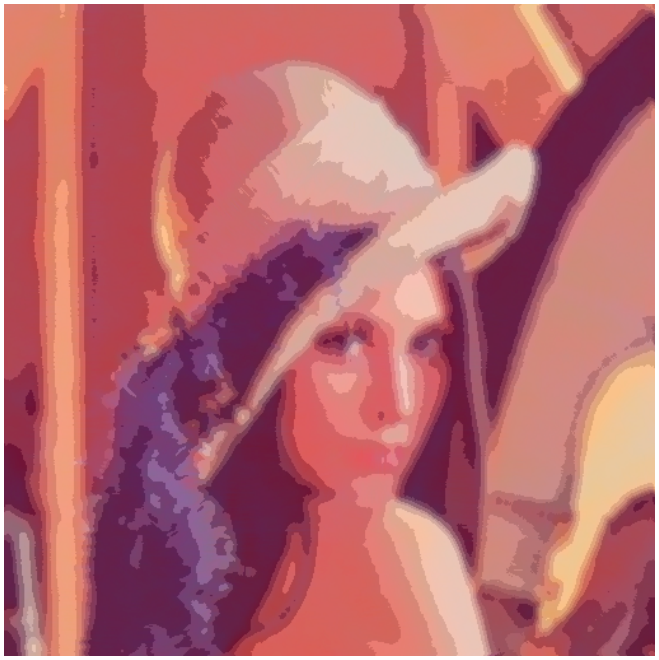

Fig. 4. Fbl quantized with 8 classes

provided by the previously computed ETF (see sec. 3.2), indicating in which pixels we must compute the DoG.
The DoG is computed perpendicular to the edge (i.e. in the gradient way). As we move away from origin pixel, the DoG computed becomes less important.

For each pixel x of grayscale picture I, the FDoG is computed as follow:

$$H(x) = \int_{-S}^{S} \int_{-T}^{T} I(l_{x,s}(t)) f(t) G_{\sigma_m}(s) dt ds$$

The different terms of the equation are explained:

- $G_\sigma(x)$ is a Gaussian kernel of a standard deviation σ following the equation (2)
- S define the size of the curve. The size is computed from the standard deviation σ_c.
- c_x is the flow curve along the edge for the given pixel. The curve is centered by x thus we define $c_x(0) = x$ The curve have a size of $2 \times S$
- s is the point on the curve c_x.
- t is the tangent placed at the point $c_x(s)$. The tangent information is previously computed with ETF.
- T defines the size of the segment $l_{x,s}$. The value of T is computed from the standard deviation σ_s
- $l_{x,s}$ is the line segment perpendicular to $t(c_x(s))$ and parallel to the gradient vector. It as a size of $2 \times T$.
- $f(t)$ correspond to a one-dimensional DoG:
  $f(t) = G_{\sigma_c}(t) - \rho \cdot G_{\sigma_s}(t)$.
  This DoG is computed from the grayscale picture. $f(t)$ is computed for the segment $l_{x,s}$.
- From the standard DoG, $\sigma_s = 1.6 \times \sigma_c$.

The Fig 5 can provide easier explanations. If you paid attention at the above explications, you can obtain different results with five parameters detail in sec. 3.5.

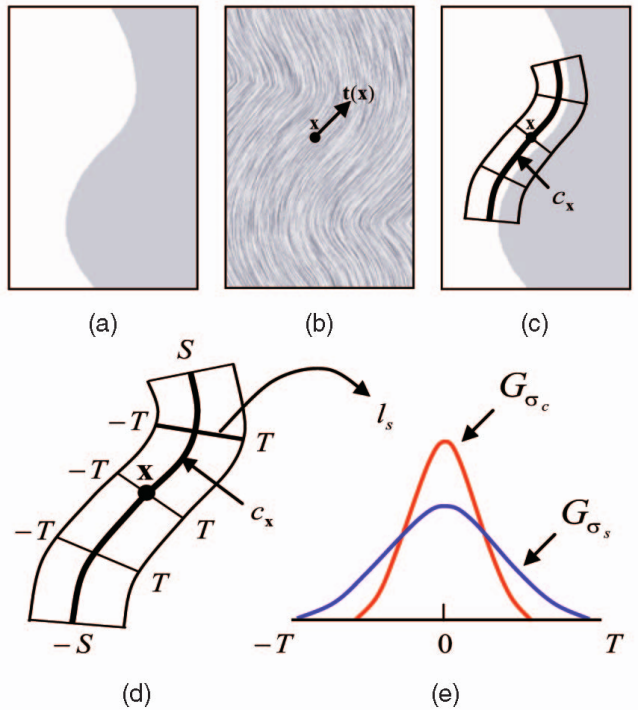

Fig. 5. FDoG filtering. (a) Input. (b) ETF. (c) Kernel at x. (d) Kernel enlarged. (e) Gaussian components for DoG.

3.4.1. Binary Thresholding

We consider the computed value $\mathcal{H}(x)$ as an edge depending of the above thresholding:

$$\tilde{\mathcal{H}}(x) = \begin{cases} 0 & \text{if } \mathcal{H}(x) < 0 \text{ and } 1 + \tanh \mathcal{H}(x) < \tau \\ 1 & \text{otherwise} \end{cases}$$

The result of $\tilde{\mathcal{H}}$ gives a white picture with black edges, like the one in Fig. 6.

3.4.2. From continuous to discrete

The complex part when implementing H were to find a way to know which is the next value to visit along the curve c_x. A pixel as a discrete neighboring of 8 pixels, not a bunch of continuous value.

Fig. 6. FDoG and thresholding. $\sigma_m = 3$, $\sigma_c = 1$, $\rho = 0.985$, $\tau = 0.5$, 1 iteration

To know which value to visit after a particular pixel $c_x(s)$, we add the rounded normalize ETF vector of the pixel to its coordinates. The coordinates of the next pixel p_{s+1} to visit when we are at $p_s = (x, y)$:

$$p_{s+1} = \begin{pmatrix} x + \lfloor t_x(p_s) \rceil \\ y + \lfloor t_y(p_s) \rceil \end{pmatrix}$$

3.4.3. Algorithm complexity

As you can notice, the algorithm complexity is N^3. For each pixel, we compute for each pixel along the curve of size $2 \times S$ the one dimensional DoG of radius size T.
We can decrease the algorithm complexity: The curve c_x depends on the central pixel x for which we compute the fDoG, but the one dimension DoG for a given pixel will have the same value between two curves (because the direction depends on the gradient). Knowing this, we can compute the one directional DoG for all pixels in a first time and sum the DoG values along the curve c_x regulated by G_{σ_m} in a second time. The algorithm now has a complexity of $2N^2$.

3.4.4. Iterative FDoG

The FDoG can be applied iteratively, increasing the strength of the edges at each new iteration. Between each iteration of the FDoG, we *superimposing* the computed edges on the grayscale picture. To superimpose the picture, we superpose the black founded edges on the grayscale picture and we smooth it with a Gaussian Blur. More edges will be found in the following iteration, resulting in stronger edges as you can notice in Fig. 7 compared to Fig. 6.

Fig. 7. $\sigma_m = 3$, $\sigma_c = 1$, $\rho = 0.985$, $\tau = 0.5$, 2 iterations

3.5. Parameters

In this part we describe the way you can have different results by modifying the algorithm parameters.

For the Edge Tangent Flow, we can modify the kernel radius and the number of iterations. As a reminder, the ETF will reflect for one pixel the most significant isophote direction over an area defined by a radius. Thus a bigger kernel value will give more global isophote direction and a small kernel value will give a more detailed map.
The number of iterations impacts the smoothness of the result, but 2 or 3 iterations are sufficient.

For the flow-based Difference of Gaussian, you have five different parameters:

- σ_m whose define the size of the curve. A bigger σ_m will lead to a bigger curve radius and the longest edges will be extracted.
- σ_c defines the strengh of the edges. If we give a bigger value, the segment size T will be longer and more values are taken. Two neighbour pixels will be considerate as edges and be extracted.
- ρ is the noise sensitivity between $[0.97, 1]$. A bigger value will give more important value for the DoG resulting of a biggest sensibility.
- The number of iterations. See the sec. 3.4.4 for more informations.

- The τ parameter use for the thresholding (see sec. 3.4.1)

For the Flow-based Bilateral Filter, we have five different parameters too:

- σ_e defines the curve size $2 \times S$ for the bilateral filter along the edge. A big value will smooth a long curve.
- σ_g defines the segment size $2 \times T$. A important value smooths in the gradient way a long segment.
- r_e set the standard deviation for the *similarity weight* function along edges. An important standard deviation gives an important kernel size and important value to the similarity weight function thus more important smoothing in the edges direction.
- r_g set the standard deviation for the similarity weight function along gradient. A value smaller than r_e allow to smooth more aggressively in the edge direction, preserving the details around the edges compared to a classic bilateral filter as you can see in Fig. 3.
- Number of iteration. During one iteration we pass the bilateral filter in the gradient way and in the edge way. Four or five iterations are sufficient to smooth the picture.

For the Quantization, the number of classes depends of the wished result. To obtain a cartoon look-a-like image, the best option is to use a low number of classes (like 4). To obtain an image with more details, the number could be higher, we advise to choose number of classes between 8 and 16. Higher than 16, the luminance quantization create too much color. The number of classes and the amount of colors obtained are different. The resulting picture will have more colors than the number of classes.

4. RESULTS

The different results are present at the end of the document with originals pictures. Some of them are personal photographies, do not broadcast them. The table 1 gives the parameters used.

5. CONCLUSION AND FURTHER IMPROVMENTS

To go further, we tried to apply this algorithm to videos. In a sequence, the shapes in the scene are relatively consistent frame by frame. So the lines that we extract for a frame are close to the previous lines. For the color, their variantions our eyes detect is due to the luminance. Because of quantization, the variation of colors is low and consistent frame by frame. In order to reproduce an old cartoon effect on the video, we reduced the frame rate (divided by 5).

We could have better drawing result using a saillant map or a depth map. With one of those maps, we could smooth a specific part of the original picture and use it in our process. With this preprocess, the picture will be less detailed in the smoothed area. Passed through our non-photorealistic algorithm the resulting cartoon will have fewer edges and more flattened color regions on these smoothed areas.

Params/Figure	Fig. 8	Fig. 9	Fig 10
σ_m	3	3	3
σ_c	1	2	0.7
ρ	0.98	0.985	0.99
τ	0.5	0.5	0.5
iteration FDoG	2	2	3
σ_e	2	2	2
σ_g	2	2	2
r_e	50	50	50
r_g	10	10	10
iteration FBL	4	4	4
K-means	8	4	8

Table 1. Parameter used for annexes pictures

6. REFERENCES

[1] Kang Henry, Lee Seungyong, and K. Chui Charles, "Flow-based image abstraction," in *IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS*. IEEE, Jan/Feb 2009, vol. 15, pp. 62–77.

[2] Kyprianidis Jan Eric and Döllner Jürgen, "Image abstraction by structure adaptive filtering," in *EG UK Theory and Practice of Computer Graphics*, Soo Lim Ik and Tang Wen, Eds., 2008.

Fig. 8. Japanese Temple

Fig. 9. London subway

Fig. 10. Intra Muros, Saint-Malo, France