FINAL REPORT

2018
January

Parallel Programing to Style Transfer for Videos

GROUP 36

許睿之 0452310 湯沂達 0552302 彭詩峰 0556525

Abstract

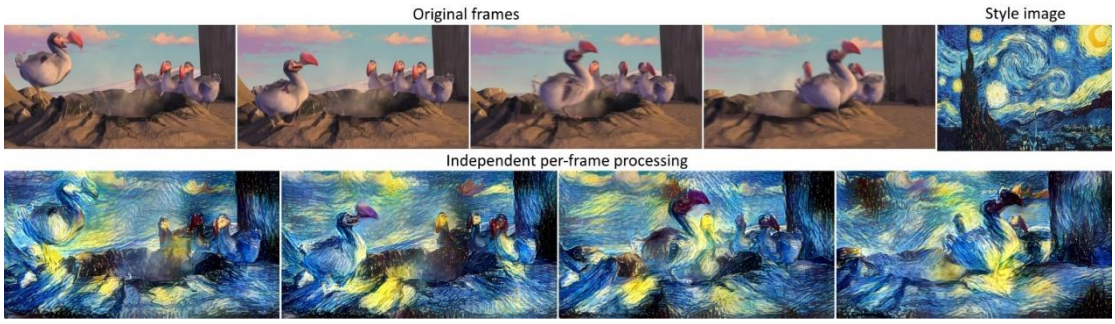
Nowadays, image processing is developing quick. Not only restoration of images gets more information and enhancement of images gets better results, or compression of the images can still save its main information.

In this report, we'll use methods based on mathematics , and it's different from deep learning that extremely quicker than deep learning. Since we just want to transfer only one style, we'll introduce "cartoon style" that use "R.O.F total variation" which is a method to de-noising. And we'll derive the detail of TV method. At last, mainly, there are comparisons of results that make parallel programing to algorithm.

Introduction

Style transfer belongs to the texture transfer problem, the goal is to extract the synthetic texture from the source image, and use the contents of the target image to be constraints. There are many nonparametric algorithms that can be used to reconstruct the natural texture from the source image. The current technology based on the nonparametric algorithm while also using a variety of techniques to retain the structure of the target image.

However, the above problems usually require a high-level deep learning method, which uses large data (Neural Network) or a single image (target image) to obtain a large number of feature blocks through different convolution (Convolution Neural Networks) as training data to train a set of style transfer model, and then detect the source image of the corresponding feature block for style transfer. So that the model with a general analysis of the ability is still quite difficult. But the recent development of Convolutional Neural Network has been able to create a powerful computer vision system can extract the high-level meaning of the image. In addition, CNN that have been specially trained (such as object recognition) by enough tagged images can extract high-level image texture from the generic feature summarized in the training set, and even apply other visual information processing tasks such as material Identification, art style classification and so on.



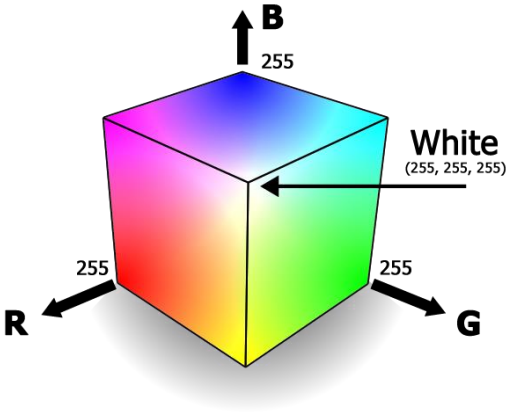
These methods often require a lot of time and data to carry out. Therefore, we think that although there is only one style of transfer, we can find a way that is faster and easier than deep learning, and we can develop a number of different single-style fast algorithms for style transfer and then make a tool transferring different styles.

Solution

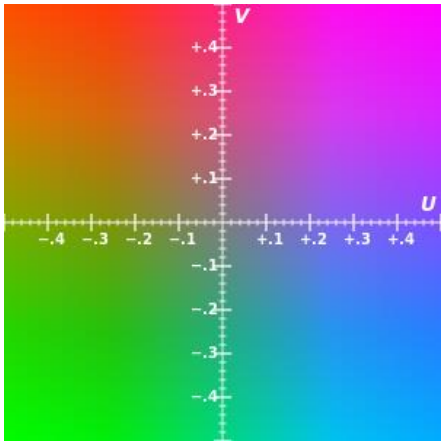
First, we know that video is made of multiple images which are played continuously in a unit of time and audio. That is, a 30 fps video means there are 30 frames per seconds in this video.

Therefore, we can load a video and separate it to several images to do image processing. With several color images (if video is colorful), we need to turn their color space from RGB to YUV, generally, since images in RGB space may have influence when one of channel has some energy changes. This means that if the larger B value on most pixels, the larger R value and G value (in the brightest white point). To modify a channel in RGB space, you need to consider the other two channel changes, so the color change will be more difficult. To solve the problem above, we need YUV color space to help us. YUV is a color encoding system typically used as part of a color image pipeline. It encodes a color image or video taking human perception into account, allowing reduced bandwidth for chrominance components, thereby typically enabling transmission errors or compression artifacts to be more efficiently masked by the human

perception than using a "direct" RGB-representation. The Y'UV model defines a color space in terms of one luminance (Y') and two chrominance (UV) components. The Y'UV color model is used in the PAL composite color video (excluding PAL-N) standard. Previous black-and-white systems used only luminance (Y') information. Color information (U and V) was added separately via a sub-carrier so that a black-and-white receiver would still be able to receive and display a color picture transmission in the receiver's native black-and-white format.



RGB color space



YUV color space

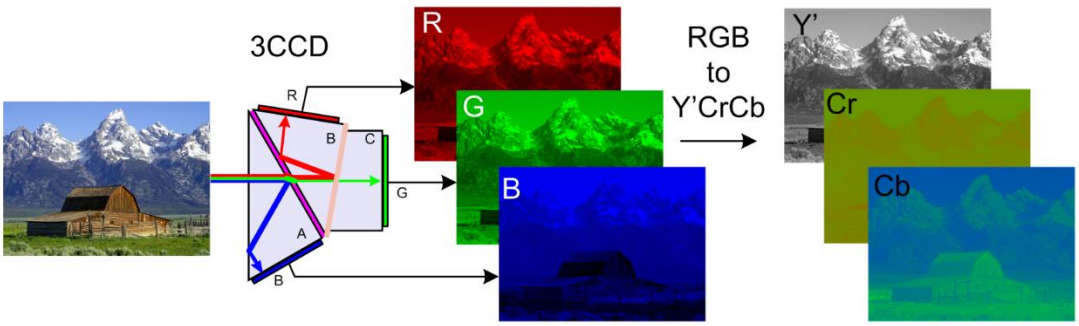
The third axis Y(luminance) is the light energy intensity

Here is the space transformation formula.

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.5 \\ 0.5 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix}$$

where $R, G, B, Y, U, \text{and } V \in [0, 255]$.

The transformation is invertible. We can observe that the Y channel consists of R, G, B in proportion so that we can process only for Y channel. And it will be effective to results in RGB color space.



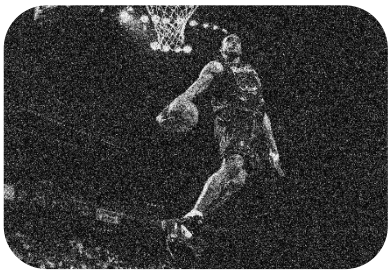
Transformation illustration

Y'CrCb is one of YUV color space

After the color space transformation, we have Y channels to process. Then we can let them to do R.O.F total variation de-noising.

Method

Original



Noise

Denoise 500steps

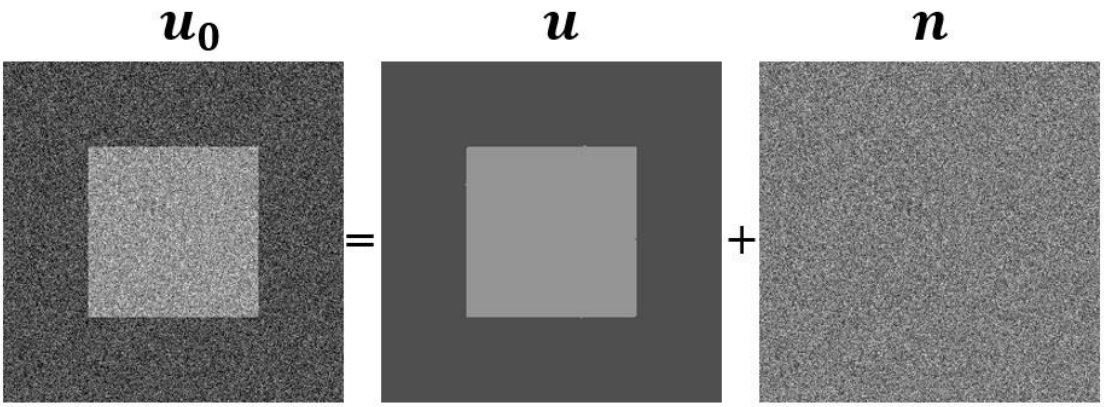
Denoise 1000steps

R.O.F Total Variation De-noise:

Let the observed intensity function $u_0(x, y)$ denote the pixel values of a noisy image for $x, y \in \Omega$. Let $u(x, y)$ denote the desired clean image, so

$$u(x, y) = u_0(x, y) + n(x, y)$$

where Ω is image domain $(Width, Height)$, and n is the additive noise.



When we turn an image into a gradient image, we can see that the gradient of normal image shows the edges of the original image. However, the noise image shows irregular white points and unclear edges. So, when we want to de-noise an image, we have to minimize the intensity of its gradient image. Then we have a minimization problem:

$$\min \iint_{\Omega} \sqrt{u_x^2 + u_y^2}\, dxdy$$

subject to

$$\iint_{\Omega} u\, dxdy = \iint_{\Omega} u_0\, dxdy$$

since the white noise $n(x, y)$ is zero mean and

$$\frac{1}{2}\iint_{\Omega} (u - u_0)^2\, dxdy = \sigma^2$$

where $\sigma > 0$ is given.

With minimization problem and constraints, the variation model is

$$\min_u \iint_{\Omega} \sqrt{u_x^2 + u_y^2} + \frac{\lambda}{2}(u - u_0)^2 dxdy$$

Let $F(x, y, \lambda) = \sqrt{u_x^2 + u_y^2} + \frac{\lambda}{2}(u - u_0)^2$.

Then with Euler-Lagrange equation: $\frac{\partial F}{\partial u} - \frac{\partial}{\partial x}\frac{\partial F}{\partial u_x} - \frac{\partial}{\partial y}\frac{\partial F}{\partial u_y} = 0$, we have

$$0 = \frac{\partial}{\partial x}\left(\frac{u_x}{\sqrt{u_x^2 + u_y^2}}\right) + \frac{\partial}{\partial y}\left(\frac{u_y}{\sqrt{u_x^2 + u_y^2}}\right) - \lambda(u - u_0)$$

with $\frac{\partial u}{\partial n} = 0$ on the boundary of Ω.

The 0 in right hand side of function above would exist while image do not be iterated (stable). Then we have

$$u_t = \frac{\partial}{\partial x}\left(\frac{u_x}{\sqrt{u_x^2 + u_y^2}}\right) + \frac{\partial}{\partial y}\left(\frac{u_y}{\sqrt{u_x^2 + u_y^2}}\right) - \lambda(u - u_0)$$

Thus, the de-noising iteration would be

$$u^{n+1} = u^n + \Delta t \cdot u_t$$

Now, we need to compute λ in each update step. So, recall the equation above, left hand side and right hand side are multiplied $(u - u_0)$ and integral

$$0 = \iint_{\Omega} \left[\frac{\partial}{\partial x}\left(\frac{u_x}{\sqrt{u_x^2 + u_y^2}}\right) + \frac{\partial}{\partial y}\left(\frac{u_y}{\sqrt{u_x^2 + u_y^2}}\right)\right](u - u_0) - \lambda(u - u_0)^2\, dxdy$$

$$\because \frac{1}{2}\iint_{\Omega} (u-u_0)^2\,dxdy = \sigma^2 \Rightarrow \iint_{\Omega} (u-u_0)^2\,dxdy = 2\sigma^2$$

$$\Rightarrow 2\sigma^2\lambda = \iint_{\Omega} \left[\frac{\partial}{\partial x}\left(\frac{u_x}{\sqrt{u_x^2+u_y^2}}\right)+\frac{\partial}{\partial y}\left(\frac{u_y}{\sqrt{u_x^2+u_y^2}}\right)\right](u-u_0)\,dxdy$$

Then we have

$$\lambda = -\frac{1}{2\sigma^2}\iint_{\Omega} \left[\sqrt{u_x^2+u_y^2}-\left(\frac{(u_0)_x u_x}{\sqrt{u_x^2+u_y^2}}\right)+\left(\frac{(u_0)_y u_y}{\sqrt{u_x^2+u_y^2}}\right)\right]dxdy$$

Since the model is continuous, we need to do discretization with finite difference.

Let $x_i = ih,\ y_j = jh,\ i,j = 0,1,\dots,N$, with $Nh = 1$.

The numerical method is as follows:

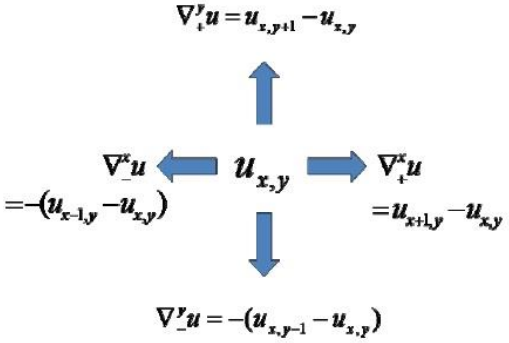
$$u_{ij}^{n+1} = u_{ij}^n + \frac{\Delta t}{h}\left[\nabla_-^x\left(\frac{\nabla_+^x u_{ij}^n}{\left(\left(\nabla_+^x u_{ij}^n\right)^2+\left(m\left(\nabla_+^y u_{ij}^n,\ \nabla_-^y u_{ij}^n\right)\right)^2\right)^{\frac{1}{2}}}\right)+\nabla_-^y\left(\frac{\nabla_+^y u_{ij}^n}{\left(\left(\nabla_+^y u_{ij}^n\right)^2+\left(\mathrm{m}\left(\nabla_+^x u_{ij}^n,\ \nabla_-^x u_{ij}^n\right)\right)^2\right)^{\frac{1}{2}}}\right)\right]$$

$$-\Delta t\lambda^n(u_{ij}^n - u_0(ih,jh))$$

with boundary conditions $u_{0j}^n = u_{1j}^n, u_{Nj}^n = u_{N-1,j}^n, u_{i0}^n = u_{iN}^n = u_{i,N-1}^n$

Remark for the equation above,

$$\nabla_{\mp}^x u_{ij} = \mp\left(u_{i\mp1,j} - u_{ij}\right), \qquad \nabla_{\mp}^y u_{ij} = \mp\left(u_{i,j\mp1} - u_{ij}\right)$$



$$\mathrm{m}(a,b) = \mathrm{minmod}(a,b) = \left(\frac{\mathrm{sgn}(a)+\mathrm{sgn}(b)}{2}\right)\min(|a|,|b|)$$

And the last,

$$\lambda^n = -\frac{h}{2\sigma^2}\left[\sum_{i,j}\sqrt{\left(\nabla_+^x u_{ij}^n\right)^2 + \left(\nabla_+^y u_{ij}^n\right)^2} - \frac{\left(\nabla_+^x u_{ij}^0\right)\left(\nabla_+^x u_{ij}^n\right)}{\sqrt{\left(\nabla_+^x u_{ij}^n\right)^2 + \left(\nabla_+^y u_{ij}^n\right)^2}}\right.$$

$$\left. - \frac{\left(\nabla_+^y u_{ij}^0\right)\left(\nabla_+^y u_{ij}^n\right)}{\sqrt{\left(\nabla_+^x u_{ij}^n\right)^2 + \left(\nabla_+^y u_{ij}^n\right)^2}}\right]$$

Algorithm 1 R.O.F Total Variation Algorithm

Input: Noise Image u_0, *iter*, Δt, σ, ϵ

1: $u_t \leftarrow u_0$, $\lambda_t \leftarrow 1$, $\mathrm{I} \leftarrow 0$, $h \leftarrow$image width

2: **repeat**

3: Compute $\nabla_+^y u_t$, $\nabla_-^y u_t$, $\nabla_+^x u_t$,$\nabla_-^x u_t$

4:

$$u_t = u_t + \frac{\Delta t}{h}\left[\nabla_-^x\left(\frac{\nabla_+^x u_t}{\left((\nabla_+^x u_t)^2\right)^{1/2} + \left(m(\nabla_+^y u_t, \nabla_-^y u_t)\right)^2}\right)\right.$$

$$\left. + \nabla_-^x\left(\frac{\nabla_+^x u_t}{\left((\nabla_+^x u_t)^2\right)^{1/2} + \left(m(\nabla_+^y u_t, \nabla_-^y u_t)\right)^2}\right)\right]$$

$$-\Delta t \lambda_t (u_t - u_0)$$

5:

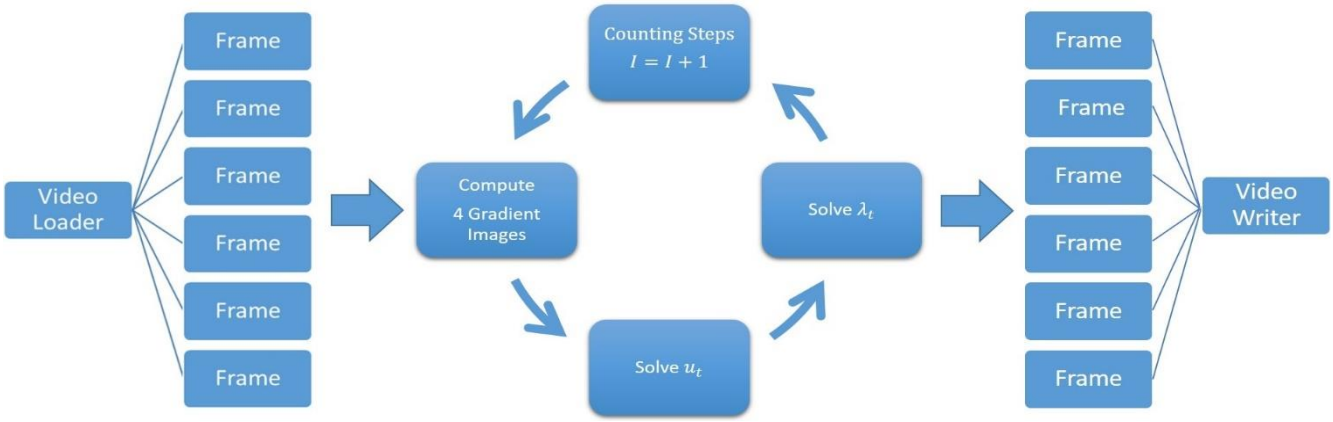
$$\lambda_t = -\frac{h}{2\sigma^2}\left[\sum_{i,j}\sqrt{(\nabla_+^x u_t)^2 + (\nabla_+^y u_t)^2} - \frac{(\nabla_+^x u_0)(\nabla_+^x u_t)}{\sqrt{(\nabla_+^x u_t)^2 + (\nabla_+^y u_t)^2}}\right.$$

$$\left. - \frac{(\nabla_+^y u_0)(\nabla_+^y u_t)}{\sqrt{(\nabla_+^x u_t)^2 + (\nabla_+^y u_t)^2}}\right]$$

6: $\mathrm{I} \leftarrow \mathrm{I} + 1$

7: **until** $\mathrm{I} \geq iter$

Output: Denoise Image u_t



Parallel Program:

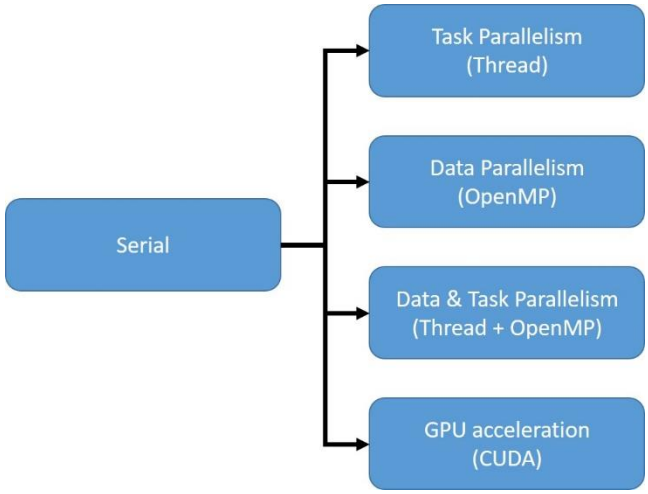
The environment of processing:

CPU: Intel(R) Core(TM) i7-4790 CPU@3.60GHz

GPU: NVIDIA GeForce GT 730

RAM: 16GB

OS: Linux Ubuntu 16.04 LTS



We have four parallel methods:

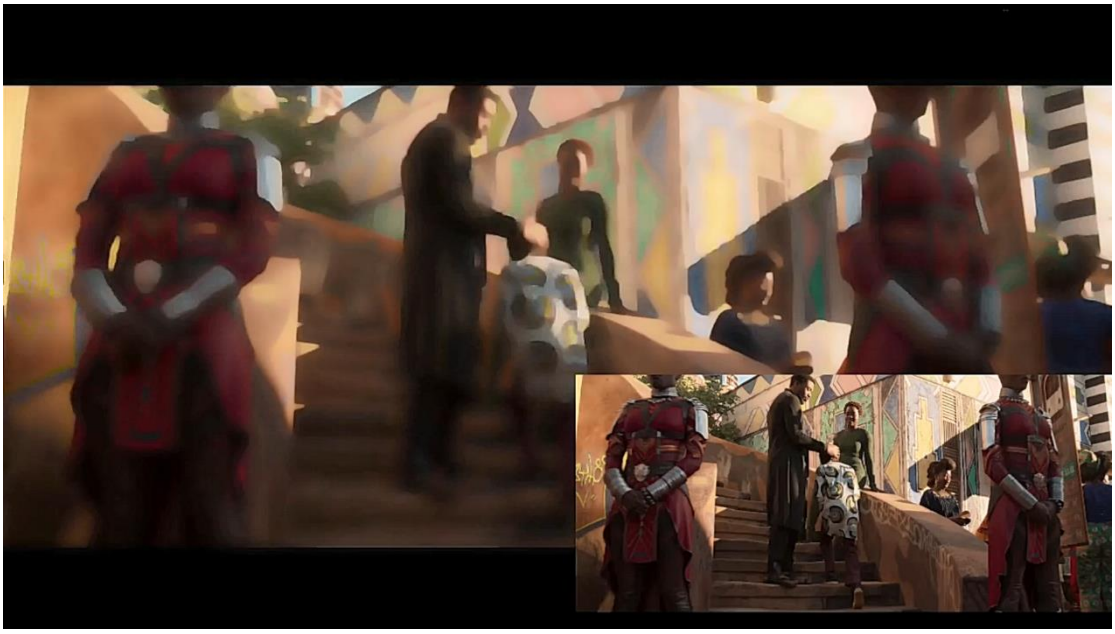
1. Since there are some independent functions. We create threads to run these function for task parallel.
2. There are independent pixels computation can be parallel, thus we can parallel them with OpenMP.
3. Hybrid 1 and 2.
4. Using CUDA

Since there is a function need to sum every element in an array. But it is hard to think the GPU version by ourselves. Thus we refer to the code of "Optimizing Parallel Reduction in CUDA - NVIDIA", it helps us to speed up compute dramatically and don't need to copy data from GPU to CPU.

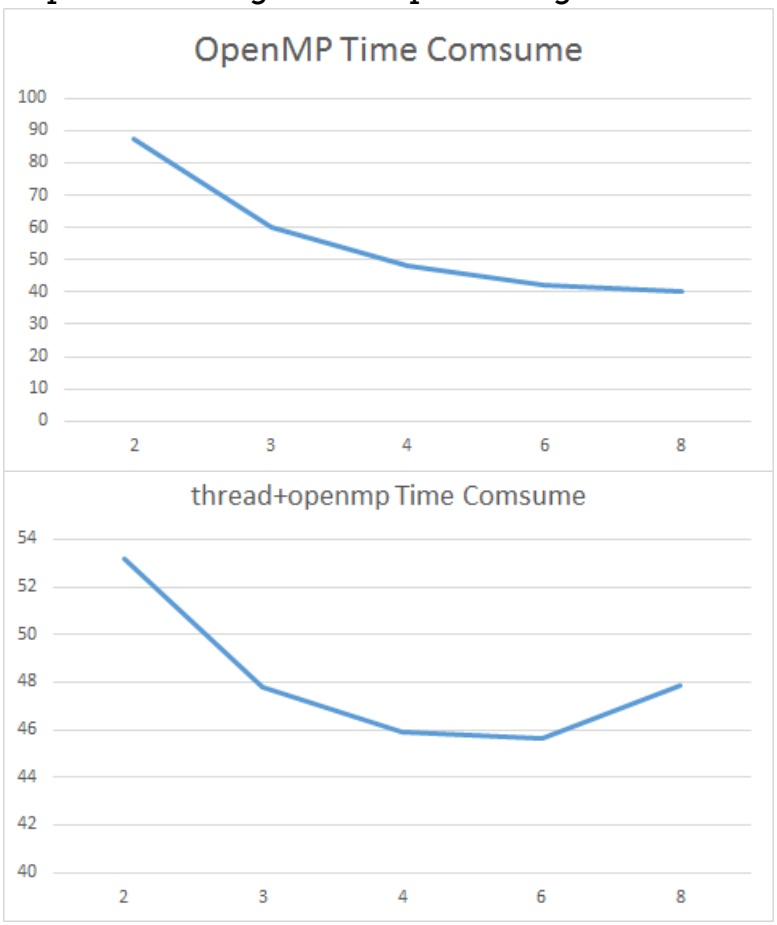
Results

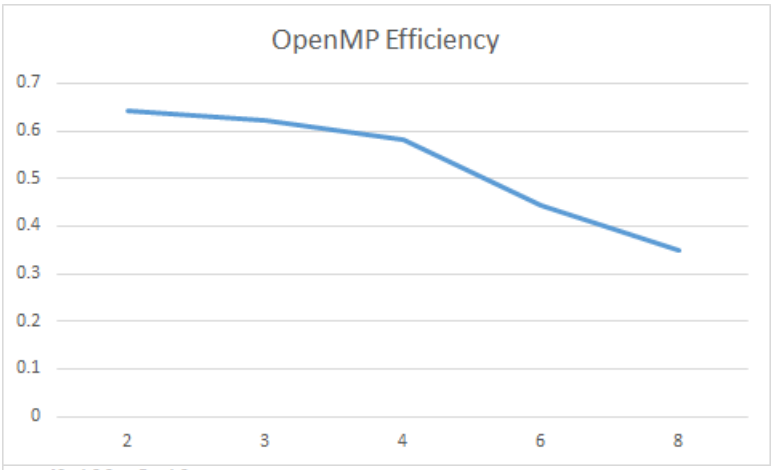
The frame of video after processing.

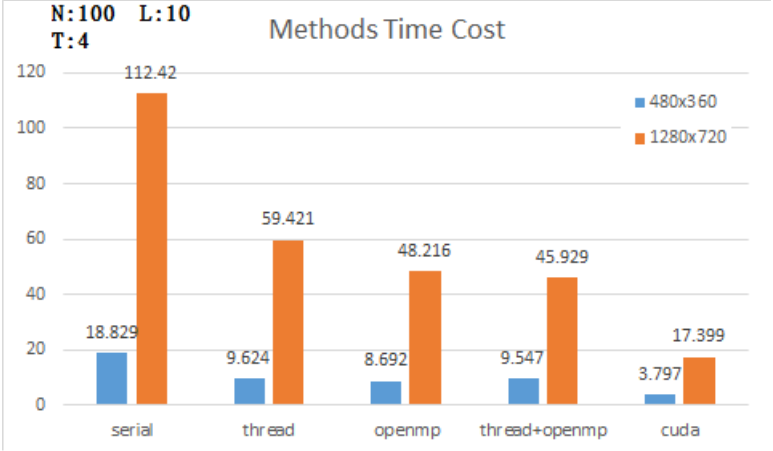
The main screen is processing video, and sub screen is original video.



Here's the comparison among different processing.





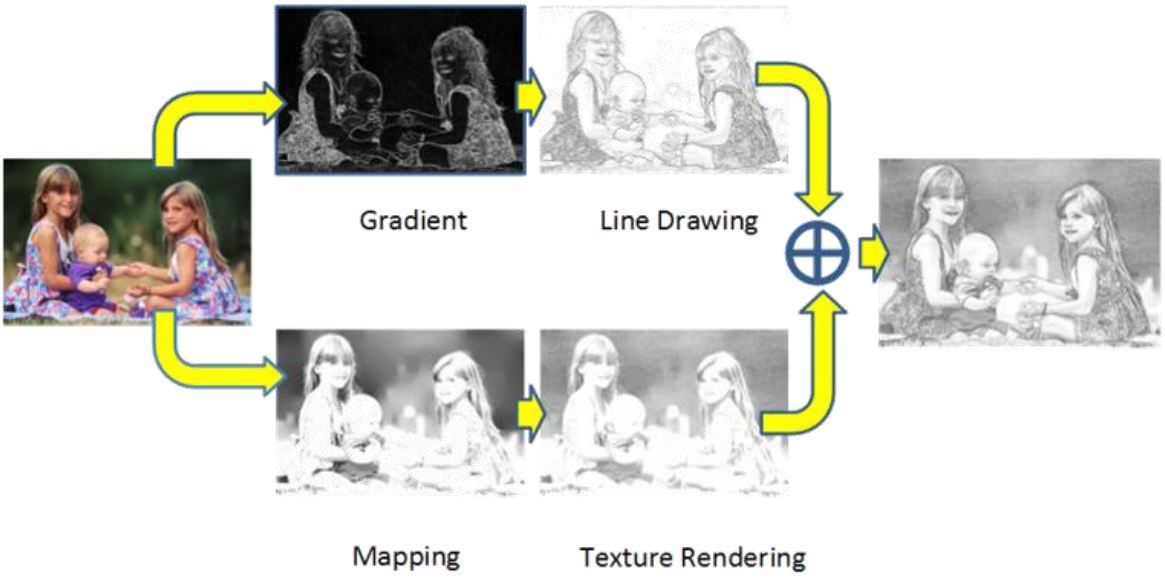


We can notice that the speed of thread+OpenMP is slower than the speed of OpenMP with 480x360 video. In our opinion, the processing is really work, but it increases the time for data transmission.

Related Work

The research read is "Combining Sketch and Tone for Pencil Drawing Production" by Cewu Lu, Li Xu and Jiaya Jia which is a method that produce pencil drawing from natural images. When sketching, we have to outline the contour of the image and construct the overall image which we call the "Structure", and then color the object according to the change in light and shadow which we call "Tone". However, we know that the way of coloring by pencil is to repeatedly slide a pencil in a specific direction and to produce shades of light according the variation of light, then the image has a special "Texture" by pencil drawing. In other words, pencil sketching is composed of "Structure" and "Tone", and finally give "Texture" for a special style.

The result of the research shows the clear edge of image and good effect of style transfer. It's also a good method that uses sizable productions of matrix. After comparing with our method, we find that it's still have room for improvement, such as the edges and the changes in textures. Besides, the method of Cewu Lu, Li Xu and Jiaya Jia uses "convolution" to make images have different changes. We think it's really suitable for parallelization.



Conclusion

We can get very good speed up if we select the right tool. For different problems, we should use different tool. In this case, since we know this computation structure has lots of independent identical computation, it is good for implement it on GPU. And because we want to do style transform of every frame in a video, we can use MPI to separate to different computers (but we don't have these).

Reference

[1] L.I.Rudin, S.Osher and E.Fatemi, Nonlinear total variation based noise removal algorithms. Physica D: Nonlinear Phnomena, 1992, 60(1): 259-268

[2] T.F.Chen, G.H.Golub and P.Mulet, A nonlinear primal-dual method for total variation-based image restoration. SIAM Journal on Scientific Computing, 1996, 20(6): 1964-1997

[3] D.P.Bertsekas, Nonlinear Programming. 2nd ed. Nashua: Athena

Scientific, 1999: 9

[4] A.Chambolle, An Algorithm for Total Variation Minimization and Applications. Journal of Mathematical Imaging and Vision 20: 89-97, 2004

[5] Mark Harris, Optimizing Parallel Reduction in CUDA. NVIDIA Developer Technology

[6] Cewu Lu, Li Xu and Jiaya Jia, Combining Sketch and Tone for Pencil Drawing Production. International Symposium on Non-Photorealistic Animation and Renddering (NPAR 2012), June, 2012