6.865 PS10 Writeup

Victoria Juan (vjuan)

December 2019

1 Introduction

For this PSET, I chose to implement texture synthesis (item 3.2.1) and inpainting item 3.6.6). Specifically, I worked on implementing computational texture synthesis (growing and hole-filling), CNN-based texture synthesis, and computational inpainting. I thought these problems were interesting because cleanly removing objects from photos and then filling in the hole left behind seems like an often-sought procedure, such as if I wanted to remove people in the background of a tourist photo. While I had some intuition for considerations like preserving the shapes and colors in the hole, it also isn't trivial to do manually with basic photo editing tools, so computational methods are particularly useful.

2 Background

Texture Synthesis by Non-Parametric Sampling, Efros and Leung 1999
In this paper, a sampling method is described that produces an imitation of a texture given an image of a texture. Previous papers were not able to produce a good algorithm for both stochastic (random) or regular (like tiles) textures. However, this paper proposes a simple algorithm which can handle both as follows:

1. **Unfilled pixels:** Get a list of all unfilled pixels in an image. For an image with a hole, the list will consist of all pixels in the hole. For a new image being grown, the list will consist of all pixels in the image besides the seed.

2. **Neighborhood window:** For each pixel, get a square window of all neighboring pixels with the pixel in consideration at the center.

3. **Find matches:** From the original image, consider only pixels that are filled. For each pixel, apply a Gaussian blur to the window of neighboring pixels with the pixel in consideration in the center. Then, compare the element-by-element sum of squared differences of neighbors within the windows of the pixel we are trying to fill and the sample pixel from the original image. Also, normalize the sum. After considering all pixels, return only the pixels whose sum of squared distances over their windows are under a certain threshold.

4. **Choose a match:** Randomly select a match. Then, fill the unfilled pixel in with this filled pixel's value.

By using the Gaussian blur over windows and randomizing selection from window comparisons, the authors of the paper model texture as a Markov Random Field, meaning that they assume focusing on local texture is enough to model a cohesive whole texture in the bigger picture, given that the distribution of a pixel's brightness within its neighborhood is independent from the rest of the image.

Texture Synthesis Using Convolutional Neural Networks, Gatys, Ecker and Bethge 2015
The authors describe a generative method to continually update an image initialized as random noise through gradients calculated from a CNN in order to produce an image with the same texture from a separate input image. This separate input image is also run through a CNN with the same inner structure such that comparisons between correlations of features in various layers of the inpupt image guide the weighting of the

other image for texture generation.

In comparison to classical computational texture generating methods, such as the one described by the first paper, this method doesn't require deep analysis or understanding of the structure, coloring and lighting of pixels within an image. Its setup is also relatively simple, since preexisting libraries and pre-trained nets exist online. However, the method takes many iterations to run.

Image Inpainting, Bertalmio, Sapiro, Caselles and Ballester 2000
This paper proposes a method to inpaint masked areas of an image by aiming to connect isophite lines arriving at region boundaries - in other words, connecting lines that should be connected, but which have been disrupted by missing areas of an image. A simplified version of the algorithm (offerd in the a10 handout) is as follows:

1. **Structure:** Calculate the structure of the image via the structure tensor, ignoring the masked parts.
2. **Poisson image editing:** Perform gradient descent (similar to iterative updating) with the Poisson of the structure tensor. The basic goal of this step is to interpolate the structure tensor in the masked regions - i.e. find the gradients (general directions, lines and colors) of the region in the masked regions that are missing from the structure tensor.
3. **Color estimation:** Use anisotropic diffusion with the interpolated structure tensor as a guide to calculate color values from the original input image. Essentially this step involves calculating the differences between this pixel and neighboring pixels based on their structure tensor values (i.e. pixels on the same structure should have the same color) and using this as a weight for getting color values.
4. **Iterate:** Ideally this step should be iterated gradually such that the color and structures grow inward from the boundaries of the regions, and updated structure tensors can be calculated at each step.

The authors state that this inpainting technique is novel because it doesn't require input from users, so that multiple structures can be filled in during the same run of the algorithm, which operates over the whole image and its missing pixels. In comparison to the texture synthesis methods, image inpainting can preserve the structure, such as the lines and shapes, of an image depicting larger, non-repetitive features. However, a downside is that the algorithm doesn't replicate texture of areas it inpaints, so large areas to be filled will look overly smooth.

3 Implementation and Results

Texture Synthesis, computational The methods I implemented can be mostly found in a10.cpp/a10.h.

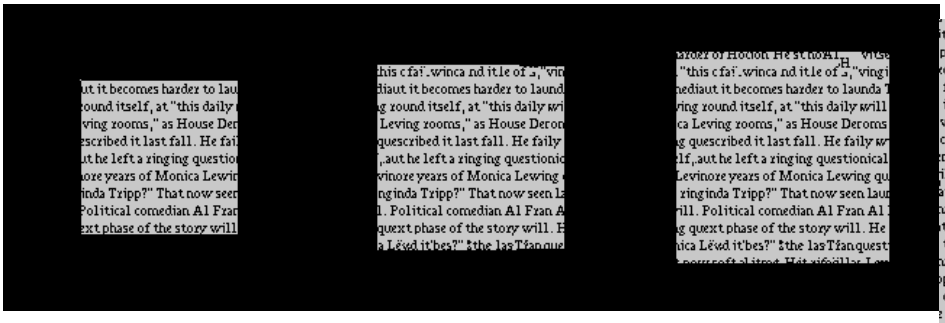

Figure 1: Texture text 0 iter

Figure 2: Texture text 5000 iter

Figure 3: Texture text 10000 iter

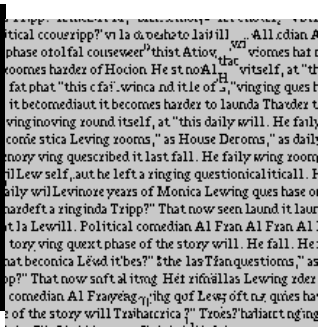

Figure 4: Texture text final

Possible improvements to my code are that it currently works only on single-channels images, as I didn't expand to 3-channel images, which would required analyses of the channels separately during the matching process.

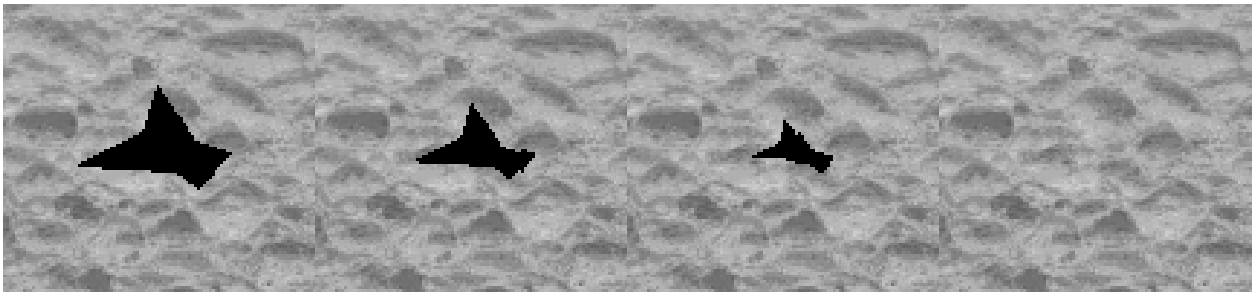

Figure 5: Texture bread 0 iter

Figure 6: Texture bread 300 iter

Figure 7: Texture bread 500 iter

Figure 8: Texture bread final

The most difficult part was finding a way to represent and retrieve the masked parts of the image, since we haven't implemented any specific binary mask before in this class. Some issues I resolved along the way were that the masked pixels were identified by having value 0, so I had to increase the brightness of pixels in the original image if they were black to avoid filling empty pixels with emptiness again; I also had to differentiate between identifying masks for hole-filling and growing images.

Texture Synthesis, CNNs The methods I implemented can be mostly found in python/ts_cnn.py
Possible improvements to my code include further debugging of matrix multiplication and correct display

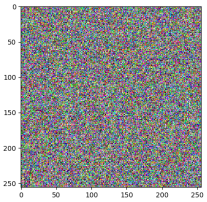

Figure 9: Initial noise image

of images (the display mode is currently incorrect).
Difficulties included the great slowness in testing, since I tested locally instead of using a remote instance, and knowing what to translate from the paper to code, and what to ignore (for implementation purposes).

Inpainting The methods I implemented can be mostly found in a10.cpp/a10.h. Problems with my code are that currently it only works on single channel images, with 1-pixel wide missing regions. This is because I ran out of time to implement an iterative method that would work for multi-pixel-wide images, so I focused on implementing something that works for masks of single pixels. To handle larger regions, the code in the inpainting method should be iterative such that the next "boundary" of missing pixels to be filled in can be found by dilating the inverse of the mask and then finding the overlap over the original mask (similar to the method in texture synthesis). After these pixels are filled, then inpainting is run from the start again for the next boundary of pixels, etc. To handle multiple channel images, the anisotrophic diffusion should predict color channel values separately.
The most difficult part of inpainting was figuring out how to correctly mask images such that the masked parts do not contribute to the structure tensor or the poisson editing. Additionally, it was difficult to understand when intermediate steps were unworking until the inpainting step, which is where the effectiveness of things like the diffusion, the structure tensor and the poisson gradient descent could be tested.

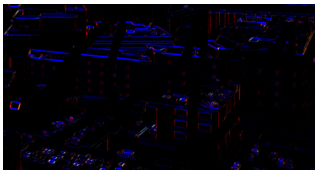

Figure 10: Structure tensor

Figure 11: Mask

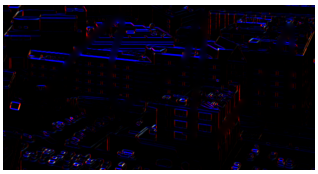

Figure 12: Poisson output

Figure 13: Inpainting output

4 Paper Links

- http://graphics.cs.cmu.edu/people/efros/research/NPS/efros-iccv99.pdf
- https://arxiv.org/abs/1505.07376
- http://graphics.cs.cmu.edu/people/efros/research/NPS/efros-iccv99.pdf