Image Quilting

Introduction

Texture synthesis is the process of algorithmically constructing a large digital image from a small digital sample image by taking advantage of its structural content.

Goal of Texture Synthesis :

Texture synthesis algorithms are intended to create an output image that meets the following requirements:

1. The output should have the size given by the user.
2. The output should be as similar as possible to the sample.
3. The output should not have visible artifacts such as seams, blocks and misfitting edges.
4. The output should not repeat, i. e. the same structures in the output image should not appear multiple places.

Multiple texture synthesis approaches have been researched and developed.
The approaches we are using in the project for texture synthesis are :-

- ***Pixel-based texture synthesis(1999 Efros and Leung)***
- ***Patch-based texture synthesis(2001 Efros and Freeman)***

Efros & Leung 1999[1]

Approach

- Our algorithm “grows” texture, pixel by pixel, outward from an initial seed. We chose a single pixel p as our unit of synthesis so that our model could capture as much high frequency information as possible. All previously synthesized pixels in a square window around p are used as the context.
- The neighborhood of a pixel is modelled as a square window around that pixel. The size of the window is a free parameter that specifies how stochastic the user believes this texture to be. More specifically, if the texture is presumed to be mainly regular at high spatial frequencies and mainly stochastic at low spatial frequencies, the size of the window should be on the scale of the biggest regular feature.

Synthesizing texture

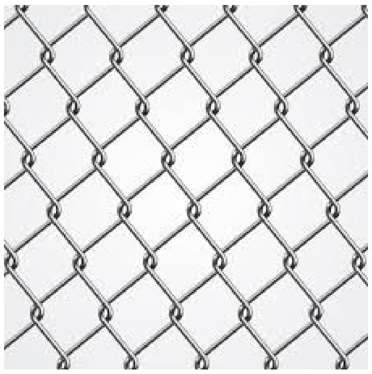
- In this work we model texture as a Markov Random Field (MRF). That is, we assume that the probability distribution of brightness values for a pixel given the brightness values of its spatial neighborhood is independent of the rest of the image.
- Let Ismp be the available texture sample image and Ireal be the real infinite texture. Let p be a pixel and W(p) which is subset of I be a square patch of width w centred at p.
- We need to find the subwindow of Ismp which best fits the current neighbouring window. Here, we can define a parameter d for comparing the 2 windows and our target is to minimise d so as to find the best fit and then

the centre pixel is added to the output image, while we mark the current pixel in the filled map for further considerations.

- The final texture is grown from a 3 by 3 seed from the sample image. Many pixels in the W(p) might be unknown, so the synthesis algorithm is modified to handle unknown pixels.
- This is done by only matching the matching on the known values and then normalizing the error by the number of known pixels while computing the conditional pdf for p.

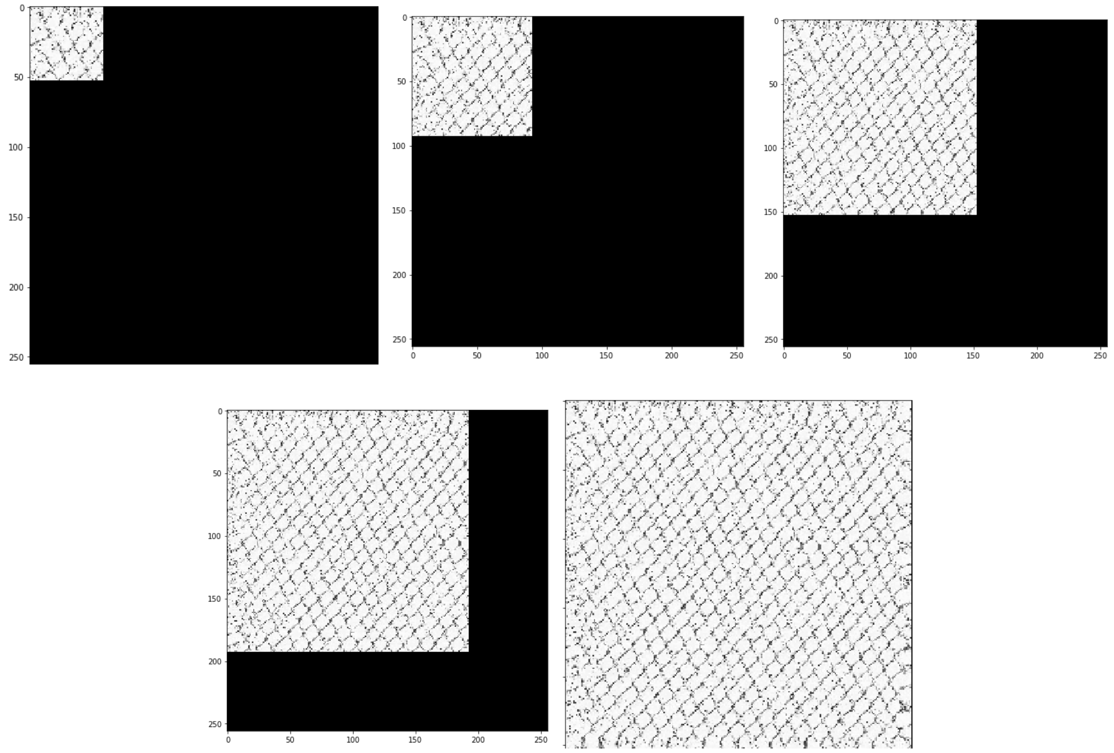
Results

- Our algorithm produces good results for a wide range of textures. The only parameter set by the user is the width of the context window. This parameter appears to intuitively correspond to the human perception of randomness for most textures.

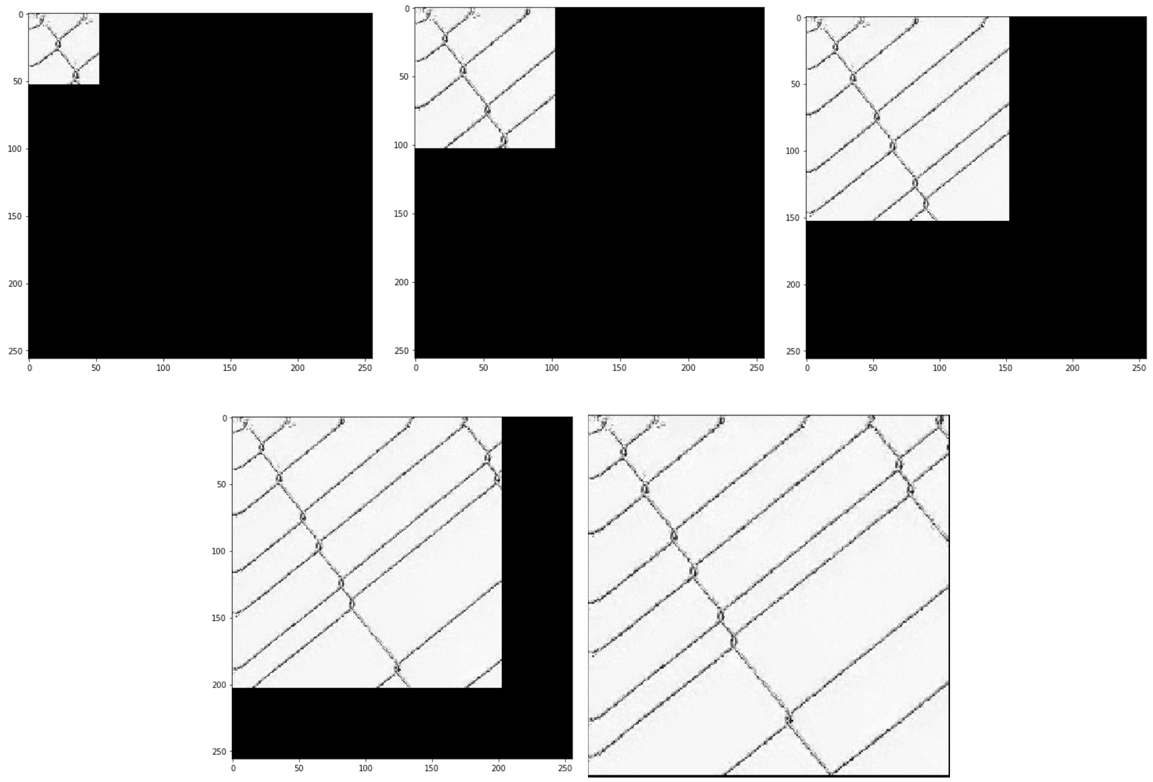


Original Image

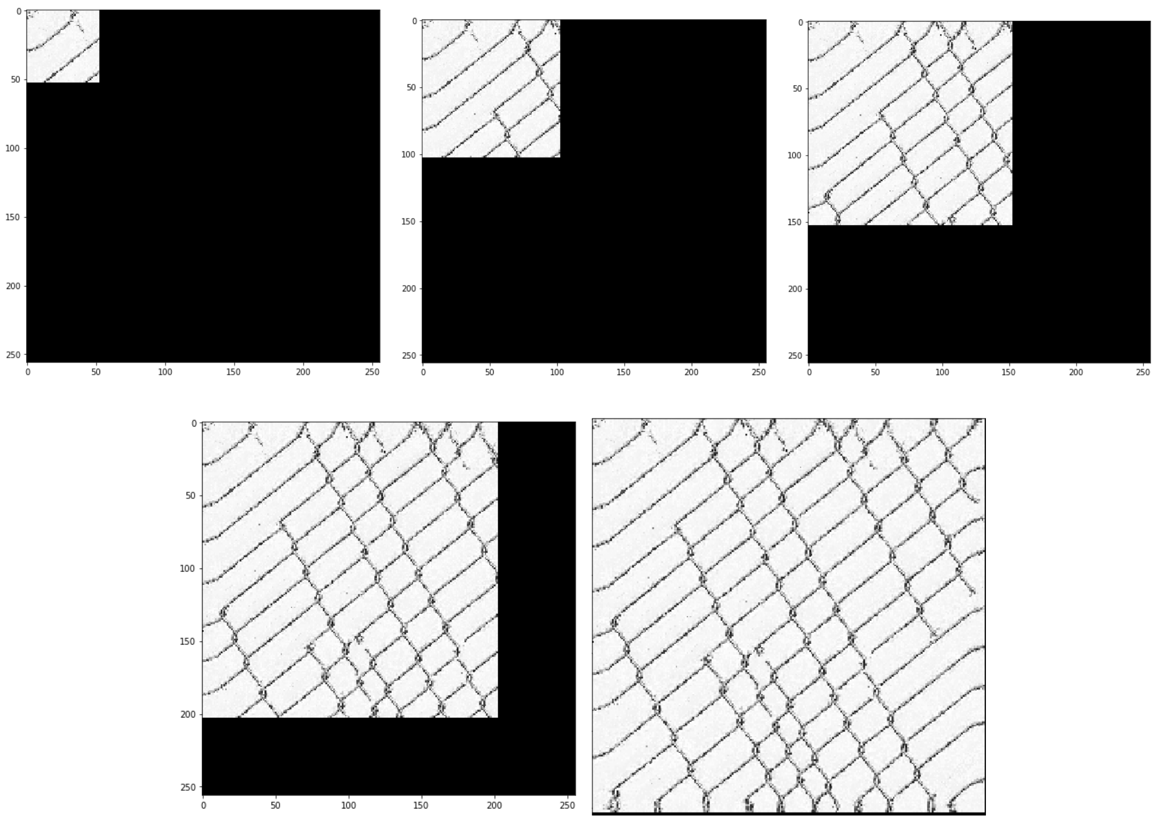
Window_len = **21**



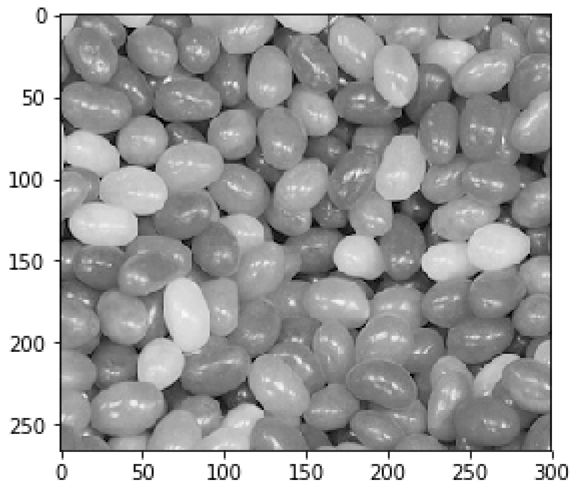
Window_len = **16**

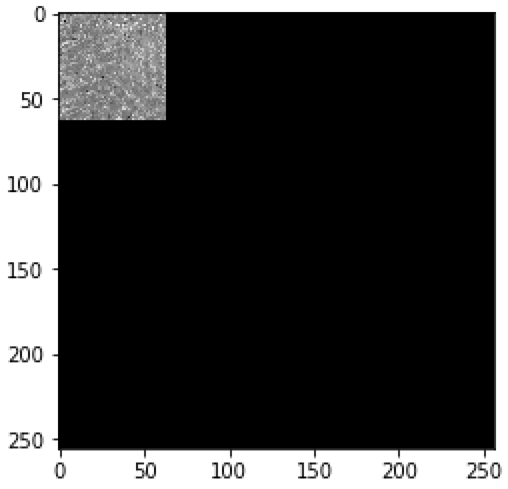


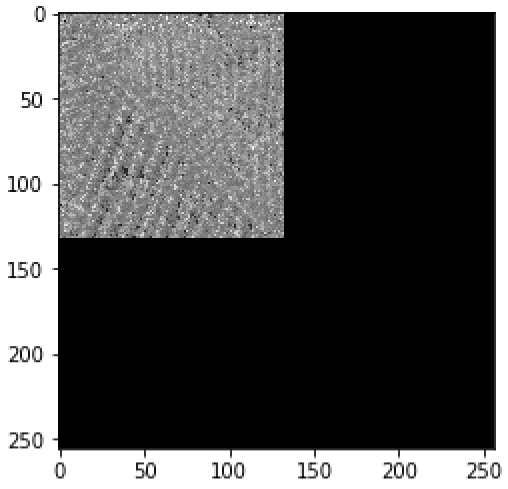
Window_len = 23

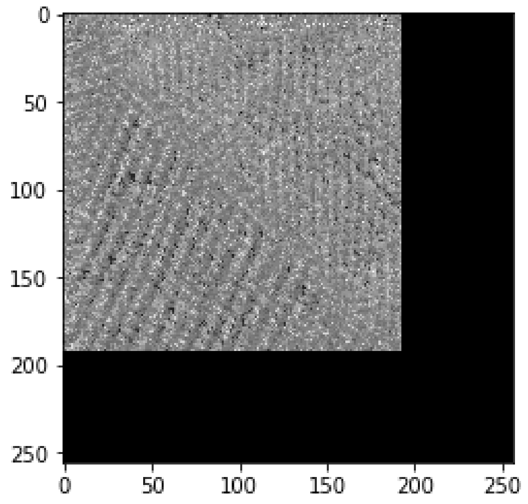


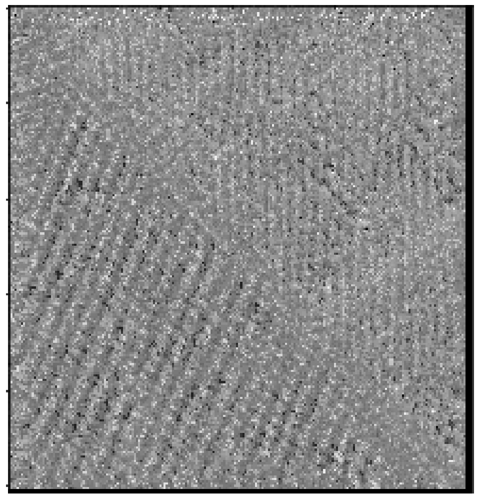
<u>Original Image</u>



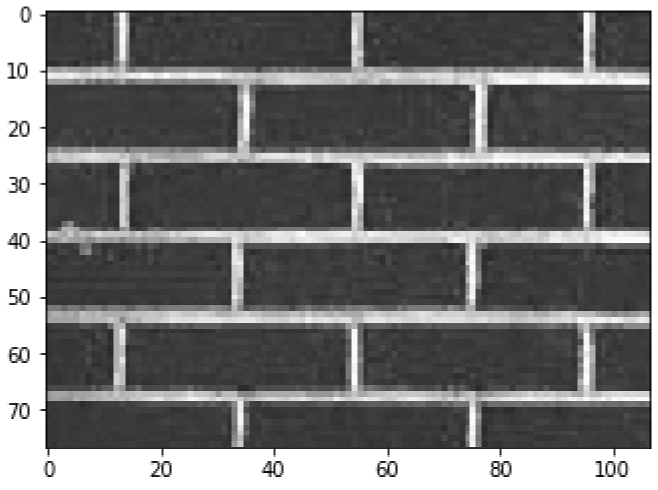




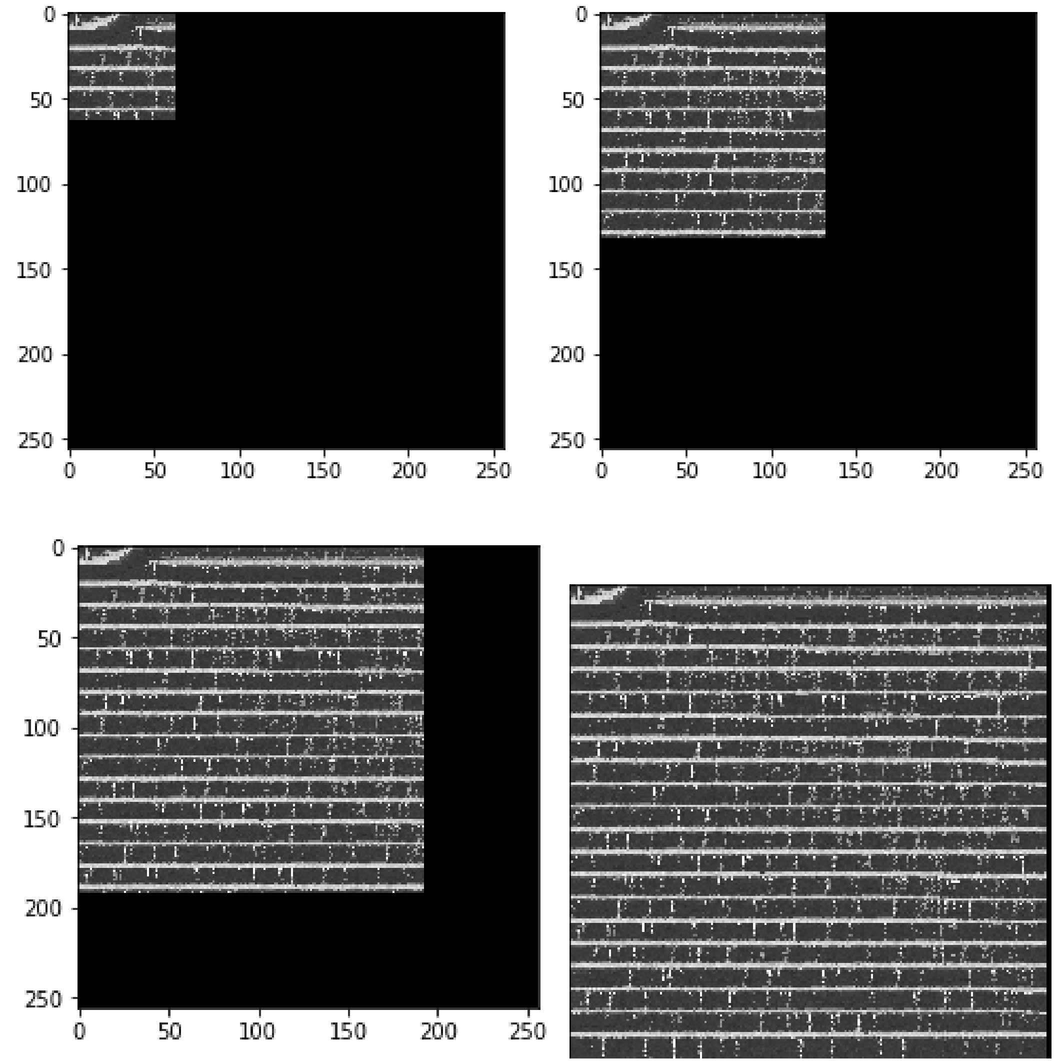




<u>Original Image</u>

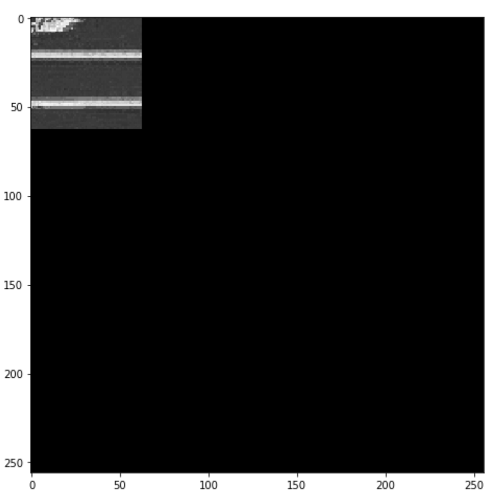


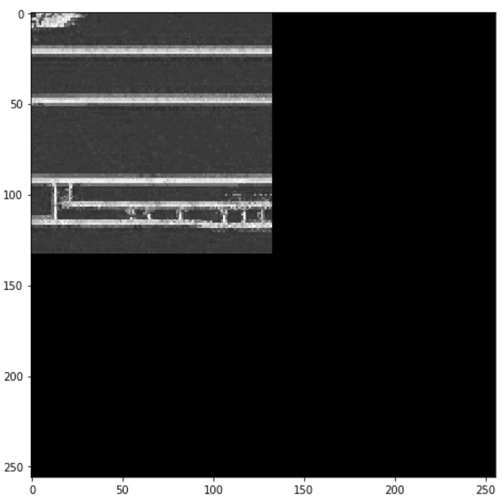
Window_len = **21**

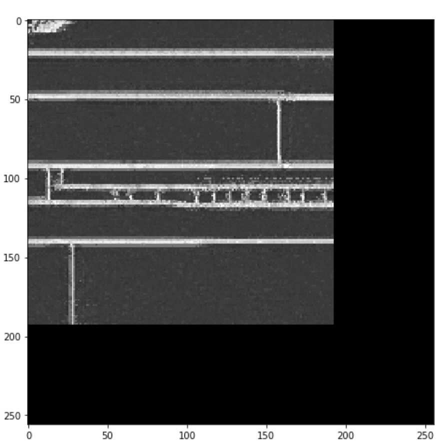


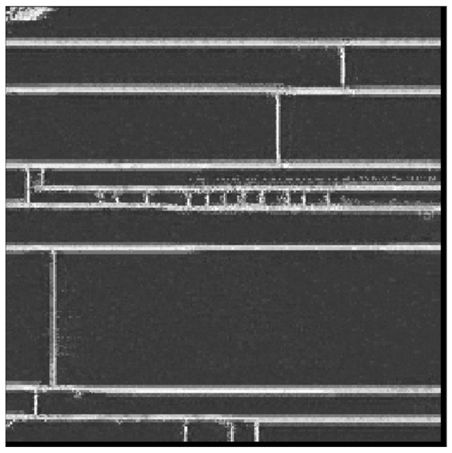
Window_len = **15**

For *Window len* = **15**, we see that the bricks don't fit that well, after tuning the parameters we found out that *window len* = **21** was giving the better results.









Limitations

- One problem of our algorithm is its tendency for some textures to occasionally “slip” into a wrong part of the search space and start growing garbage or get locked onto one place in the sample image and produce verbatim copies of the original. These problems occur when the texture sample contains too many different types of texel

Image Quilting' 2001[2]

Idea

- For most complex texturesery few pixels actually have a choice of values that can be assigned to them. That is, during the synthesis process most pixels have their values totally determined by what has been synthesized so far.
- Taking this into consideration, we define the unit of synthesis in this approach to be a block Bi .

Approach

- The approach for calculating the next best match is similar to the previous approach with the unit of synthesis, a block, rather than a pixel.
- This makes the process much faster than the previous approach.
- Now, we introduce some overlap in the placement of blocks onto the new image. Now, instead of picking a random block, we will search SB for such a block that by some measure agrees with its neighbors along the region of overlap.
- Finally, we will let the blocks have ragged edges which will allow them to better approximate the features in the texture. Now, before placing a chosen block into the texture we will look at the error in the overlap region between it and the other blocks. We find a minimum cost path through that error surface and declare that to be the boundary of the new block.

Minimum Error Boundary Cut:

The minimal cost path through the error surface is computed in the following manner.

- If B1 and B2 are two blocks that overlap along their vertical edge with the regions of overlap Bov1 and Bov2 , respectively, then the error surface is defined as

  $$e = (B_{ov1} - B_{ov2})^2$$

  To find the minimal vertical cut through this surface we traverse e (i = 2..N) and compute the cumulative minimum error E for all paths.

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

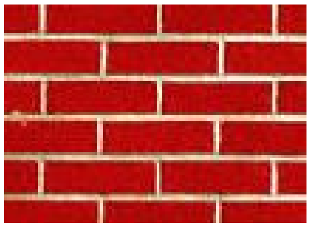
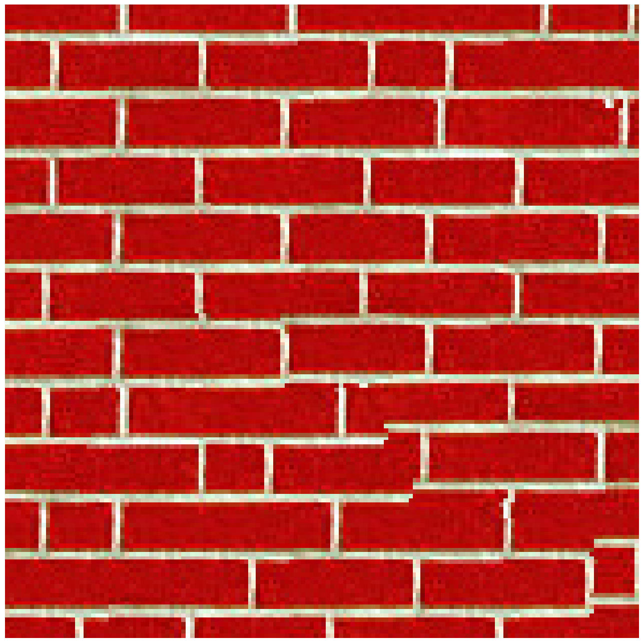
$$\mathrm{MinPaths}_{i,j} = \operatorname{argmin}(E_{i-1,j-1}, E_{i-1,j}, E_{i-1,j+1})$$

The Image Quilting Algorithm:

- Go through the image to be synthesized in raster scan order in steps of one block.
- For every location, search the input texture for a set of blocks that satisfy the overlap constraints within some error tolerance. Randomly pick one such block.
- Compute the error surface between the newly chosen block and the old blocks at the overlap region. Find the minimum cost path along this surface and make that the boundary of the new block. Paste the block onto the texture. Repeat.

Result

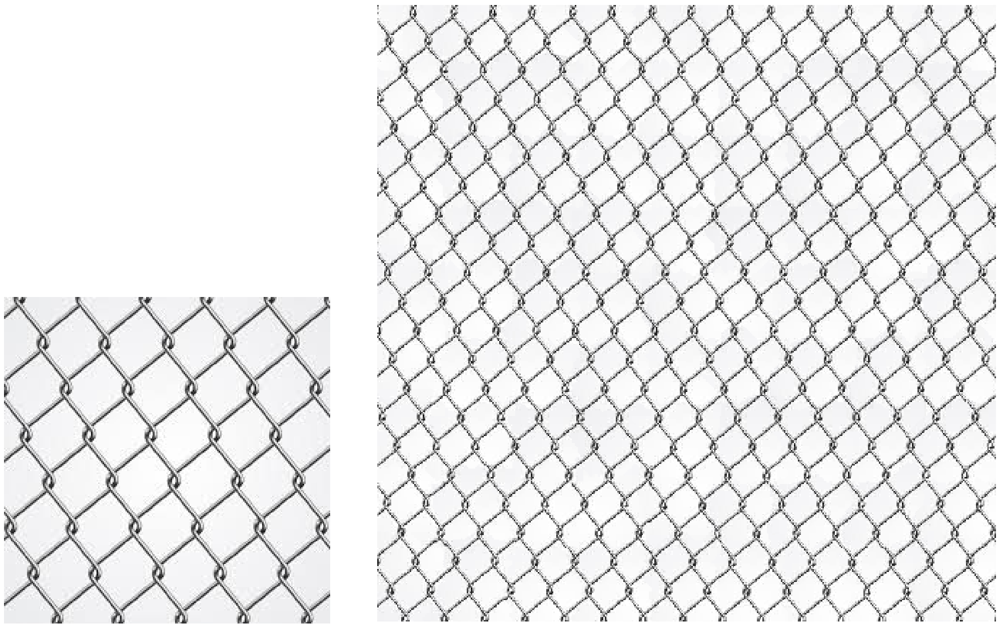
- The results of the synthesis process for a wide range of input textures are shown in the jupyter notebook. Algorithm works quite well for most of the input images and is found to be effective for semi structured images. The algorithm is runs really fast providing decent outputs.
- This work demonstrates the enormous importance of local structure for successfully modelling the visual phenomenon

.

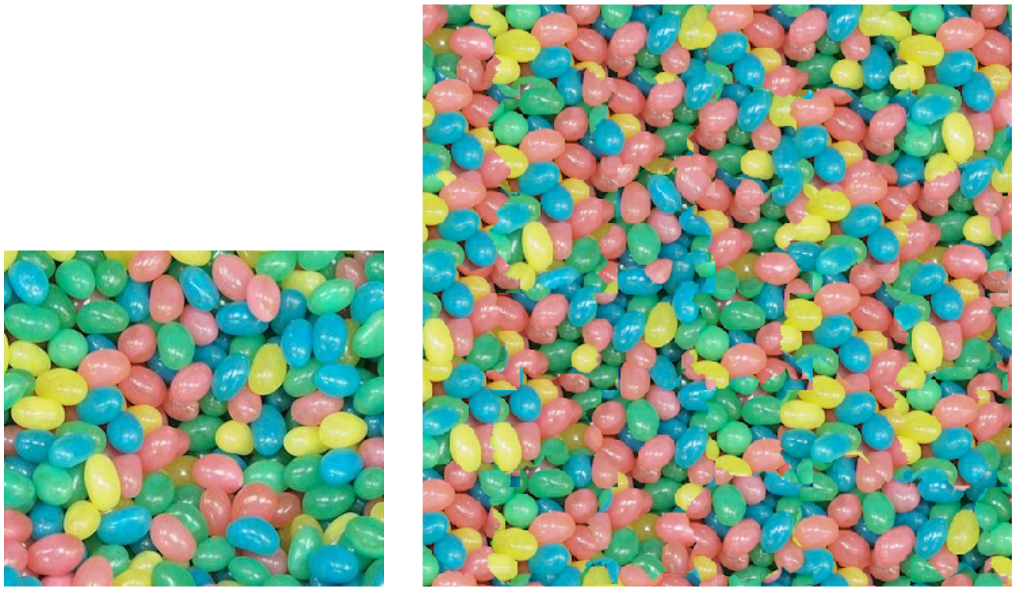
Factor = 2

Error Threshold = 0.05



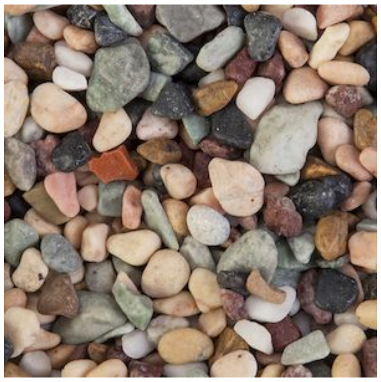
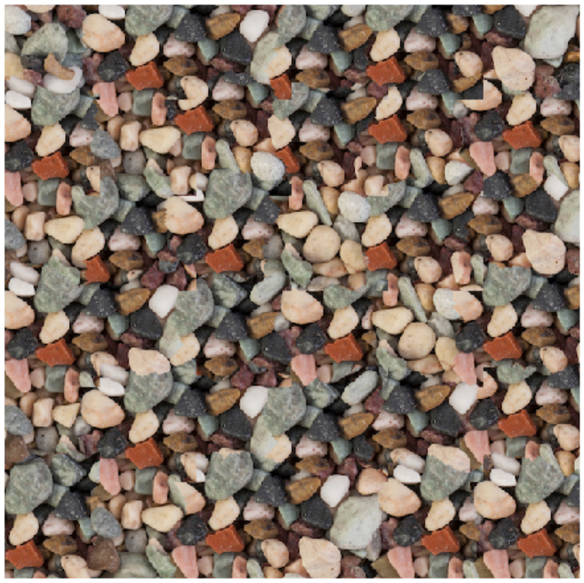
Factor = 3

Error Threshold = 0.15



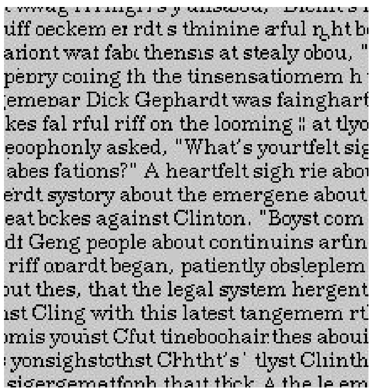
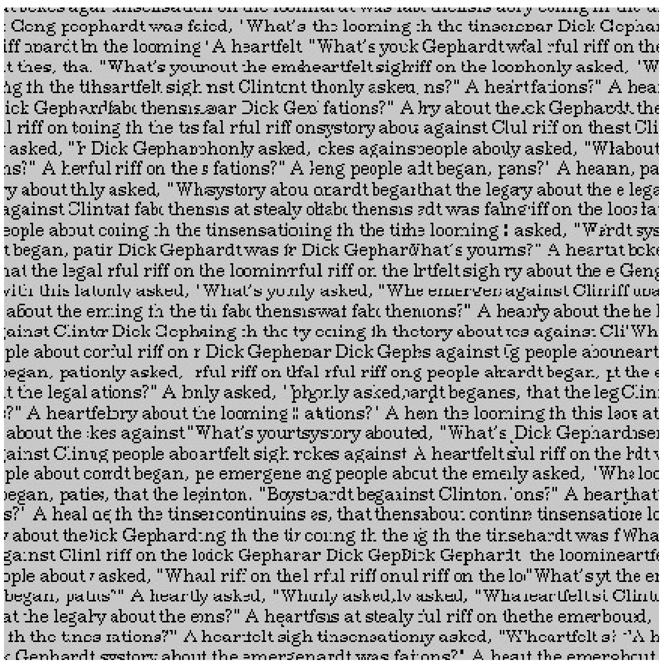
Factor = 2

Error Threshold = 0.15

Factor = 3

Error Threshold = 0.10

Factor = 2

Error Threshold = 0.05

Current Progress

1. Implemented and observed results from research paper Texture Synthesis by Non-parametric Sampling.
2. Implemented the Image Quilting part from research paper Image Quilting for Texture Synthesis and Transfer

Milestones Left

- Improving the performance of implemented algorithms.
- Extensive testing and observation of results of the texture synthesis algorithm implemented and presenting results for the same.
- Working towards Texture Transfer from research paper Image Quilting for Texture Synthesis and Transfer
- **BUFFER PERIOD**

Texture Transfer

Basic Idea : Take the texture from one object and "paint" it onto another object
This requires separating texture and shape.

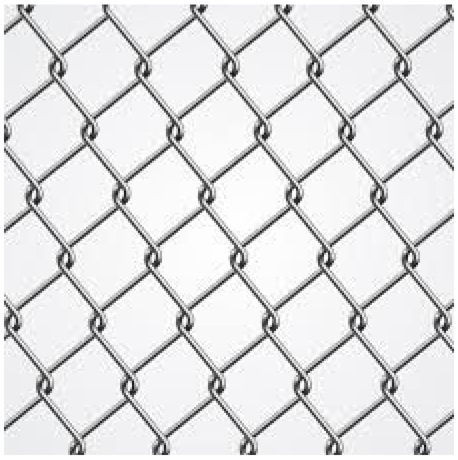
Approach

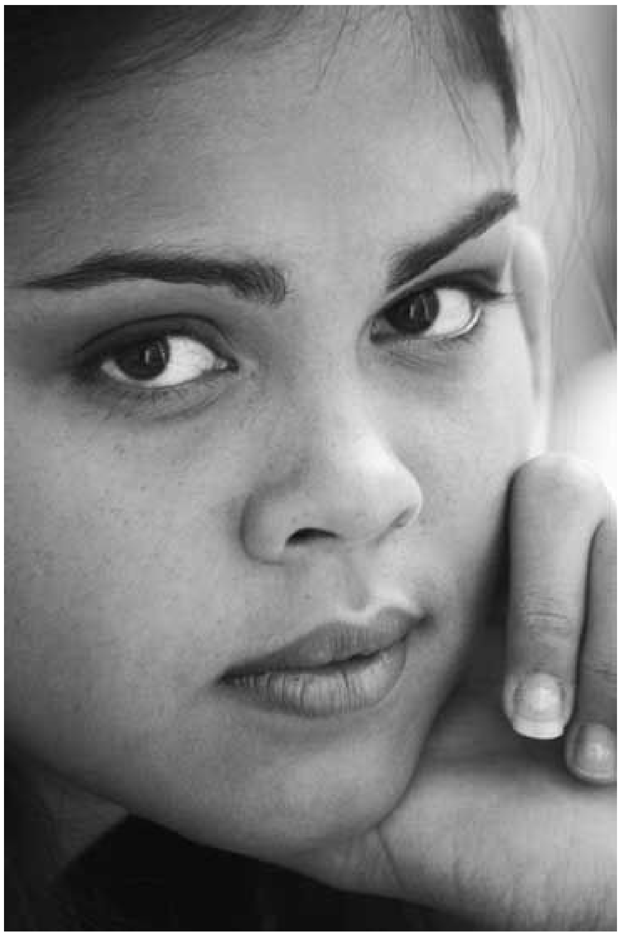
- We have implemented the texture transfer, using the image quilting algorithm described above.
- The synthesis algorithm has been augmented by adding the requirement that each patch must satisfy a desired correspondence map as well as satisfy the texture synthesis requirements.
- Image being synthesized must respect two independent constraints-
    - The output are legitimate, synthesized examples of the source texture.
    - The correspondence image mapping must be respected for proper texture transfer observations.
- The handling of the 2 terms can be done using a weighted sum of the 2 desired conditions. We consider α times the block overlap matching error and 1 - α times the squared error between the correspondance map pixels within the source texture block and those at the current target image position. The parameter α determines the tradeoff between the texture synthesis and fidelity to target image.

Challenges

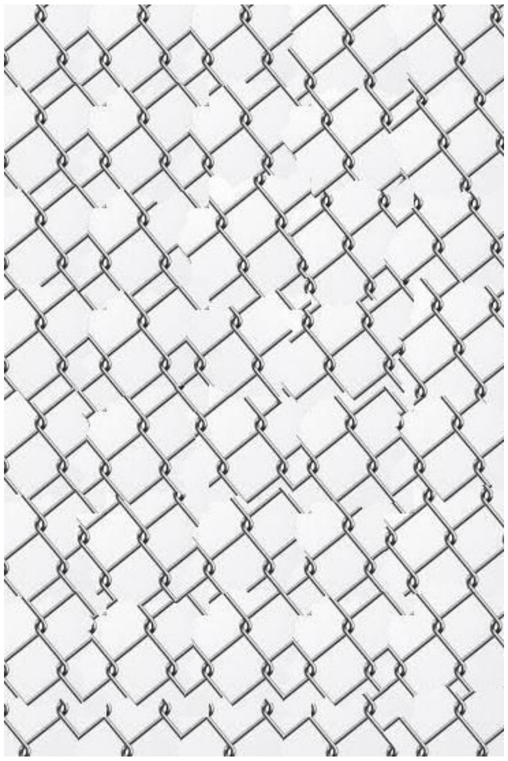
- Because of added constraint, generally one synthesis pass through the image is not enough to produce a visually pleasing result.
- We need to minimise a weighted error giving appropriate weights to the 2 constraints. How the value of α is decided can alter the results to some extent.

Input Images

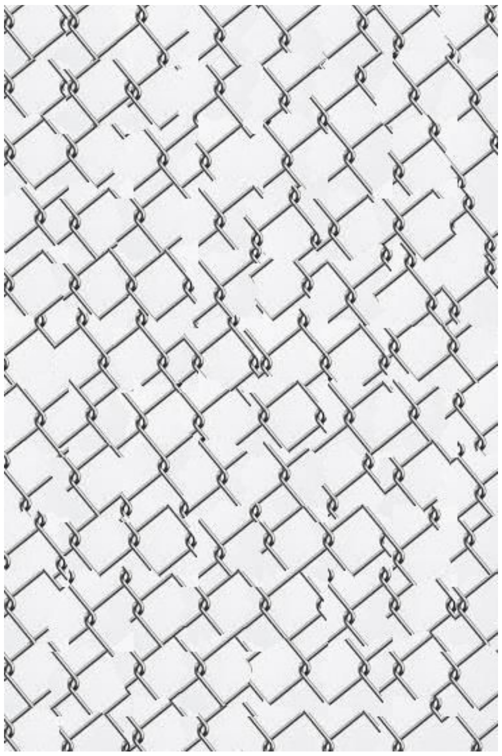




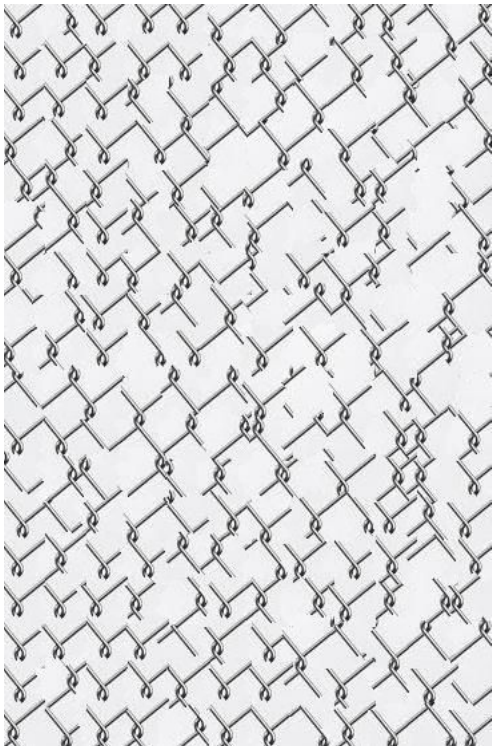
Iteration 1



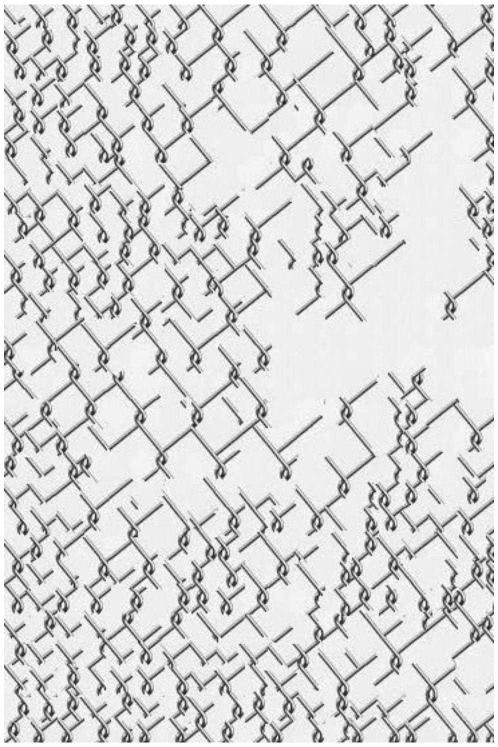
Iteration 2



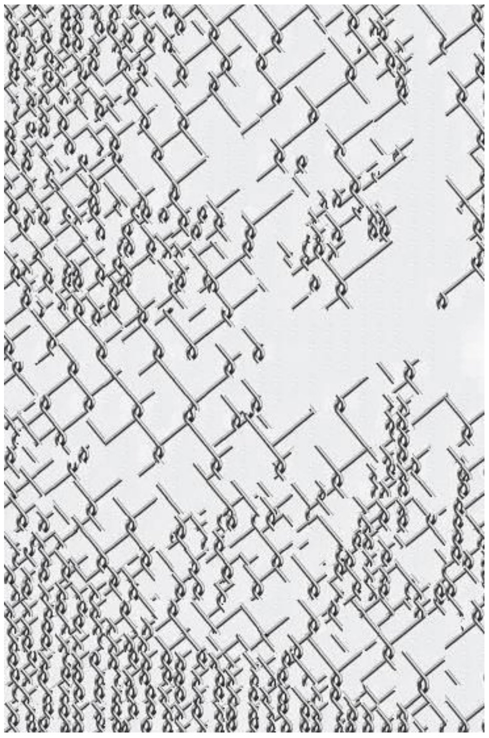
Iteration 3



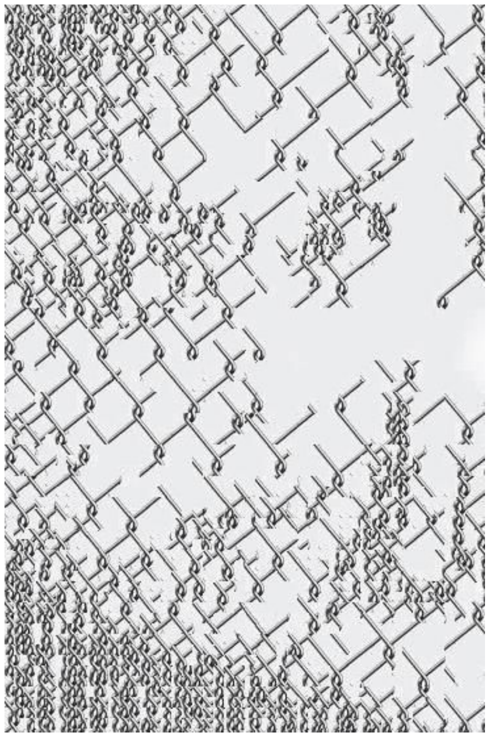
Iteration 4



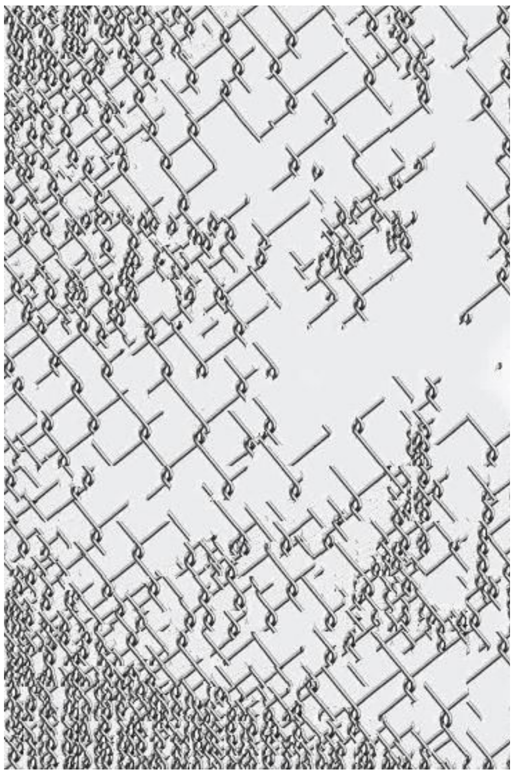
Iteration 5



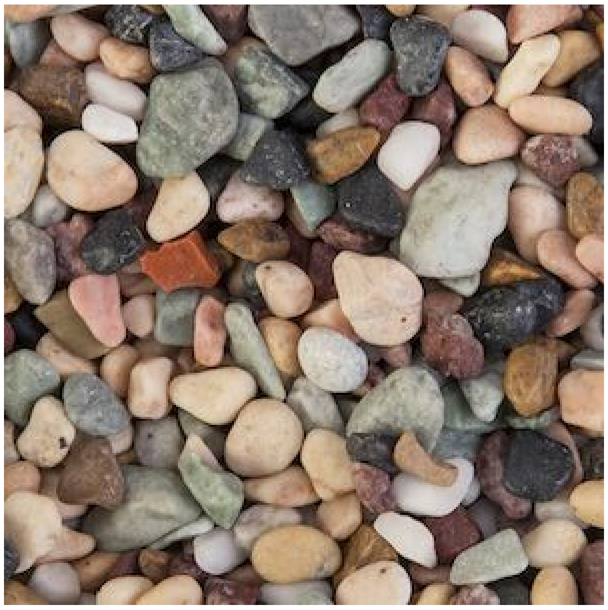
Iteration 6

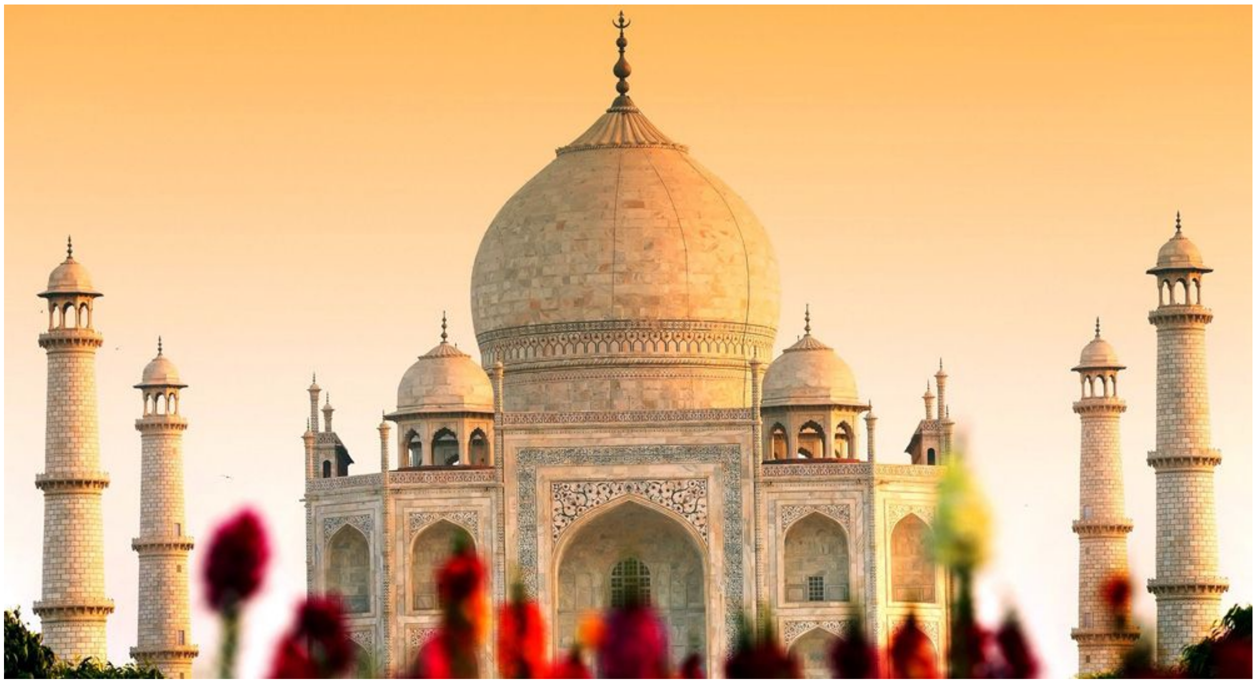


Final Output

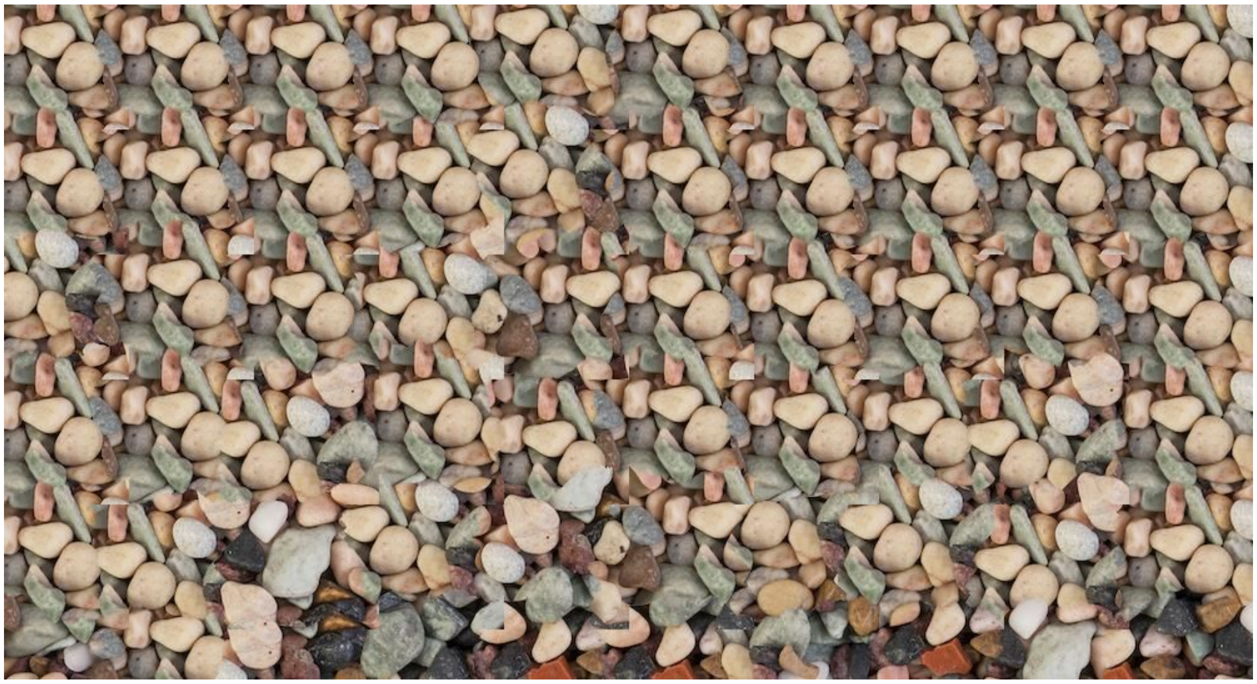


Input Images

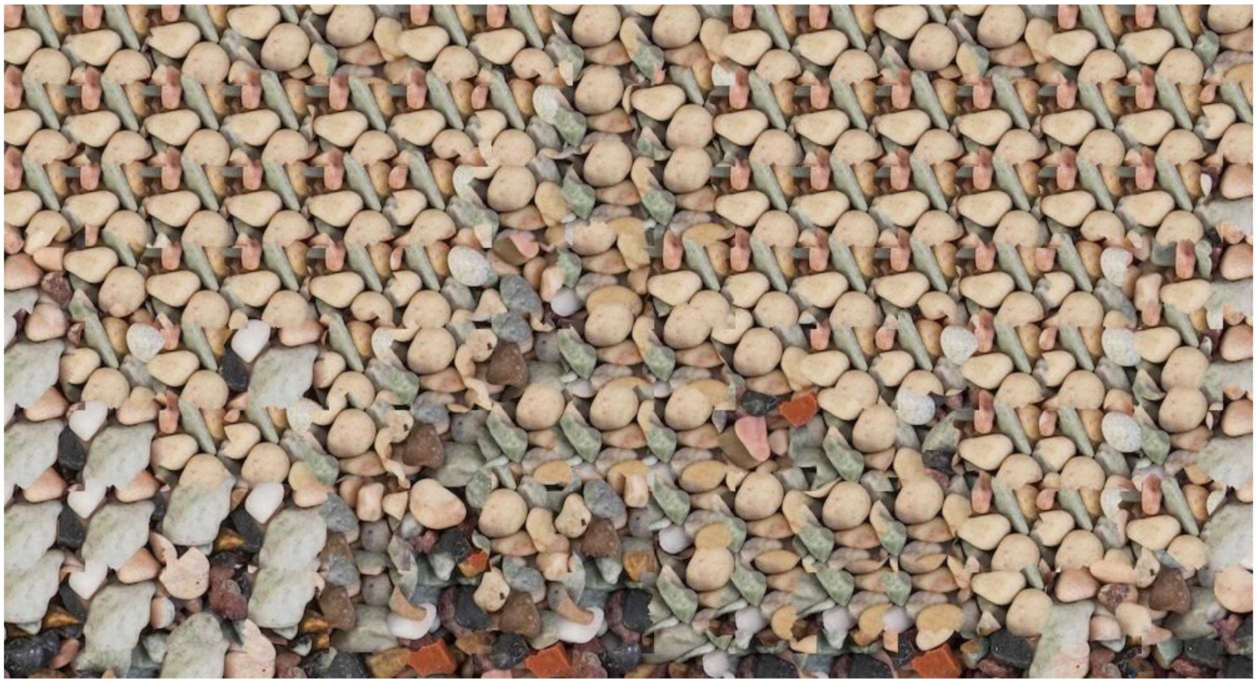




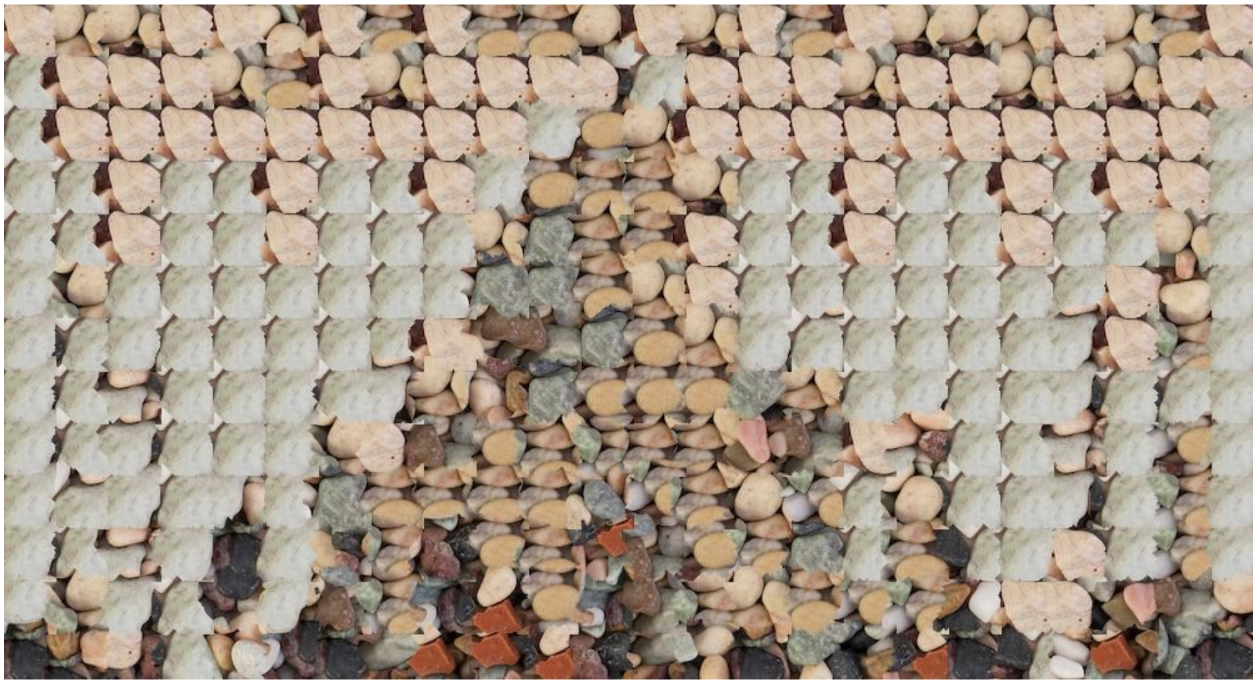
Iteration 1



Iteration 2

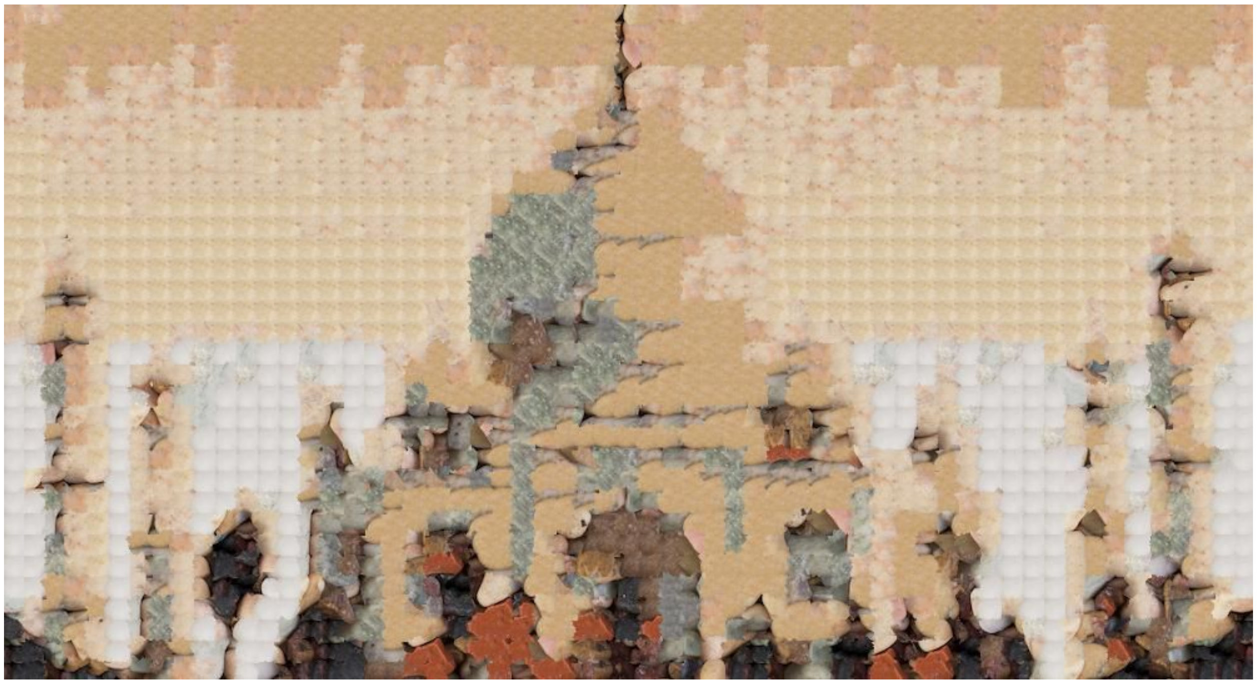


Iteration 3

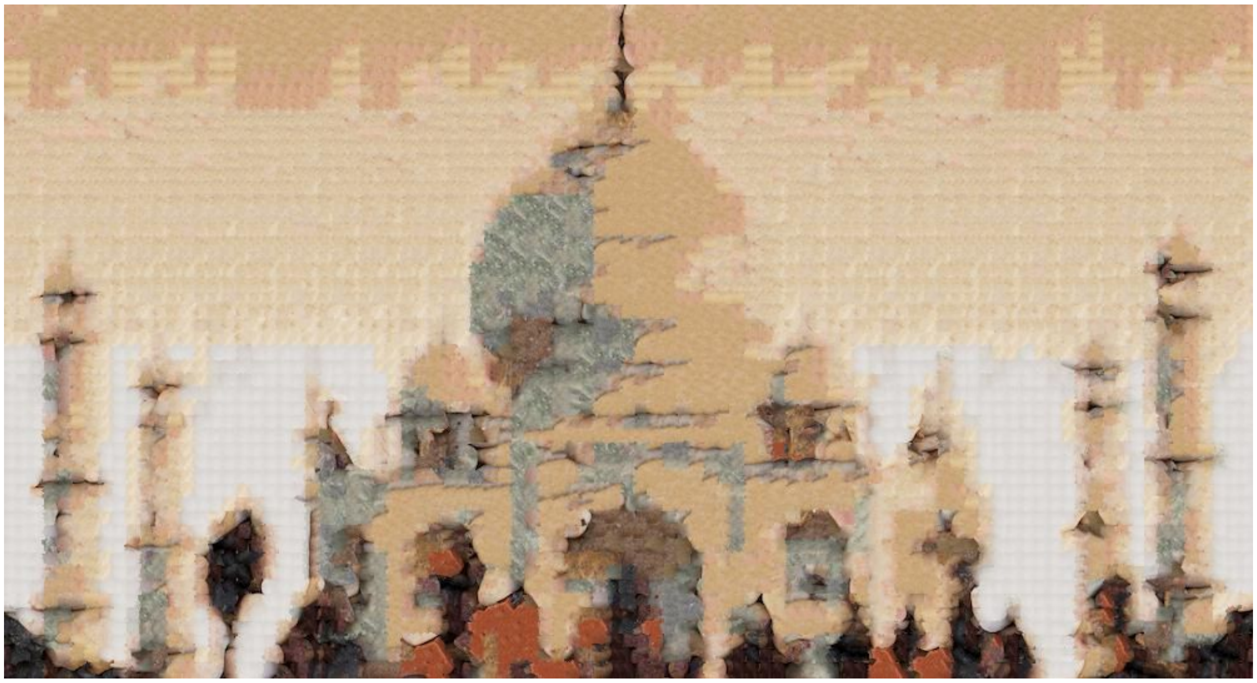


Iteration 4

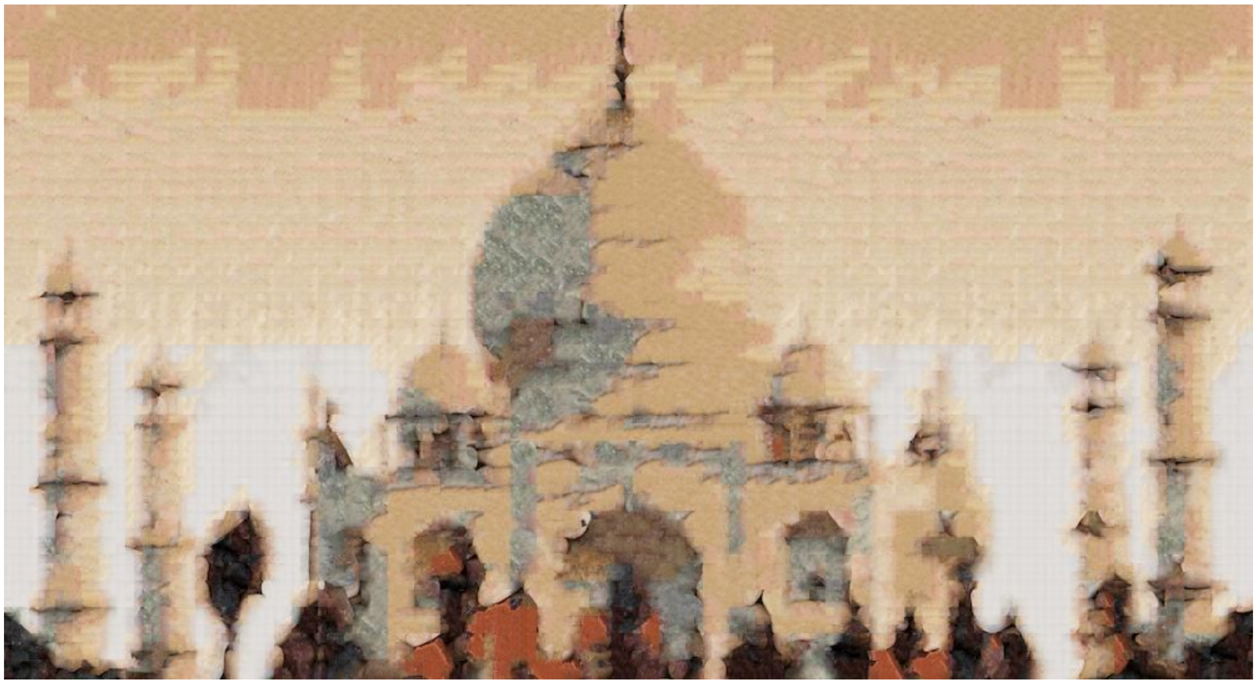
Iteration 5



Iteration 6



Final Output

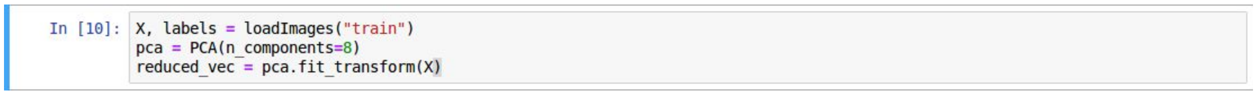


Further Experiment to check similarity of various texture images being synthesised

Idea : A classifier is created using 3 texture images and using the Linear SVM classifier.

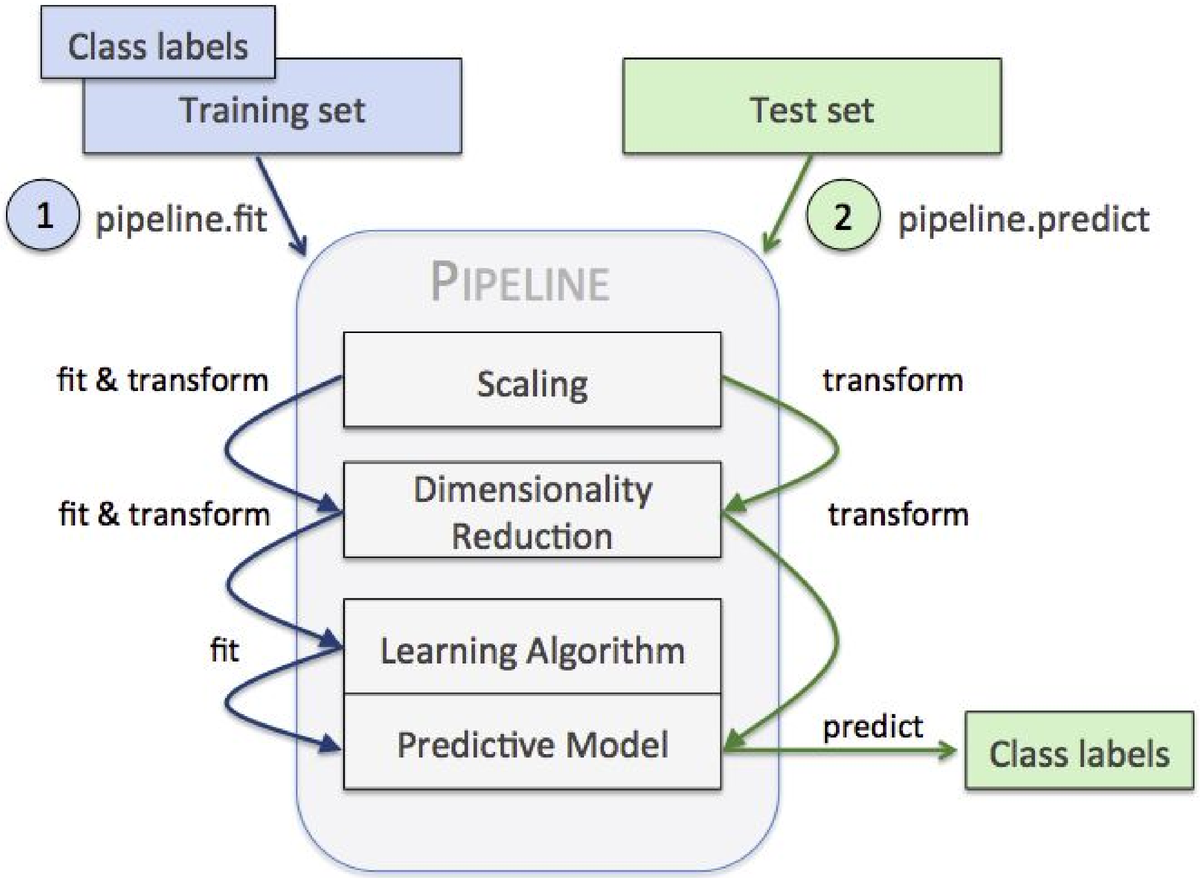
- Multiple random images are generated from the given texture images to be treated as the training data.
- PCA is done on the images treating each image on the training data
- A test set is generated using the same procedure and random values.
- We observe an accuracy of 100% on the test set showing the goodness of our algorithm via the great separability of the data generated using the outputs.

We get 100% accuracy not just while training the data without reduced complexity but even after taking the PCA with number of components as 8.



```
In [10]: X, labels = loadImages("train")
         pca = PCA(n_components=8)
         reduced_vec = pca.fit_transform(X)
```

PCA Pipeline



CLASSIFICATION REPORT

```
             precision    recall  f1-score   support

       rocks       1.00      1.00      1.00        15
  jelly_beans      1.00      1.00      1.00        16
       fence       1.00      1.00      1.00        19

   micro avg       1.00      1.00      1.00        50
   macro avg       1.00      1.00      1.00        50
weighted avg       1.00      1.00      1.00        50
```

Conclusion : This experiment emphasizes the use of texture synthesis in data generation. This can be highly useful when we need large amounts of data from a small set of sample images.