EEC 289A Homework 2

Cameron Shinn
UC Davis, Spring 2024

For this assignment we implemented a method of non-parametric texture synthesis which is able to reproduce the patterns of a source image in a new randomly generated image. Our goal was to scale this technique up on large color images with more nuanced patterns, taken from real world images. We found that this method is surprisingly effective on simple real world texture images in most cases, capturing and reproducing the patterns of simple examples. However, some cases required us to tune the parameters to effectively capture the patterns. The hyperparameters not being generalizeable makes this method tricky to make into a robust process. Our work is a reproduction of Efros and Leung's 1999 paper, "Texture Synthesis by Non-parametric Sampling".

1 Non-Parametric Texture Synthesis

Synthesizing One Pixel

The technique we used for non-parametric texture synthesis starts with the source image, I_{smp}, and takes out series of square image patches, $\omega \subset I_{smp}$, with width w for all points in I_{smp} (like convolving a window over the image). The distance between two patches can be represented as $d(\omega_1, \omega_2)$. To synthesize a single pixel, it takes the image patch $\omega(p)$ surrounding the pixel p (for now assuming all are known) and finds the closest patch from the source image patches, $\omega_{best} = \text{argmin}_{\omega} d(\omega(p), \omega) \subset I_{smp}$. Using ω_{best} as a surrogate for $\omega(p)$, it then finds all patches, ω, from I_{smp} that satisfies $d(\omega, \omega_{best}) < \epsilon$, for a configured ϵ. From those patches we randomly sample one and use its center pixel for our synthesized pixel p.

Synthesizing a Whole Image

For now we have assumed that all the pixels in the patch $\omega(p)$ are known. However, when we first synthesize the image none of the pixels have yet been determined. To solve this, we take a small patch from the image to use as our starting point. When computing our distances, we simply only compare the distances between the pixels that are known. We synthesize our image by looping over pixels in an outward spiral around the starting patch.

How We Measure Distance

For our distance measure between patches, we use the euclidean distance. When measuring distance, we don't want to evenly prioritize the similarity between all pixels in a patch. Because we are predicting the center pixel, we want to prioritize similarity of pixels closer to the center. To do this, we apply a Gaussian filter kernel to the pixel-wise distances between two patches before summing them together. We have our kernel configured to have the same width distribution no matter the filter size.

Additional Details

In addition to implementing the original paper, our focus for this project was to see how effective texture synthesis would be on the sample images provided for the assignment. Thus, we spent a lot of effort tuning the hyperparameters of the model. This includes the size of the patches extracted from the input image, the size of the starting tile in the synthesized image, the distance threshold and the Gaussian kernel standard deviation. We only used random sampling on the set of patches within the distance threshold. Given more time, we could have used a weighted random sampling that assigns higher probability to closer patches. This might have helped us overcome some of the issues. We also had to adapt our implementation to RGB images was fairly trivial. We simply add in a channel dimension as our 3rd dimension to use when computing distances. Spatial operations like the gaussian kernel are broadcast across the channel dimension.

2 Results

Our results across 12 different sample textures can be seen in Figure 1. Overall the results great and are very believable. In some of them you can see certain elements that look similar to parts of the sources image. Overall they show very similar patterns and represent their source images very well. This can primarily be due to the window size being large enough to capture most of the nuance with each of the patterns.

To find the most effective hyperparameters we tested on a small output image size. We eyeballed the Gaussian kernel distribution via visualization to find the right σ. The window size w and window distance threshold ϵ had the biggest impact on the results, so tuning those to the best output was critical. Our final set of parameters can be seen in Table 1.

3 Discussion

One thing we found was that this method does not handle perspective very well. For images taken at an angle, the patterns can appear smaller or larger based on their depth in the image. This can be seen if you look closely at the seeds texture, since the source image was taken at a slight angle. While patterns can be locally replicated pretty well, globally the perspective can look way off. We observed that images taken directly overhead of their surface tend to produce better outputs.

Another downside of this technique is that it is slow. Calculating both the distance measures from every patch from the source image and also sorting these to find the closest took lots of time. With the parameters from Table 1, it took us 1-1.5 hours to generate a single image on a V100 GPU. This is primarily due to the large window size we used. 15×15 is pretty large compared to the original paper and is possibly too generous, but in the end it *did* give us good-looking results. The decently large output image size also made collecting the final results take a while. We wouldn't be surprised at all if there are more modern techniques that can achieve better results in less compute time.

We found that images with less variation in color and brightness would sometimes fail to produce decent results. We believed this to be due to the fact that there would be many more patches within the fixed threshold ϵ in these cases. We didn't empirically validate this, but reducing ϵ helped get better images. Overall it does seem like that hyperparameter is better off being configured on a per-input basis. In general, configuring the hyperparameters is very time-consuming, so doing it on a per-input basis is not very generalizable. It would be good future work to explore modifications to this technique that is more robust to different kinds of textures.

4 Conclusion

Overall we thoroughly enjoyed this project. The actual technique is relatively simple compared to modern deep learning techniques, but implementing it from scratch was not easy and made for a fun challenge. It felt very empowering not needing to use any existing code and implementing their technique purely based on their description in the paper. Seeing the results was really interesting and felt like magic. Playing around with the hyperparameters and seeing the resulting changes in the output was very intriguing. Our source code can be found at `https://github.com/cameronshinn/tex-synth`.

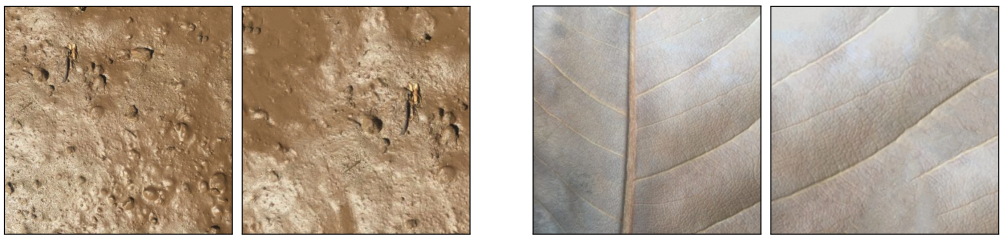
5 Contributions

Since we were a one person group, everything was done by Cameron Shinn for this assignment.

Hyperparameter	Value(s)
Synthesized image size	400×400
Window size, w	15×15
Starting patch size	7×7
Sampled window distance threshold, ϵ	0.001
Gaussian kernel standard deviation, σ	0.3
Nearby window sampling	uniformly random

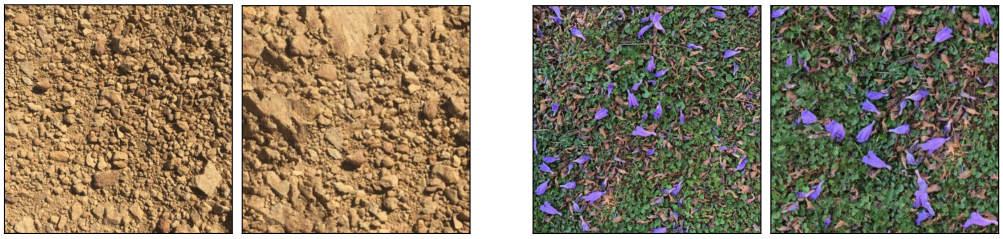
Table 1: The hyperparameters we found to be effective at synthesizing new textures from the ones given as a part of the assignment.

(a) Wood texture **(b)** Synthesized image **(c)** Snow texture **(d)** Synthesized image

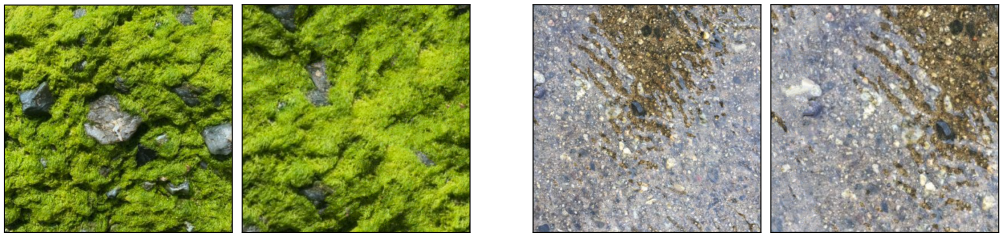


(e) Mud texture **(f)** Synthesized image **(g)** Leaf texture **(h)** Synthesized image

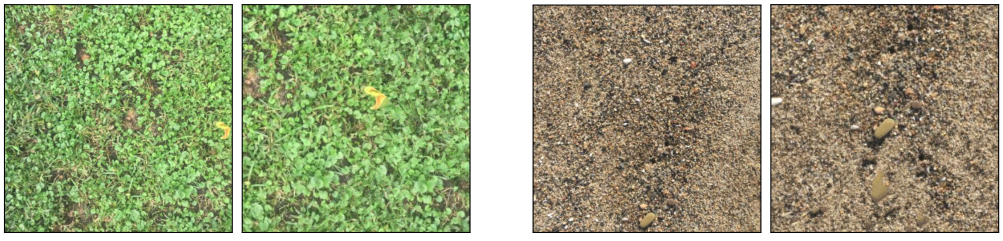
(i) Flower texture **(j)** Synthesized image **(k)** Seed texture **(l)** Synthesized image



(m) Rocky texture **(n)** Synthesized image **(o)** Grass texture **(p)** Synthesized image



(q) Moss texture **(r)** Synthesized image **(s)** Water texture **(t)** Synthesized image



(u) Clover texture **(v)** Synthesized image **(w)** Dirt texture **(x)** Synthesized image

Figure 1: Our final results for our image synthesis implementation. Note that the synthesized images are zoomed in because they are slightly smaller than the source images.