IAT 455 Final Project: Texture Synthesis
Team 6 members:
Jianghui (David) Dai
Teng Pin (Eric) Pan

- **Overview**:
The goal of this project is to create an application (Figure 1.) that allows the users to select images from a predefined set of texture images, and produces three synthesized images through three different patch-based methods. The three methods build on top of one another and produce better-synthesized images one after another. The three methods in order of built-ups are Random placement of blocks, Neighboring blocks constrained by overlap, and Minimum error boundary cut.

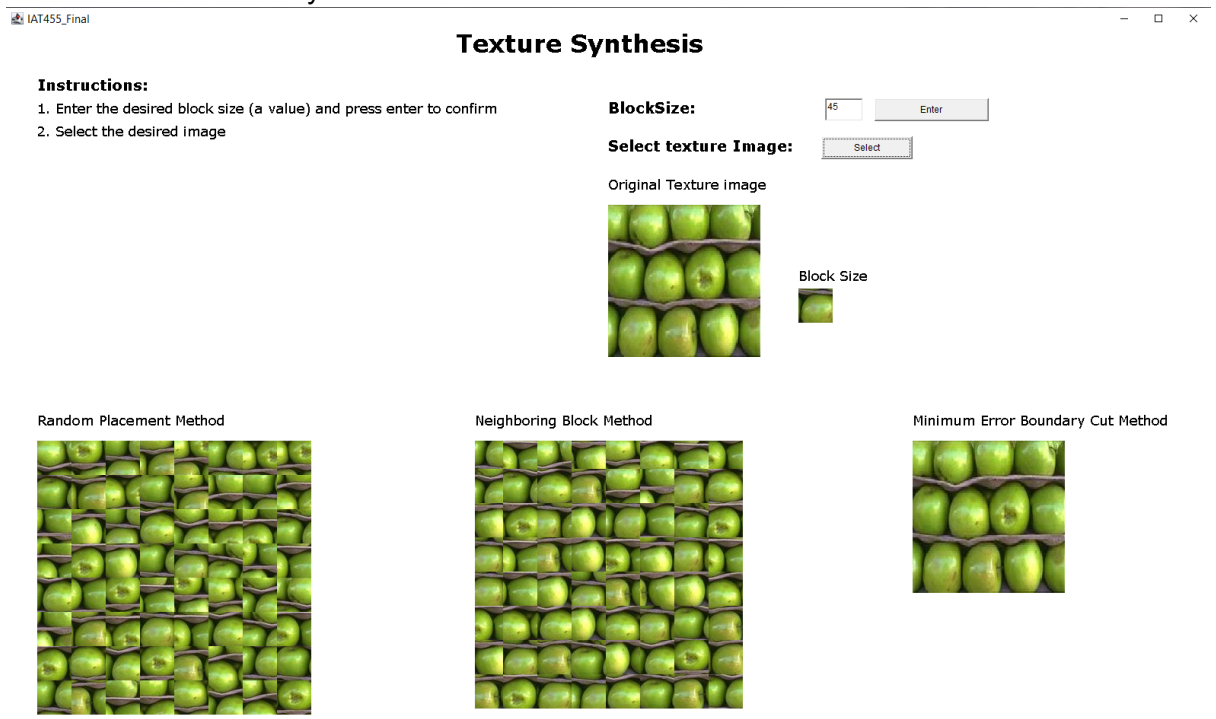

Figure 1. Screenshot of the texture synthesis application.

- **Research**:
There were few issues that were encountered during the research and code implementation of the selected topic.

1. **Issue:** Inside Efros and Freeman's paper, it briefly explains patch-based texture synthesis approach to generate the final synthesized images. It achieves the results by comparing the selected parts (patches) of the original image directly with one another. In the neighboring Block method, the patches of image are used if they are similar enough to one another. However, this method leaves visible edges (Figure 2) in between patches (blocks) which is fixed with the implementation of the third method (Minimum Error Boundary Cut).
2. **Solution:** Since direct simple patch to patch comparison cannot be done in Java, we first generate an initial patch by randomly selecting a point in the original images and create the rest of the patch by mapping the neighboring pixel until the specified block size is generated. Then we compare all the pixels in the boundary (area of blocks overlapping) of the generated patch to the corresponding pixels in the boundary of another patch and compare their similarities. If the result shows a difference within the tolerance value, then the compared patch is placed beside the previous patch.

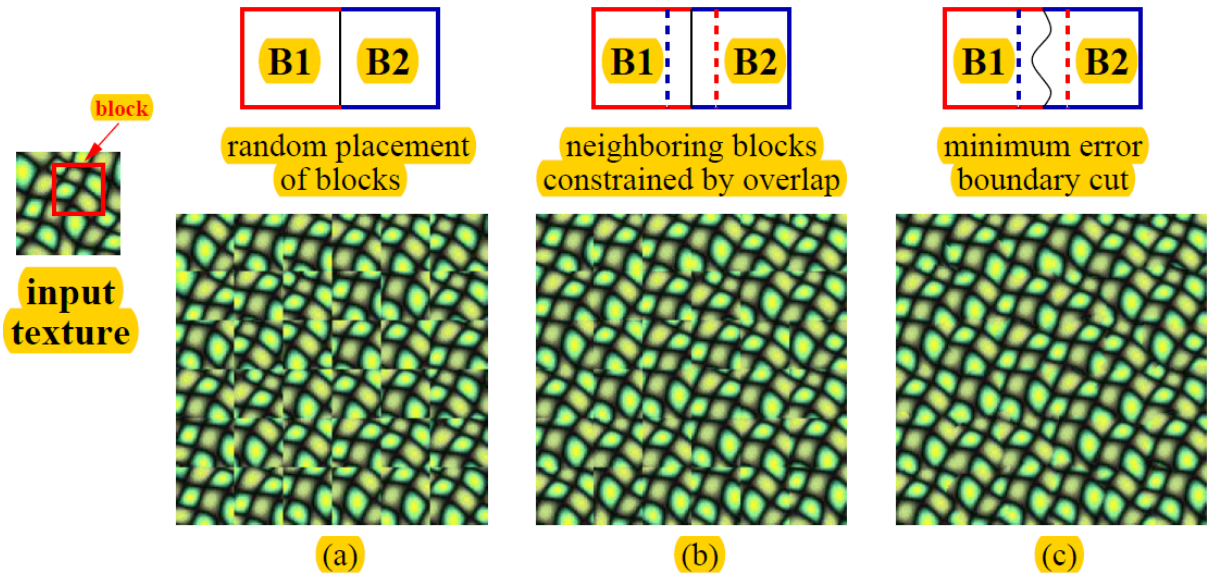

Figure 2. Screenshot of Efros and Freeman's patch-based texture synthesis approach.

3. **Issue:** In the neighboring blocks method, we originally precut the texture image into patches, and then loop through these precut patches to find the best fitting one that matches with our target texture patch. This approach leads to a high chance of using the same precut patch over and over again causing the final result with unnatural repetitions.
   Solution: Instead of pre-cutting the original texture image into patches, we randomly select points from the original image and then use them to generate patches and compare until the least different patch is found.

- **Work description:**

Week 10

Both of us started to research the topic and tried to implement the first random placement method. At the end of the week, David successfully implemented the random placement method, and he also made a basic AWT UI to show the result.

Week 11

At this stage, Eric rewrote the previous interface and used the Swing library to build a basic user interface allowing the user to select input images from a given set. At the end week, we started to implement the neighboring block method and tried to understand what the algorithm is about.

Week 12

In the final week, we continued to improve the neighboring block method together. At the end of the week, David successfully produced a better neighboring block method and got some desired results. Then, we modified the neighboring block method together and changed how a random block is produced from the source image to remove some of the relatedness of the synthesis images. In the end, Eric tried to write the minimum error boundary cut method but failed to understand how to map the minimum error from each overlap row back to the overlap area for each pixel.

- **Final result**:
We completed the random placement method and the neighboring block method for the topic of image synthesis,but we failed to implement the minimum error boundary cut method.

The random placement method:
We achieved this method by writing a random block helper method first. The random block method produces a random block based on the source image, and the user defines its size before calling the function. It iterates through the synthesis image's width and height to place each random block to the output image.

The neighboring block placement method:

- We achieved this method by writing three helper methods: the findMinimumRightBlock method, findMinimumBottomBlock method, and findBlockMethod. Then, in the neighboring block placement method, we used the random block method to generate the first top left block for the output image.

- For blocks in the first column of the output image, we use the findMinmumBottomBlock method to find a random block from the source image with the most similar color to the top block and place it under the top block.

- For blocks in the first row of the output image, we used the findMinmumRightBlock method to find a random block from the source image with the most similar color to the left block and place it besides the left block.

- For other columns and rows, we used the findBlock method to find a random block from the source image, which has the most similar color to the top block and the most similar color to the left block and place it besides the left block (under the top block).

The minimum error boundary cut method:
This was the most difficult method to implement and is the only one we failed to implement. What sets this method apart from the previous methods is that it calculates the minimal cost path through the error surface (overlap boundaries of patches) to address the issue of unnatural (not matching) edges between each patch. Figure 3 showcases our understanding of the approach. The blue and red squares each represent pixels from the corresponding neighboring patch boundaries, and through the equation

$$E_{i,j} = e_{i,j} + min(E_{i-1,j-1}, E_{i-1,j}, E_{i-1,j+1})$$

we are then able to compute the minimum error E for all paths. Error E is then used to map out the boundaries of each patch such that the "seam" (illustrated by purple squares with black dots in Figure 3) of the patches are as much as possible similar to both patches. However, the problem comes when trying to implement this equation into the code as we couldn't map the corresponding minimum error pixels back to their corresponding patch boundary. This causes the chain effect in which we could not generate the minimal cost path to blend the edges of the patch together to create a more natural texture synthesis.

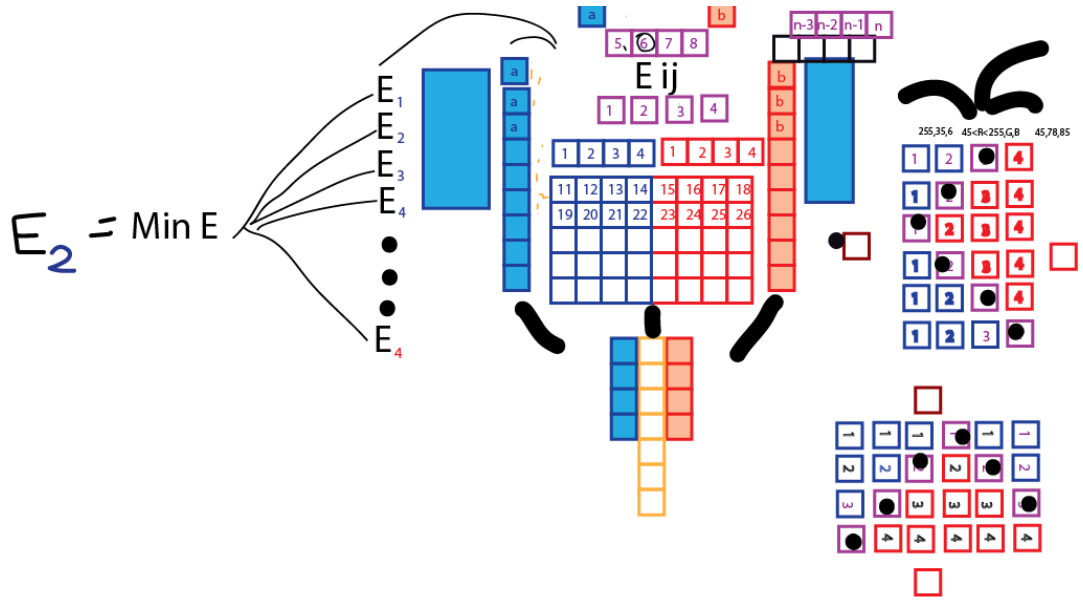

Figure 3. Screenshot of trying to understand how the minimum cost path through the error surface is computed.

- **Distributions of tasks:**

David:

I successfully implemented the randomBlockPlacement method and the neighboringBlockPlacement with Eric's help, and I also made some adjustments to Eric's interface.

Eric:

I primary worked on implementing the user interfaces, aid David in improving the second method (the neighboring block placement), and attempted to implement the third method (Minimum Error Boundary Cut).

- **Conclusion:**

David:

I learned the basic idea of patch-based image synthesis and implemented some synthesis methods into actual codes. I think the experience of following an academic paper and recreating the algorithm from the paper is worth having if I want to do the research studies in the future.

Eric:

I learned different approaches trying to perfect the results of texture synthesis through patch-based methods. For instance, this includes how the three methods presented in Efros and Freeman's paper are a built-up / iteration of the previous methods to address the problems of synthesizing texture images (i.e. reducing blockiness of the boundary between synthesized images in a patch-based approach).

- **Reference:**

Alexei A. Efros and William T. Freeman. 2001. Image quilting for texture synthesis and transfer. In Proceedings of the 28th annual conference on Computer graphics and interactive techniques. Association for Computing Machinery, New York, NY, USA, 341–346. DOI:https://doi-org.proxy.lib.sfu.ca/10.1145/383259.383296

Choudhary, D. (2020, June 21). Texture synthesis : Generating arbitrarily large textures from image patches. Retrieved March 20, 2021, from https://devashi-choudhary.medium.com/texture-synthesis-generating-arbitrarily-large-textures-from-image-patches-32dd49e2d637