Patch-Match for Style Transfer

ANUBHAV CHAUDHARY; 2016013 and YASHIT MAHESHWARY; 2016123

ACM Reference Format:
Anubhav Chaudhary; 2016013 and Yashit Maheshwary; 2016123. 2019. Patch-Match for Style Transfer. 1, 1 (April 2019), 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Github Link

1 INTRODUCTION

Adding color to a grayscale images is a neat system that we see in a lot of softwares like Adobe Photoshop, gimp, etc. Most of the color grading software that are used in the industry are proprietary therefore their implementation is not known publicly. Color is usually added to improve the visual fidelity of old images or images which can't be captured with color (electron microscopic images). The process of coloring a grayscale image requires a colored input image which will be used to color a grayscale image depending on the pixel luminance and the neighbourhood statistics of the pixel. This implementation of Style Transfer over the GPU provides an optimized implementation to allow colorization of gray-scale images using existing similar colored images with the Patch-Match Algorithm. [1]

2 LITERATURE SURVEY

Before patch match nearest-neighbor search [3] was used to provide similar patches in an image. However, the cost of computation of finding this patch was high and instead a patch match algorithm was devised.

The Patch Match algorithm is widely used for adding missing patches or moving subjects [4] in an image while replacing the background of the place from where the subject is moved. The algorithm is based on the fact that good patch match can be found by doing random sampling in the image.

But this implementation of the algorithm involves transfer of a colored patch from a colored image to a gray-scale image (that is to be colored).

The concept behind the implementation can be achieved using two methods:

(1) A **colored image**, it's **gray-scale variant** and a **target image** that is to be colored are sent as an input to the algorithm. The algorithm finds the matching patch in the provided colored image and finds a suitable match based on the luminance of both the images, and transfers the entire patch onto the gray-scale image.
(2) A similar setup to the previous method is required. After a suitable patch has been found, the center pixel in the patch is transferred onto the gray-scale image to the corresponding center of the matched patch.

On comparing both the implementations, we found that Method 1 is faster; but it didn't seem to distinguish the boundaries of objects within the image; thus, causing some loss of information in the final image. We decided

Authors' address: Anubhav Chaudhary; 2016013; Yashit Maheshwary; 2016123.

XXXX-XXXX/2019/4-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

to put more emphasis on the quality of the image, as the performance would significantly be improved by implementing the same algorithm on the GPU.

3 ANALYSIS OF THE ALGORITHM

The algorithm requires a **colored image**, it's **gray-scale variant**, and a **target image** that is to be colored. A pixel wise matching approach is used to compare the luminance of the pixels in the converted gray-scale image and the image to be colored. A pixel wise matching approach is used to compare the luminance of the pixels in the converted gray-scale image and the image to be colored. For each pixel in the target image we go through each pixel in the gray-scale source image and try to find the best color for it using patch match algorithm which compares the luminance values of the neighbourhood pixels in a patch (usually of 5x5 size) to determine the best matching pixel to copy the color from.

(1) Complexity of algorithm is $O(n^4)$.
(2) As shown in figure 1 below there is no dependency between any tasks.

Fig. 1. Dependency Graph. Each circle represents a pixel to be colored as a task.

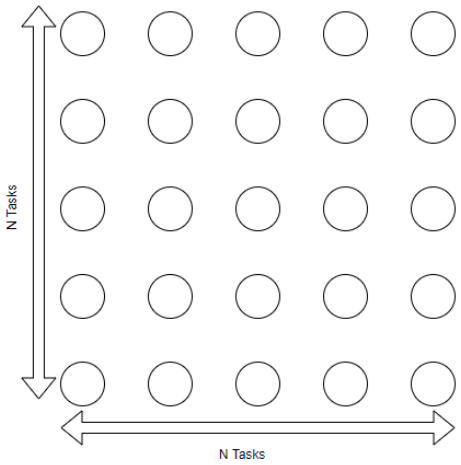


Fig. 2. Pseudo Code

```
for all pixels t_p in target_img:
  for all pixels s_p in src_gray_img:
    bestMatch = s_p
    if (PatchMatch(s_p, t_p) < threshhold):
      if (PatchMatch(s_p, t_p) < PatchMatch(bestMatch, t_p)):
        bestMatch = s_p

    copyColorValues(s_p, t_p)
```

4 PARALLELISATION STRATEGY

The complexity of our current algorithm is $O(n^4)$. While running the algorithm on the GPU we can reduce the computations required to iterate over all the pixels of the image. As we have a lot of threads in the GPU we can devote each thread to compute the best match for each pixel in the target image. Since the computation of the best matches for each individual pixel is independent, we can simply parallelize each one of them on the GPU.

We can use different streams to work on different parts of the image depending on the size of input image which would result in further improvement in the speedup obtained.

5 RESULTS[2]

The CPU code was run on a PC with 16 GB of ram and an i7-4710hq processor. The following results were obtained while trying to color various gray scale images of different sizes. The time taken for each result is mentioned in table 1.

5.1 64 x 64 images


Color Input


Grayscale Input


Target Image


Final Image

5.2 128 x 128 images


Color Input


Grayscale Input


Target Image


Final Image


Color Input


Grayscale Input


Target Image


Final Image

5.3 256 x 256 images


Color Input

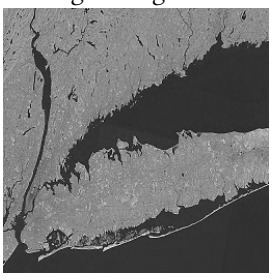

Grayscale Input


Target Image


Final Image


Color Input


Grayscale Input


Target Image


Final Image

5.4 512 x 512 images


Color Input


Grayscale Input


Target Image


Final Image

6 MILESTONES FOR FINAL DELIVERABLE

(1) Implement the same algorithm using a GPU so as to utilize the high number of threads that are present on a GPU
(2) Reduce bank conflicts and maximize memory utilization so as to get maximum possible speedup.

Image Size	64 x 64	128 x 128	256 x 256	512 x 512
Sample 1	2.34s	39.18s	649.70s	7560.58s
Sample 2		40.37s	647.51s	

Table 1. The time taken (in seconds) to color different size images on CPU.

(3) Optimize the code for memory vs speed trade off.

REFERENCES

[1] Tomihisa Welsh, Michael Ashikhmin, Klaus Mueller [*Transferring color to greyscale images*]
[2] Google Images
[3] Nearest Neighbor Search - Wikipedia
[4] Connelly Barnes, Eli Shechtman, Adam Finkelstei, Dan B Goldman [*PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing*]