

Project: Artistic Rendering of Digital Images

Jyoti Saroha (MT17019)
Ronak Kumar (2015080)
Vishal Gulia (MT17069)

Problem Statement

Non-photorealistic rendering is a booming area of research. Artistic rendering is a subset of non-photorealistic rendering. The main problem we are solving here is to embed the texture of source image into the target image. The input parameters for our program to run are the texture image and the target image into which we want to embed the texture of the source image. Integrating a texture into another image generates a drawing or a sketch kind of effect in the output image. The resultant image is similar to the provided digital image but has got the artistic effects because of the source texture. Our algorithm uses Image Analysis and Texture Transfer techniques like Fast Texture Transfer to efficiently extract the texture from the source image and embed it into the target image.

Individual Contribution

All of us have worked together on understanding the associated algorithms and completing the final code.

To specify: Vishal has worked very hard to understand the algorithms that we were required to use. Then we three collectively converted that algorithm to code. Jyoti did the evaluation and analysis after that. While Ronak worked on developing the Graphical User Interface for the project.

Introduction

The area of computer graphics has mostly been dominated because of its various attempts to model natural phenomenon into digital images. Over the past, a technique known as NPR (Non-Photorealistic rendering) has evolved for improving the clarity and quantity of information conveyed from an image by converting it into visualizations. These illustrations have various advantages as it focuses user's attention to physical features like shape, texture, etc. We found that certain algorithms exist for generating these non-photorealistic images from the input digital images. Past researchers have tried to create drawings which however don't stylistically look like artistic drawings. Using other techniques like neural networks and building the model by training images

only works for a certain set of images which does not appear to be a general solution. These techniques also require a lot of time unless you use a pre-trained model. These algorithms are computationally expensive too. Other techniques like image quilting can be used to generate images which can resemble artistic images but doesn't embed the texture accurately.

Our aim in this project is to implement a general algorithm which embeds the texture of an image into the target image effectively and simultaneously doesn't take a lot of time to run.

Literature Review

Various techniques in the past have been implemented to generate sketch-like effects in normal digital images. Many researchers tried to transform normal camera images to appear like handwritten drawings made by some artists. Efros / Freeman and Hertzmann et al. were some of the first scholars implementing texture synthesis algorithms into images. Hertzmann et al. [1] tried to use certain images as a base to train the code and embed the texture in the new image. It used multiscale autoregression. Efros and Freeman [2] used the stitching/quilting technique to stitch up small pieces to create an image that looks similar to the target image. The problems associated with these older algorithms were that they didn't provide a general solution and were very slow. Another techniques researchers used were pixel modifications or coherent synthesis technique which makes uses of individual pixels rather than using chunks of data each time. The technique Efros used didn't require training , and it didn't consider such large number of individual candidates in the image. Following these algorithms, many fast texture transfer algorithms were proposed which involves fast rendering and embedding of texture into the target images and simultaneously didn't require a lot of computation. Ashminkin [3] in his paper on fast texture transfer described the use of coherent synthesis analysis and candidate pixels for texture transfer. Image enhancement techniques by texture synthesis. Shah and Bhatt [4] used pixel and patch-based texture synthesis technique for avoiding image distortion. Shah and Bhatt [4] used pixel and patch-based texture synthesis technique for avoiding image distortion. Some techniques also used how self-similarity can be used to generate artistic looking images from digital images without involving any texture source.

Dataset

Our program does not require any pre-training as it involves numeric computation on the given input and target images.

Methodology

I. Initialization of the Output Image

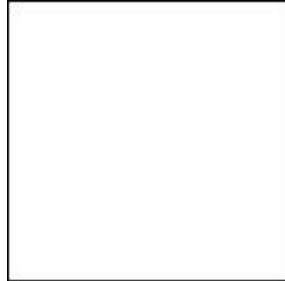


Fig 1 - Source image

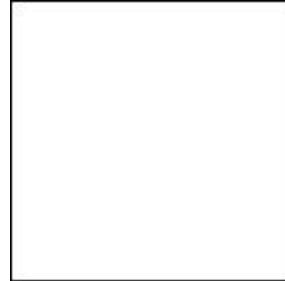


Fig 2 - Target image

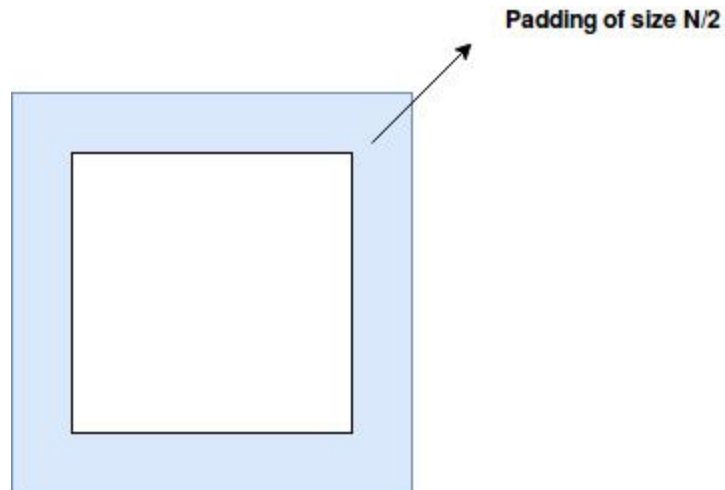


Fig 3 - Padded version of the output image

We will initialize the size of the resulting image with the size of the target image with zeroes. To save the output image from harsh edges and blurred corners, we will do padding at the boundary of the resulting or output image. This padding will be of the $N/2$ size where N is the size of the specified kernel. For filling these unknowns padding pixel values, we will take random pixel values from the source image and fill them using these random values.

II. Generate candidate pixels

Now padded pixels have some pixel values , and all other region of the resulting image is empty totally. We will fill them using these precomputed values. Now if we will consider the next empty pixel in the center of the neighborhood kernel or filter, then precomputed values will make an L-shaped figure as follows -

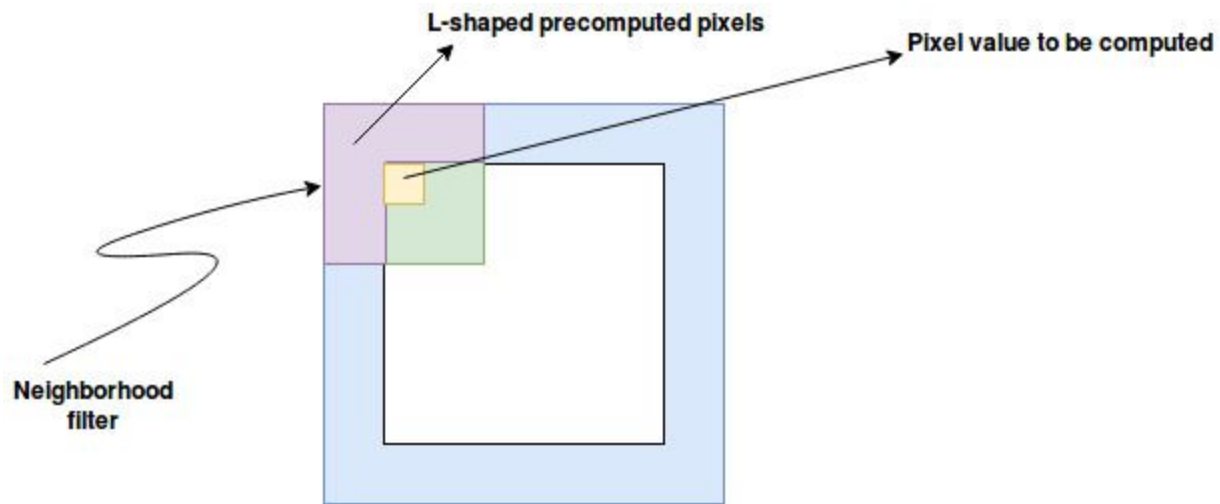


Fig 4 - L-shaped figure

Now with the help of these precomputed pixels in the L-shaped figure, we will determine the pixel value of incomplete pixels. First, we will find the precomputed pixels locations in the source image, then with probability p , we will choose a random location in the source image as candidate pixel. It may be the case that two precomputed pixels have the same neighborhood present in the source image. So we will remove these duplicate candidates and make them a single entity.

III. Calculate neighborhood distance

We have found the location of the precomputed pixel using above steps; now for every candidate pixel found in the source image, we will randomly select the neighborhood of these located pixels in the source image after removing the duplicate candidates.

Now we will calculate the difference between two entities as follows:

1. Average intensity of source image and the target image
2. Distance between the completed portion of the resulting image and L-shaped neighborhood of the candidate pixel.

Formula used:

$$\text{Neighborhood distance} = W * (I^{\text{source}} - I^{\text{target}})^2 + (1/\#\text{pixels})^2 * \text{Dist}(N^{\text{Lresultant}}, N^{\text{Lsource}})$$

Some parameters used in above formula as follows -

- **W** : weight for the neighborhood distance, it will be change for every other style cases.
- **I(source)** : the neighborhood of the candidate pixel in the source image
- **I(target)** : refers to the neighborhood of the pixel under consideration in the target image
- **#pixels** : the number of pixels in the neighborhood kernel or filter, it will normalize the importance of the distance between source and result.
- **N^{Lresultant}** : L-shaped neighborhood in the resulting image
- **N^{Lsource}** : L-shaped neighborhood in the source image

Now the candidate pixels whose neighborhood distance is less than all the computed distances, then we will save this specified pixel into the resulting image.

IV. Texture convergence

Only one iteration is not sufficient for removing harsh edges, blurred and non-mixing texture effects. So we will follow the above steps in every other iteration and update the results in the resulting image. Number of iterations will vary for every other image. After a number of iteration, It will give us better results and our texture will be converged properly.

Evaluation

I. State of the Arts results

Results from [2]:



In this, they have followed two phases, i.e. design phase and application phase. It uses multiscale regression. The input is image A and A', where A' is the filtered out that we desire to get from an image A. In the design phase, a model is trained to filter the image based on inputs A and A'. Now comes the application phase, here the trained model is applied on the input image B. It gives an output B'.

Since this model includes training and regression, this works quite slow. Another drawback in this approach is that this filtering approach can't be generalised for any just effect that we want.



Results from [1]:



The approach used by [1] is of “image stitching”. Though the results that we get from this algorithm are quite well, but at the high of computational time. It considers chunks of pixels at a time to calculate a single pixel in the resultant image. This makes the computational cost too high and can’t be used for user applications.

II. Proposed method results

Fast Texture Transfer Algorithm [3]:

Source image (crayon.jpg)	
Target Image (korean.jpg)	

<p> p = 0.05 m = 1 iterations = 1 n = neighbourhood size </p>	 <p>n=3</p>	 <p>n=5</p>	 <p>n=7</p>	 <p>n=9</p>	 <p>n=15</p>
<p> n = 5 w = 1 iterations = 1 p = probability </p>	 <p>p=0.05</p>	 <p>p=0.1</p>	 <p>p=0.2</p>	 <p>p=0.4</p>	 <p>p=0.5</p>
<p> n = 5 p = 0.2 iterations = 1 w = weight </p>	 <p>w=0.5</p>	 <p>w=1</p>	 <p>w=2</p>	 <p>w=3</p>	 <p>w=5</p>
<p> n = 5 p = 0.2 w = 1 iterations = iterations </p>	 <p>i=1</p>	 <p>i=2</p>	 <p>i=3</p>	 <p>i=4</p>	 <p>i=5</p>

Table 1 - comparison of results by varying parameters

Texture transfer and artistic rendering is a subjective process. It depends on the perception of the viewer that what he/she likes to see. A result might be more appealing to one for some set of parameters while for other viewer it might be good only for other set of parameters.

Maintaining the originality of the target image also depends on the choice of the user. Following is the comparison of the result of the Fast Texture transfer algorithm and results from our proposed algorithm.





 <p>Source Image (t1.jpg)</p>	 <p>Output of Fast Texture Transfer algorithm</p>
 <p>Target Image (girl3.jpg)</p>	 <p>Output of our algorithm (originality=0.25)</p>

Table 2 - comparison of results from FTT and proposed algorithm

III. Discussion and analysis on results

As mentioned before, artistic rendering is a subjective process. For this, this algorithm have five adjustable parameters. These can be adjusted for different results according to the user.

1) **neighborhood (neighborhood)**

This parameter controls the pixels to be considered for the output image. It is observed from Table 1, 3x3 has too many sharp edges. While 15x15 has completely smoothed edges.

2) **probability (p)**

This parameter determines the probability of adding the pixel in the final output. This affects the smoothness of the image. It is observed in Table 1, 0.2 gives better results. A low probability of 0.05 leads to sharp edges, while a high probability of 0.5 smooth out edges and loses low frequency components.

3) **weight (w)**

In formula above, w signifies the importance to the average intensity. $w=1$ has got better results, as low weight is blurring the image. While higher weights are leading to losing low frequency components.

4) **iterations (iterations)**

This depends largely on the perception of the viewer. It also depends on the source texture. More coarse textures converges on higher values of w .

5) **originality**

This defines how much similarity the user want in the output image to the target image. A value of 0.25 seems to work well, as it conserves the properties of the target image in the output image.

Conclusion

In this project we have successfully implemented the texture transfer algorithm with some modifications. With this we were able to get some better results, which are discussed above. Though this is a simple algorithm, but it is sufficient to give considerably good artistic effects to the target image based on the source texture. More complex and computationally costly high algorithms can also be used. This project can be extended by applying to other fields of image processing and domains. Obviously, improving the quality of the output result also.

Acknowledgment

Algorithm in source code:

Developed from the algorithm discussed in [3], the fast texture transfer algorithm. While modifications are our own.

Images:

Images are taken from Google Images, no copyright.

We would like to thank **Prof. Rajiv Ratn Shah** for his constant support and encouragement throughout the project.

References

- [1] Efros, Alexei A., and William T. Freeman. "Image quilting for texture synthesis and transfer." *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 2001.
- [2] Hertzmann, Aaron, et al. "Image analogies." *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 2001.
- [3] Ashikhmin, M., 2003. Fast texture transfer. *IEEE Computer Graphics and Applications*, (4), pp.38-43.
- [4] Shah, Ankit, Parth Bhatt, and Kirit J. Modi. "Image enhancement techniques by texture synthesis." *Int. J. Emerg. Technol. Adv. Eng* 2 (2012): 97-101.