Artistic Rendering of Digital Images

Serkan Kavak and Adnan D. Tahir
Email: {serkankavak, adt}@eskisehir.edu.tr
Department of Electrical and Electronics Engineering
Eskisehir Technical University, Turkey

***Abstract*— Artistic rendering is a subset of non-photorealistic rendering with the intention of using artistic effects in rendered images. This paper describes a texture transfer method to render digital images in artistic styles. Texture transfer algorithms transfer the source texture to a target image. Since artistic style is subjective, this algorithm has three adjustable parameters that allows a user to change the effect of the style in the output image. This paper also demonstrates some instances for the parameters and output images.**

I. INTRODUCTION

Non-photorealistic rendering (NPR) is an area of computer graphics that focuses on enabling a wide variety for digital art. In contrast to traditional computer graphics, which has focused on photorealism, NPR is inspired by artistic styles such as painting, drawing, technical illustration, and animated cartoons.[1] Artistic rendering, a subset of non-photorealistic rendering, allows us a great ability to focus the attention of the viewer with creating stylist effects. Nowadays, artistic rendering can be used to turn everyday photos into artworks.

While there are many different methods for artistic rendering, most of them are specialized into a specific artistic effect. However, the purpose of this paper is to create artistic rendering for all types, so a texture transfer algorithm is used due to its generality.

Texture transfer algorithms take an image to be used as a source texture and a target image. The target image is transformed to an output image in such a way that output image is a combination of source image and target image but the target image still keeps the general shape of itself. With generating a texture transfer algorithm, any artistic effect can be seen on the target image with changing the source image. This paper includes Section II, proposed method and details of the implemented algorithm, Section III, results of the algorithm with instance images, and Section IV, discussing conclusion and future considerations.

II. PROPOSED METHOD

The algorithm we implemented is based on fast texture transfer algorithm[2] with some modifications. Fast texture transfer algorithm uses a method called coherent synthesis technique which processes every single pixel individually during the texture transfer process. The algorithm we used is shown in Fig.1. We started by initializing the output image first so that our algorithm can run smoothly on it. Texture transfer part in Fig. 1 is where pixel-by-pixel processes come into work. The final part of the algorithm is creating an output with detected best pixel values.

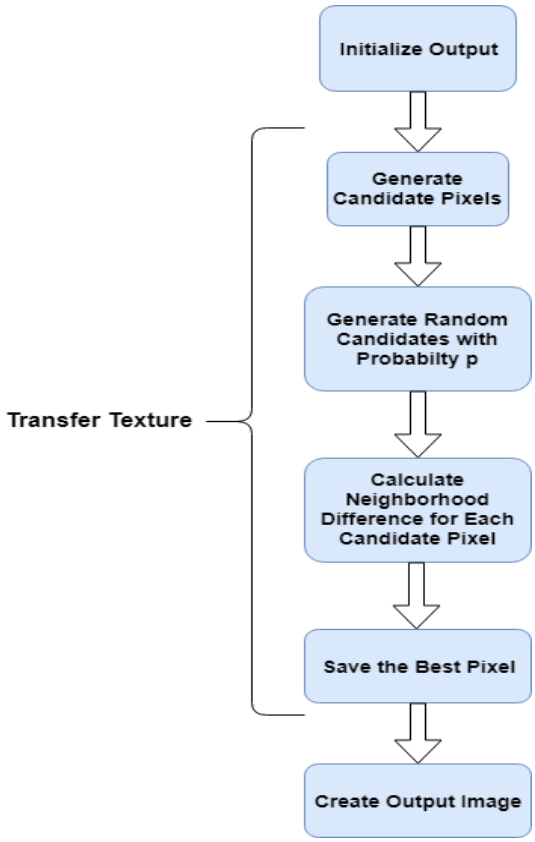

Fig. 1. Algorithm of the work

A. Initialization of Output

The coherent synthesis technique requires that an L-shaped neighborhood of completed pixels should be available for every new pixel synthesized in the output image. In [2], the target image is copied into the output image and the target image uses these values to compute the candidate pixels and the neighborhood difference. However, this can create harsh edges in the output image especially around the image borders. In order to avoid this issue, we created extra pixels to the borders with N/2 width around the output image. This can be look like a padding but since the only concern is having an L-shaped neighborhood, we don't need to add extra pixels to the bottom border. The pixels in these borders

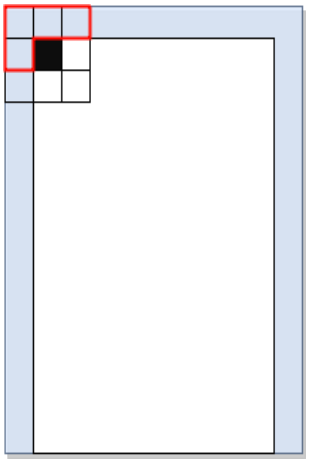

Fig. 2. Initialization of Output Image

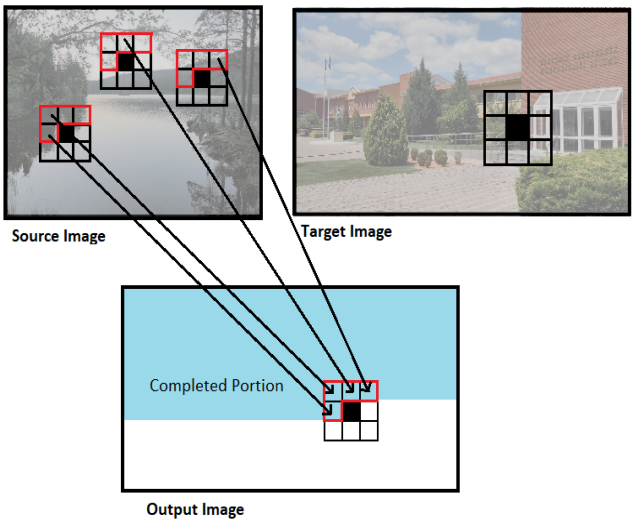

Fig. 3. Generating Candidate Pixels

are randomly assigned from the source image. This allows us to compute the first couple of pixels and edges of the output image. Fig. 2 describes an initialization of output image for a 3x3 neighborhood. The border width would be one pixel on three sides of the image.

B. Texture Transfer

The algorithm synthesizes texture from the source image in a pixel by pixel process. The output array is first initialized in Section II. A. Then, the algorithm goes through the output array in an order. For each pixel, there are four steps (also pictured in Fig. 1):

- Generate candidate pixels
- Add random candidate pixels with probability p
- Calculate neighborhood difference for all candidate pixels
- Save candidate pixels with smallest difference

The algorithm considers a kernel which has NxN size. It surrounds the current pixel to generate new candidate pixels. Fig. 3 shows an example with using a 3x3 kernel size. The output image contains the current pixel in black, 4 completed pixels in the 3x3 kernel in blue, and 4 uncompleted pixels in the 3x3 kernel in white.

The previously completed pixels are looked up in the source image. The new candidate pixels are chosen based on the location of the previous pixel in relation to the pixel currently under consideration. Fig. 3 shows the previously chosen pixels in the source image. The new candidate pixels are in black.

Finding candidate pixels in this way allows the algorithm to grow areas of texture in the output image. However, it can also lead to harsh edges in the output image when the algorithm runs in the boundaries of the source image or changes texture areas. Therefore, a random candidate pixel from anywhere in the source image is added to the candidate list. This reduces the size of the texture growth areas allowing for smoother transitions. While a probability of 0.05 is generally sufficient to reduce these edge effects, greater probabilities may create superior artistic effects.

To find the neighborhood difference for every candidate pixel, we used a similar algorithm to [2] with some changes. The neighborhood difference is calculated using the difference in intensity between the source and the target image and the L2 distance between the completed portion of the output image and the L-shaped neighborhood of the candidate pixel:

$$D^2 = w(\overline{N_s}\text{-}\overline{N_t})^2 + (1/n)^2 L_2(N_{rL}, N_{sL})$$

Ns refers to the neighborhood of the candidate pixel in the source image, Nt refers to the neighborhood of the pixel under consideration in the target image, and Nr refers to the neighborhood of the candidate pixel in the output image. In Fig. 3, the neighborhood size is 3x3, so 9 pixels are considered. The average intensity value (indicated with $\overline{Ns}$ and $\overline{Nt}$) of this neighborhood is used such that no one pixel bears too much weight. This allows the algorithm to pull textures with large intensity variations into a section of the output image. This part of the neighborhood difference is weighted with parameter w. A value of 1.0 tends to work well for most use cases [3]. However, it can be changed for different stylistic effects.

In the second half of the formula, n refers to the number of pixels in the L-shaped neighborhood (e.g., 4 in Fig. 3). This normalizes the importance of the distance between the source and the result. The L2 distance, the Euclidian distance between all RGB pixel values, is taken between the L-shaped neighborhood in the output image and the L-shaped neighborhood in the source image. In Fig. 3, these L-shaped neighborhoods are highlighted with red borders.

When the candidate with the smallest neighborhood difference is found, the RGB pixel values and the location in the source image are stored and the algorithm moves on to the next pixel in the order.

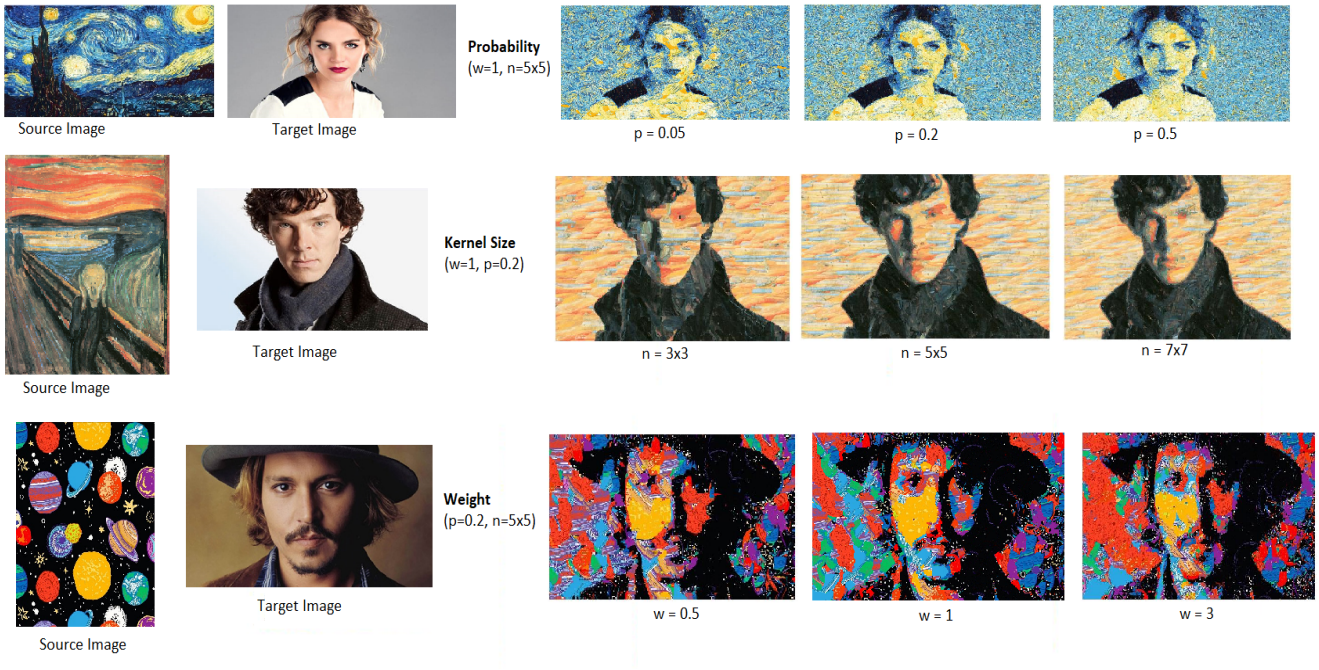

Fig. 4. Variation of outputs to parameters. Image sources [3][4][5][6][7]

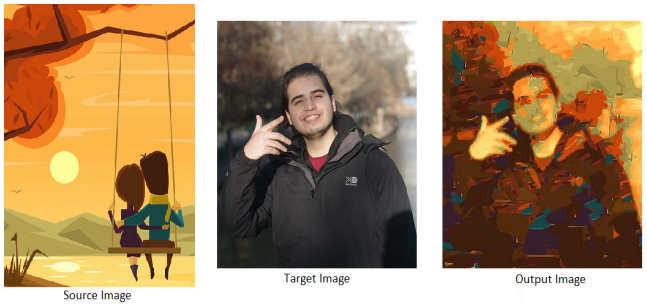

Fig. 5. Another example with parameters: p=0.3, w=3, n=5x5

Fig. 6. Another example with parameters: p=0.5, w=1, n=5x5. Image sources [8][9]

III. RESULTS

There are three parameters that can be adjusted within the algorithm:

- NxN Kernel size (n)
- Probability of adding a new pixel (p)
- Weight on avarage intensity difference between the source and target (w)

The kernel size affects the convergence of the output image as larger kernels provide a greater number of candidate pixels. The effects of kernel size can be seen in Fig. 4. A 5x5 kernel seems to work well for most source images. The 3x3 kernel has too many sharp edges, and the 7x7 loses some of the high intensity change in texture.

The probability of adding a new pixel affects the smoothness of the resulting image. Considering a random candidate pixel causes smaller areas of texture growth. Higher probabilities increase smoothness because the smaller texture areas fit together better, but smaller texture areas mean fewer lower intensity change texture components in the output image. Fig. 4 shows the effects of different probabilities. A probability of 0.2 seems to work well for most images. The 0.05 probability causes the output image to have too many sharp edges, while the output image with the 0.5 probability starts to lose the low intensity change texture components.

The weight on the average intensity difference between the source and the target affects the amount of detail from the target image that is shown. The effects of the weight can be seen in Fig. 4. A weight of 1 seems to work well for most images. The 0.5 weight causes the output image to appear blurry, while the weight of 2 highlights the shape of the target image more in the output image.

Extra examples are added and can be shown in Fig.5 and Fig. 6. From the images, it can be seen that for every different source and target images parameters vary and variation of parameters change the effect of the images considerably.

IV. CONCLUSION AND FUTURE CONSIDERATION

The pictures in Fig. 4, Fig. 5 and Fig. 6 are chosen explicitly to gather better output images. Our algorithm does not work very well to create artistic images all the time.

We tried to process the images iteratively so that at the end of each iteration the best candidate pixel could converge more. We implemented this to our algorithm and get very good results. However, it requires too much execution time (even now an image around 400x400 resolution is processed about in 8-10 minutes and each iteration would multiply this execution time) so we erased it from the algorithm. Further work can be based on a better optimization in the algorithm.

REFERENCES

[1] Wikipedia. "Non-photorealistic rendering". https://en.wikipedia.org/wiki/Non-photorealistic_rendering Last edited on 27 November 2018

[2] M. Ashikhmin, "Fast texture transfer," in IEEE Computer Graphics and Applications, vol. 23, no. 4, pp. 38-43, July-Aug. 2003

[3] Starry Night by Vincent van Gogh. http://wallpaperswide.com/ the_starry_night-wallpapers.html

[4] Tugce Kazaz. https://www.haberturk.com/tugce-kazaz-kimdir-1636689

[5] The Scream by Edvard Munch, 1893 https://www.edvardmunch.org/the-scream.jsp

[6] Benedict Cumberbatch. http://www.bbcamerica.com/anglophenia/2013/09/benedict-cumberbatch-as-an-actor-youre-looking-for-theinfinite

[7] Johnny Depp http://www.beyazperde.com/haberler/filmler/haberler-56212/

[8] Reflection image from homework 2 http://eem.eskisehir.edu.tr/DersDuyuru.aspx?dersId=181&duyuruId=2383

[9] Department image from homework 1 http://eem.eskisehir.edu.tr/DersDuyuru.aspx?dersId=181&duyuruId=2349