ECE471 - Combining Texture Synthesis and Diffusion for Image Inpainting

1st Misha Roesli
dept. Engineering
University Of Victoria
Victoria, Canada

2nd Carson Reid
dept. Engineering
University of Victoria
Victoria, Canada

3rd Nadia Bernard
dept. Engineering
University of Victoria
Victoria, Canada

***Abstract*—Inpainting is a technique to used to restore damaged images. This paper outlines an implementation of inpainting from the paper "Combining texture synthesis and diffusion for image inpainting" by A. Bugeau, M. Bertalmío. [1]**

***Index Terms*—Image inpainting, object removal, texture synthesis, structure, diffusion, image decomposition, isophotes, geometric inpainting, luminance**

I. Introduction

Inpainting is a technique used to fill in missing parts of an image. This technique may be useful in applications such as image restoration from physically altered photos, or object removal of undesired parts of an image. Inpainting is an important area of research because it can help preserve the integrity of the original artwork, or restore old images. This technique can also be widely used by non-technical people to improve their photo albums and remove unwanted objects from their backgrounds. Lastly, the problem is still far from being solved and could be explored further.

This paper discusses the implementation and validation of the proposed solution in the conference paper "Combining Texture Synthesis and Diffusion for Image Inpainting" [1]. The target application of this implementation is to help remove people or objects from the background of photos. This is accomplished by using photos from an image inpainting dataset [2] which has been used for image inpainting trials and will help with identifying key features of the algorithms such as the smoothness, texture, and structure. Once a photo is selected, 'black masks' will be manually applied to be inpainted by the algorithm to complete the photo. These 'black masks' are a set of pixels with values set to 0. The goal is not necessarily trying to recreate the photo by having the algorithm determine what should exist in the empty region; the intention is to create a photo in which removed parts of an image are replaced with continuations of their surrounding areas.

The implemented algorithm described in the chosen paper such that it is not obvious that something is missing from the photo and that the photo is not significantly distorted on simple object removals. For example, the removed object would be replaced with a continuation of the background.

II. Literary Review

The technique of inpainting images has been around for many years, originally people would attempt to match the style of another artist's painting for restoration purposes. Attempting to fill the missing or worn spaces to create seamless flow. Creating the illusion it was never damaged to begin with. In the last two decades, inpainting has become a digital process. Beginning with simple extrapolation then eventually leading to combining techniques to obtain the best results [3]. Two important methods of inpainting emerge from these: structural and textual. Finally there is a combination of these methods in the paper "Combining Texture Synthesis and Diffusion for Image Inpainting" [1] written by Aurélie Bugeau and Marcelo Bertalmio. This paper is from the Fourth International Conference on Computer Vision Theory and Applications in January 2011. This method is currently one of the most advanced. However, when the image is complex it still does not consistently produce convincing results.

"Combining Texture Synthesis and Diffusion for Image Inpainting" describes inpainting as the reconstruction of missing data of an image. The two inpainting methods discussed include geometric and texture synthesis. Geometric inpainting methods will try to "fill-in the missing regions of an image through a diffusion process" [1] by propagating the information known on the boundary of the missing regions towards the interior. These tend to be computationally expensive since they consist of "finding the global minimum of an energy function, by converging to the steady state [of a] partial differential function" which requires lots of iterations to reach convergence. Texture synthesis methods consist of synthesizing texture samples given some original samples in the image. This helps with propagating the patterns on the edges towards the center of missing regions.

The diffusion process of geometric inpainting often gives "blurred results [but] allows for continuity of the contours" [1], while texture synthesis "permits to conserve the textures but usually fail at preserving the edges and big structures" [1] (edges). The paper decides on combining these two methods since they both improve on each other's weaknesses. This algorithm is an extension of the algorithm from Bertalmio et. al. [4]. The main principle of the algorithm from Bertalmio is to "define weights [for the means of the already calculated image values] such that the image information is transported in the direction of the isophotes" [1]. Isophotes are defined as the contour of equal luminance in an image, or in other words, the edges between structures of different luminance.

The paper suggests a new algorithm that combines two inpainting methods; it combines the algorithm from Tschumperle [5] for the diffusion process and Criminisi et. al's algorithm [6] for the texture synthesis. Tschumperle's geometric inpainting method [5] was chosen over Bertalmio's [4] third-order partial differential equation (PDE) as Tschumperle's produces less blur, is a second-order PDE, and is less computationally expensive. Furthermore, it produces "satisfactory results even after a small number of iterations" [1]. Criminisi's texture synthesis algorithm [6] was chosen over Bertalmio's [3] as it improves on the algorithm by setting an inpainting order for the pixels (depending on the edges and the confidence) and copies an entire patch instead of a single-pixel which is quicker. Setting an inpainting order leads to better results since it fills in the pixels which the algorithm is more confident in first which can help fill in the future pixels.

"Combining Texture Synthesis and Diffusion for Image Inpainting" combines techniques from a variety of papers to obtain the best result; it then explains that the results obtained are satisfactory as there are no discontinuities on the boundary of the mark and textures which are well propagated. [1] However, blurring can still be a problem as shown with their last image example. Further work needs to be completed to improve results.

The rest of this paper will describe the approach, methodologies and mathematics of the implementation of "Combining Texture Synthesis and Diffusion for Image Inpainting", lastly it will evaluate the solution and draw conclusions.

III. Proposed Approach

The following section details the approach of the algorithm, including the methodology of the code and the paper's it is based on. The approach to the project was originally broken down into the following steps:

1) Import image and the image mask to be inpainted
2) Inpaint the image using anisotropic smoothing
3) Decompose the newly inpainted image into a textural and structural image
4) Compute diffusion tensors, eigenvalues, and eigenvectors for the structural image
5) Compute pixel priorities for all pixels in the inpainting mask
6) Inpaint the textural image
7) Sum the structural and inpainted textural images for a final result

The majority of these steps have sub-steps which will be broken down and detailed in the remainder of this section. Also, the changes made to the original implementations will be discussed.

The code was written in Python, excluding a small portion for performing the initial inpainting task of step (2) which is implemented in C++. The image and mask (which is stored as JSON data detailing regions of the image to be inpainted) are first loaded into Python using OpenCV and JSON packages then the mask data is turned into an image of either 0 intensity or 255, where 255-valued pixels are pixels to be inpainted in the original image. The anisotropic smoothing portion of the CImg library was replaced sometime prior to its upload to GitHub meaning the code is not retrievable. There it was replaced with the anisotropic smoothing method of our own implementation. This new method employs *A Fast Spatial Patch Blending Algorithm* which was deemed acceptable to replace the old method. The Python code was originally supposed to make a system call to run the separate C++ code which would load in the image and mask, then use the CImg library to inpaint it and save its output to a new file. During development there were issues with how the CImg library parsed command line arguments that posed difficulties, so a workaround was used where a debugger was paused at the step that would run the C++ code and it was then run from a command line. The structure and texture decomposition is performed by repeatedly applying linear equations for a fixed number of iterations to separate the image into u, the structure portion of the image, and $f - u = v$, the texture component (where f is the input image). The linear equations being applied are as follows:

$$u^0 = f, g_1^0 = \frac{-1}{2\lambda}\frac{f_x}{|\nabla f|}, g_2^0 = \frac{-1}{2\lambda}\frac{f_y}{|\nabla f|}$$

$$H(g_1, g_2) = \frac{1}{\sqrt{g_1^2 + g_2^2}}, \lambda = 0.1, \mu = 0.1$$

$$c_1 = \frac{1}{\sqrt{\left(\frac{u^n_{i+1,j} - u^n_{i,j}}{h}\right)^2 + \left(\frac{u^n_{i,j+1} - u^n_{i,j-1}}{2h}\right)^2}},$$

$$c_2 = \frac{1}{\sqrt{\left(\frac{u^n_{i,j} - u^n_{i-1,j}}{h}\right)^2 + \left(\frac{u^n_{i-1,j+1} - u^n_{i-1,j-1}}{2h}\right)^2}},$$

$$c_3 = \frac{1}{\sqrt{\left(\frac{u^n_{i+1,j} - u^n_{i-1,j}}{2h}\right)^2 + \left(\frac{u^n_{i,j+1} - u^n_{i,j}}{h}\right)^2}},$$

$$c_4 = \frac{1}{\sqrt{\left(\frac{u^n_{i+1,j-1} - u^n_{i-1,j-1}}{2h}\right)^2 + \left(\frac{u^n_{i,j} - u^n_{i,j-1}}{h}\right)^2}},$$

$$u^{n+1}_{i,j} = \left(\frac{1}{1 + \frac{1}{2\lambda h^2}(c_1 + c_2 + c_3 + c_4)}\right)\left[f_{i,j} - \frac{g^n_{1,i+1,j} - g^n_{1,i-1,j}}{2h} - \frac{g^n_{2,i,j+1} - g^n_{2,i,j-1}}{2h} + \frac{1}{2\lambda h^2}(c_1 u^n_{i+1,j} + c_2 u^n_{i-1,j} + c_3 u^n_{i,j+1} + c_4 u^n_{i,j-1})\right],$$

$$g^{n+1}_{1,i,j} = \left(\frac{2\lambda}{\mu H(g^n_{1,i,j}, g^n_{2,i,j}) + \frac{4\lambda}{h^2}}\right)\left[\frac{u^n_{i+1,j} - u^n_{i-1,j}}{2h} - \frac{f_{i+1,j} - f_{i-1,j}}{2h} + \frac{g^n_{1,i+1,j} + g^n_{1,i-1,j}}{h^2} + \frac{1}{2h^2}(2g^n_{2,i,j} + g^n_{2,i-1,j-1} + g^n_{2,i+1,j+1} - g^n_{2,i,j-1} - g^n_{2,i-1,j} - g^n_{2,i+1,j} - g^n_{2,i,j+1})\right],$$

$$g^{n+1}_{2,i,j} = \left(\frac{2\lambda}{\mu H(g^n_{1,i,j}, g^n_{2,i,j}) + \frac{4\lambda}{h^2}}\right)\left[\frac{u^n_{i,j+1} - u^n_{i,j-1}}{2h} - \frac{f_{i,j+1} - f_{i,j-1}}{2h} + \frac{g^n_{2,i,j+1} + g^n_{2,i,j-1}}{h^2} + \frac{1}{2h^2}(2g^n_{1,i,j} + g^n_{1,i-1,j-1} + g^n_{1,i+1,j+1} - g^n_{1,i,j-1} - g^n_{1,i-1,j} - g^n_{1,i+1,j} - g^n_{1,i,j+1})\right].$$

Where pixels outside of the image are indexed, reflection padding is used. To approximate ∇f, kernels

$$k_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}, \text{ and } k_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

are applied to the input image to attain f_x and f_y, respectively.

When applying these equations, the same issue arose in multiple areas. In both the H function and the c variables having both operands under the radical equal to zero (which is possible with completely black areas of the input image) and the paper these equations were from did not detail how to account for this, so a small factor was added to each denominator that could become zero. After 100 iterations the final u is subtracted from f to give both u and v.

The next step is to obtain the eigenvalues of each pixel to determine if there is a need to grab the new pixel intensity from patch. First, the structure image is normalized as it could potentially contain values outside of 0 to 255 for the pixel intensities. Next, the gradients are obtained; however, the main difference is that Gaussian smoothing is not used in the process in order to simplify the math. Once the gradients have been obtained in both the x and the y direction, the values are stacked for each pixel so that each pixel would have an associated vector. To obtain the tensors, the outer product is applied with each channel's gradient vector against itself. This is then summed up together creating a 2x2 matrix for each pixel. This is shown with the equation below as used by Tschumperle in his paper on PDE's [8].

Insight into the local geometry surrounding a pixel is pertinent to the region filling algorithm. To know whether a pixel is in a region that is defining structure of the picture, e.g. a strong edge, can give guidance into whether a pixel from the texture or from the structure is a more suitable choice to paint into the picture being inpainted. Calculating the eigenvectors θ^+,θ^- and their corresponding eigenvalues λ^+,λ^- which point along directions of most and least image variation from a given pixel can provide local structure information. The intensity of both λ^+ and λ^- are important to drawing any conclusions about their values. The calculation of these eigenvalues is as follows:

$$\forall \mathbf{X} \in \Omega, G_{(\mathbf{X})} = \sum_{i=1}^{n} \nabla I_{i(\mathbf{X})} \nabla I_{i(\mathbf{X})}^T$$

where

$$\nabla I_i = \begin{pmatrix} \frac{\partial I_i}{\partial x} \\ \frac{\partial I_i}{\partial y} \end{pmatrix}$$

The intuition behind why this equation is correct is that due to its property of being a semi positive matrix; all values are positive and any vector addition will result in a positive vector. Additionally, there is a linear relationship between the image channels and a single channel ie. RGB to Grayscale.

Once the 2x2 matrices are obtained, the λ^+ and λ^- values can be extracted using Numpy. Pixel priorities were also omitted to further simplify the approach. The texture image is inpainted using the lambda values that are calculated.

The code then iterates over all pixels that still need inpainting, performing the following:

1) Compute the sum of squared differences (SSD) between the 9x9 patch surrounding the pixel (Ψ) and all other 9x9 patches in the source image
2) Select the lowest SSD patch $\bar{\Psi}$
3) $\forall p \in \bar{\Psi}$, determine if the source pixel is a strong structure pixel by comparing the difference in its eigenvalues λ^+, λ^- to the average β of differences in the image $\bar{\Omega}$:

$$\beta = \frac{\sum_{q \in \bar{\Omega}} \lambda^+(q) - \lambda^-(q)}{|\bar{\Omega}|} \tag{1}$$

   If $\lambda^+(p) - \lambda^-(p) < \beta$, the source pixel is a strong structure pixel and when copying from $\bar{\Psi}$ to Ψ the pixel value from the structure image will be used. If the condition does not hold, the source pixel will be from the texture image and the value will not be changed.
4) Copy all pixels from $\bar{\Psi}$ to Ψ and their respective sources (structure or texture, determined in (3))
5) Set all pixels in the mask image corresponding to newly inpainted pixels in Ψ to be 0, marking them inpainted

After all pixels have been inpainted and no mask pixels with a nonzero value remain, recombine the structure and texture images u and v and save the result as the final output.

IV. Evaluation

The dataset used was from the European Conference on Computer Vision [2]. The dataset included images with several black masks applied to them. These removed sections from the images are provided in another file which could later be compared with the inpainted version. The experiments done include peak signal-noise ratio (PSNR) and structural similarity (SSIM) analysis for our inpainted image.

The first result of the algorithm created was for inpainting using anisotropic smoothing. This was tested on one image as seen in Figure 1. As can be observed, general coloring of the missing parts of the image were successfully retrieved. However, there are discontinuities in various part as there was no specific ordering to how each pixel is selected. Generally this means it is smoothed over.

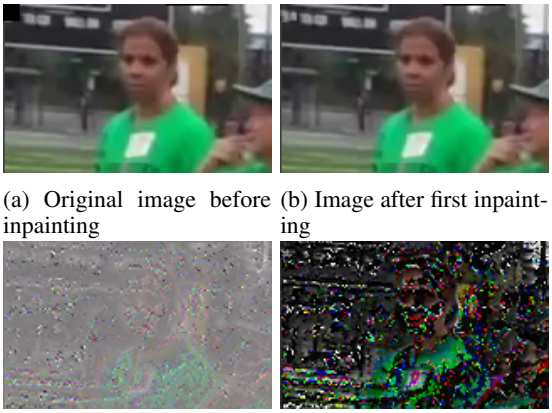
(a) Image before inpainting (b) Image after inpainting

Fig. 1: Mini inpainting the image

Another example is seen below in Figure 2.

(a) Image before inpainting (b) Image after inpainting

Fig. 2: Mini inpainting the image



(a) Original image before inpainting (b) Image after first inpainting

(c) Structure portion of inpainted image (d) Texture portion of inpainted image

Fig. 3: Inpainting the image

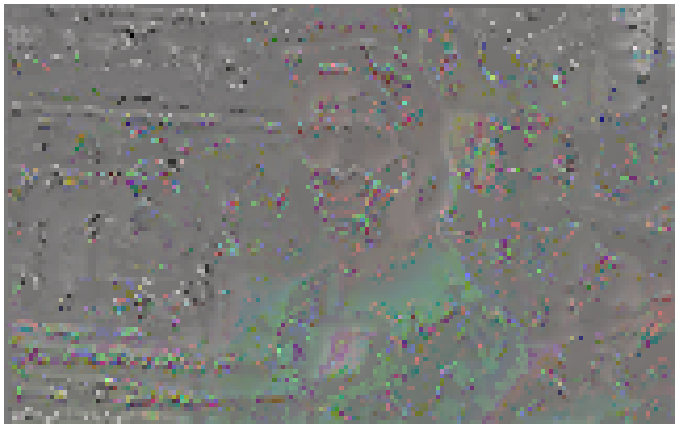


Fig. 4: Final inpainted image

The results obtained were not satisfactory. The images that were produced were gray, and had lots of noise left in them as can be seen in figure 3 below. In part a), the original image is displayed and in part b) the inpainted image image using Tschumperle's anisotropic smoothing method. In part c) and d) we see the resulting structure and texture portions of the final image which is shown in Figure 4.

In computing the structure and texture decomposition of an image difficulties were run into where divisions of intermediary results tried to divide by zero, and the selected workaround may have played a role in the noise introduced to our inpainted images. Since the output of the decomposition was incorrect, the values had to be re-normalized to [0-255] to have a proper RGB mapping. This also definitely played a role in producing poor results.

The calculated SSIM for the original image to the final inpainted result is 0.17863. This is a very low result and indicates very little structural similarity to the original image. The calculated PSNR for our inpainted image to the original is 27.71106. This is a decent result, but normalization and large areas of little information are most likely the culprit.

Overall, the algorithm took about 20 seconds per pixel in the mask to complete, which on average would take about a full day to compute per image. This is unreasonable and likely due to several problems in our code, including workarounds for division by zero, scaling, extreme case which may not be taken into account, and rounding. As can be seen in our final image, structure and texture decomposition could also have been a problem as the final image retained some of the properties from the decomposition.

V. Conclusion

In this paper, there was an attempt to implement an inpainting algorithm that fills missing data from images. Then both structural and textural inpainting was used to produce a clean, realistic image.

Once implemented the computation time was found to be unreasonable; this is believed to be due to potential bugs such as divide by zero errors and scaling issues.

Implementing the algorithm was a challenging experience. The results did not turn out as expected. During implementation, there were multiple challenges that mostly arose in converting mathematical formulae to runnable, correct code. Complications like dealing with division by zero, integer overflow, and long compute times were our biggest implementation

challenges. While the full algorithm did not produce results, results were seen from the individual intermediate steps. These gave us a deeper understanding of the algorithm and the complexity of computer vision as a whole.

REFERENCES

[1] A. Bugeau, M. Bertalmío, "Combining Texture Synthesis and Diffusion for Image Inpainting," In Proc. Fourth International Conference on Computer Vision Theory and Applications, 2009, pp. 26-33. Available: https://www.researchgate.net/ [Accessed: Jan. 31, 2020].

[2] "ChaLearn", Chalearnlap.cvc.uab.es, 2016. [Online]. Available: http://chalearnlap.cvc.uab.es/dataset/30/description/. [Accessed: Feb. 20, 2020].

[3] "An overview of Image Inpainting", International Journal of Innovative Research in Computer and Communication Engineering, 2017 [Online]. Available: https://www.researchgate.net/publication/

[4] M. Bertalmio et al, "Simultaneous structure and texture image inpainting," *IEEE Trans. on Im. Processing*, vol. 12, no. 8, pp. 882-889. Available: https://www.researchgate.net/ [Accessed: Jan. 31, 2020].

[5] Tschumperlé, D, "Fast anisotropic smoothing of multi-valued images using curvature-preserving pde's," *Int. Journal of Comp. Vis*., vol. 68, no. 1, pp. 65-82. Available: https://link-springer-com.ezproxy.library.uvic.ca [Accessed: Jan. 31, 2020].

[6] Criminisi, A. et al, "Region filling and object removal by exemplar-based inpainting," *IEEE Trans. on Im. Processing*, vol. 13, no. 9, pp. 1200-1212. Available: https://www.researchgate.net/ [Accessed: Jan. 31, 2020].

[7] Bornemann, F. and März, T. "Fast image inpainting based on coherence transport," *Journal of Mathematical Imaging and Vis*., vol. 28, no. 3, pp. 259-278, 2007. Available: https://link-springer-com.ezproxy.library.uvic.ca [Accessed: Jan. 31, 2020].

[8] D. Tschumperlé, "Fast Anisotropic Smoothing of Multi-Valued Images using Curvature-Preserving PDE's.", Hal.archives-ouvertes.fr, 2020. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00332800/. [Accessed: 27- April- 2020].