Color Transfer between Images with parallel method

1. The participants: 0556562陳鴻君 0556097陳威翔
2. Abstract

As image-processing technology developing, more and more method appeared. We want to implement “color transfer algorithm”, and use openMP library to speedup. About color transfer algorithm, its purpose is to transfer target image’s color characteristics to source image. Shown as Fig. 1.



**=**

**+**

Result

Target

Source

Fig. 1.

1. Introduction

Image-processing is getting popular nowadays. By the way, you still need to adjust every pixel’s value. For the photographed hardware, such as camera or lens of smartphone, are made in high resolution. Without parallel method for calculating, the traditional way dealing with images one pixel by another serially costs lots of time. Compared with traditional serial way, we use parallel method to make it faster.

We decide to use openMP library to parallelize. Through experiments, we find out that only use parallel for, which is a function provide by openMP library, to parallelize the color transfer program is not fast enough. So we additionally come out the second method that divide the image by its size. After this improvement, the result speedup is raised from 2 to 2.4. Another issue, in our mind, the speedup after the program parallelized depends on the number of threads we used. Actually it is not. We use eight threads to parallelize but the speedup is approximately 2.4. We will discuss the reason in conclusions paragraph.

1. Proposed Solution

Color transfer is a method that borrows one image’s color characteristics from another. There are two images necessary, source image and target image. Source image is the original standard image and the color of target image intends to be altered.

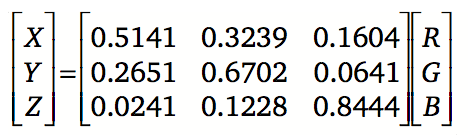
For color transfer algorithm, it has three mainly steps:

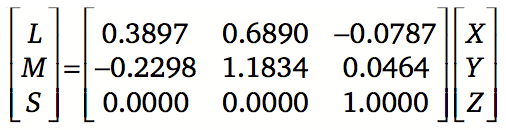
Step1: Forwardly convert pixels in the RGB color space to the 𝒍𝜶𝜷 color space.

Step2: Statistically process each pixel of the image in the 𝒍𝜶𝜷 color space and generate the corresponding result pixel.

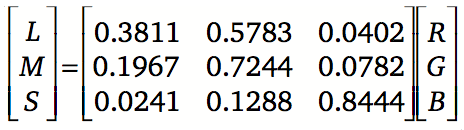
Step3: Reversely convert result pixels back to the RGB color space.

In step1, we want to convert each pixel in RGB color space to 𝒍𝜶𝜷 color space for both source and target images. It has four sub-steps RGB to XYZ, XYZ to LMS, LMS to **LMS**, and **LMS** to 𝒍𝜶𝜷.

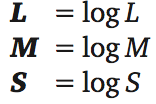
 (1)

 (2)

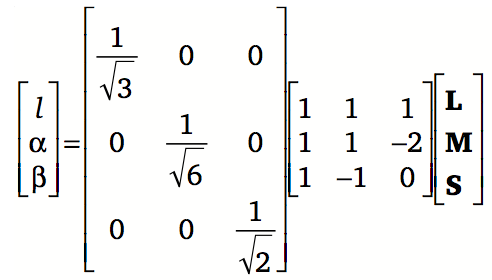
Combining these two matrices gives the following transformation between RGB and LMS cone space:

 (3)

The data in this color space shows a great deal of skew, which we can largely eliminate by converting the data to logarithmic space:

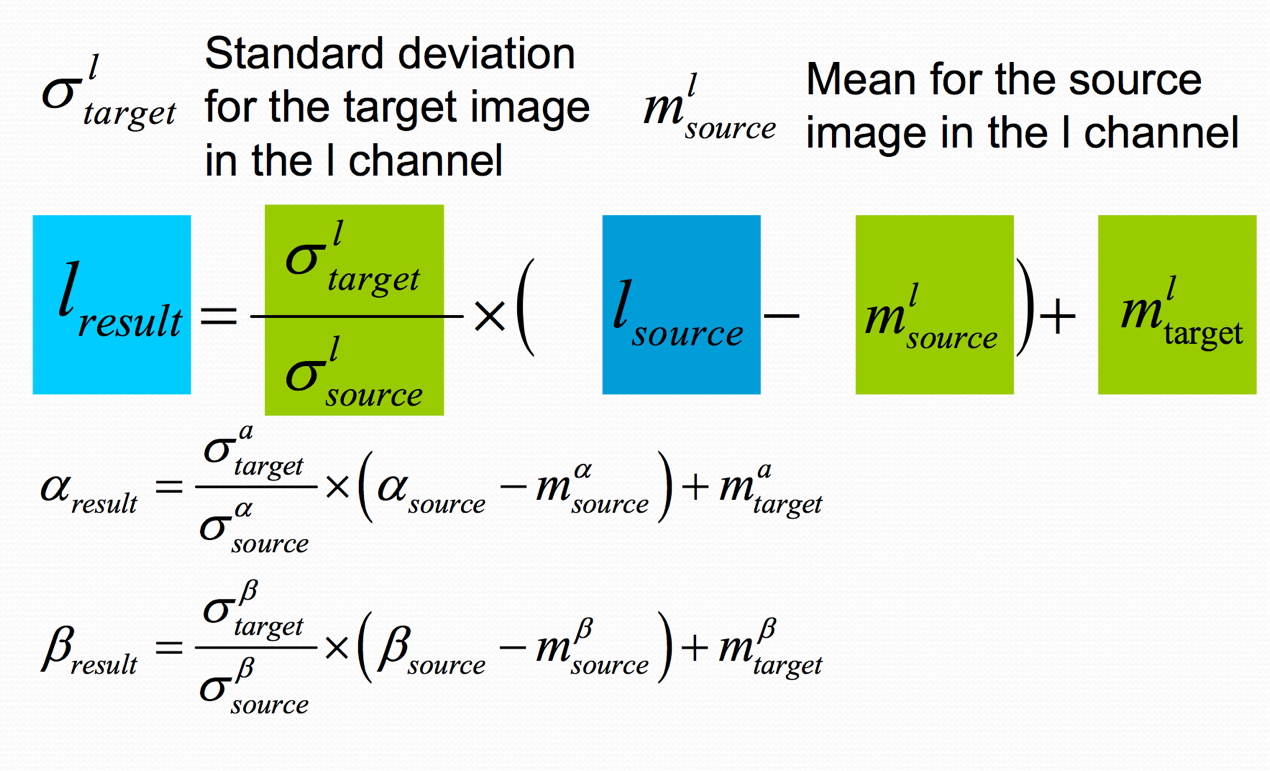
 (4)

The last sub-step is **LMS** to 𝒍𝜶𝜷:

 (5)

Then, complete all sub-step for RGB to 𝒍𝜶𝜷.

Before statistical pixel processing step, we have to calculate standard deviation and mean for both source and target image. Then follow the formula below:

(6)

The final step in color transfer algorithm is reverse color space conversion. It is simple reverse matrices multiplication, so we do not show the detail here.

1. Experimental Methodology

Here, we proposed two parallel method to speed up the program.

In the first approach, we use the ‘parallel for’ directive in openMP to parallelize all the for-loop in the program naively.

In the second approach, we divide the image into multiple tiles, which were assigned to each thread to execute the color space transfer in and between images. By this approach, we might reduce some overhead between threads when calculating the mean value and standard deviation of the R, G, B channel.

After complete coding the two version of our color transfer program with the additional parallel method, we put our program onto the machine to run the program. Our program is run on an i7-6800K CPU, 32GB RAM computer, which contain 6 cores and support up to 12 threads.

As for the input data, we used various sizes of images, such as 512\*512, 1024\*1024, 2048\*2048, 4096\*4096.

We use 1, 2, 4, 8, 12 threads to run our first version respectively. However, for the second version, we was limited by the size of our input data, which we can only use the number of threads that could divide the size of our input images. So we can only test for 1, 2, 4, 8 threads for the second version of our program. For the performances and speedups of our testing program, we will show in the following experimental results section.

1. Experimental Results

To compare method 1 & 2, we use a pair of images whose size is 512\*512 and test them by using different threads. For method 1, because of simply using parallel for function, we can test any number of threads. Depending on our equipment, CPU thread bound is twelve, so we use 1, 2, 4, 8, 12 threads separately. Besides, in method 2, the maximum number of threads is eight limited by the algorithm design. The number of threads must be divided by size, so 12-thread test is not suitable. We use 1, 2, 4, 8 threads to test. Fig.2 and Fig.3 show the comparing result below.

Fig.2

Fig.3

As it shown in Fig.2 & 3, the execution time of method 2 (divide by size) is extremely lower than the execution time of method 1 (use parallel for). And the speedup of method 2 is much higher than method 1 either. So our method lifts the parallel performance compared to simply use parallel for.

From the opposite, the scalability of method 2 if less than method 1, because it can not use any threads for any size. In the other hand, simply use parallel for is fixable for any threads and size. Maybe in future work, we can modify our algorithm to enhance its scalability and performance.

We also use method 2 to compare images in different size, such as 512\*512, 1024\*1024, 2048\*2048, 4096\*4096.

Fig.4

Fig.5

In Fig.4 & 5, they show the performance by different size. For the part of execution time, the estimation time of 4096\*4096 is the best. It seems decreasing by size. But for speedup, different size are similar. It seems decreasing by size indistinctly.

1. Related work

Color transfer is actually a popular topic in image processing. There were so many scholar who choose image processing as topic and especially focus on the color transfer or even borrow the techniques from it. Image colorization, which adding some color to a gray-scale images, is one of many example that used the color transfer[3]. Also there were others that use the 𝒍𝜶𝜷 color space to chromatic and achromatic.[4]

However, all the article mentioned above are processing the image pixel by pixel, which slow down the overall processing time. So, what we want to do is using the same color transfer techniques but with the help of some parallel language, we’ll be able to borrow one image’s characteristic from another and apply to the target image and also save the computational time.

1. Conclusions

We use openMP to speedup our program, but both method 1 & 2 are trapped by the synchronization of threads. For prospect and future work, we can deal with this problem or we can use CUDA or openCL to parallelize. Wish the performance will get better.

**=**

**+**

Fig.6

1. References
2. E. Reinhard, M. Ashikhmin, B. Gooch, and P. Shirley,“Color transfer between images,” IEEE ComputerGraphics and Applications, vol. 21, no. 5, pp. 34-41,September/October 2001.
3. IEEE communications letters, Vol. 10, No. 11,pp. 781-783,Nov. 2006.
4. [Przemyslaw Lagodzinski , Bogdan Smolka, Digital Image Colorization Based on Probabilistic Distance Transform, Proceedings of the 13th Iberoamerican congress on Pattern Recognition: Progress in Pattern Recognition, Image Analysis and Applications, September 09-12, 2008, Havana, Cuba](http://dl.acm.org/citation.cfm?id=1431308&CFID=859578092&CFTOKEN=10865031)
5. Jae Hyup Kim , Do Kyung Shin , Young Shik Moon, Color Transfer in Images Based on Separation of Chromatic and Achromatic Colors, Proceedings of the 4th International Conference on Computer Vision/Computer Graphics CollaborationTechniques, p.285-296, May 04-06, 2009, Rocquencourt, France