Texture Synthesis

D.Gomathi, Rajvi Shah

In this report we aim to document the work done in our project. Beginning with the introduction to textures and texture synthesis the report covers a brief overview of problems where texture synthesis can be used as a part of the solution, brief description of work previously done in this field and milestones and chosen algorithms with necessary details. The report concludes with the results of the algorithms and discussion about the results.

I. Introduction

Texture as a noun it is described in vocabulary as, (a) something composed of closely interwoven elements; specifically a woven cloth. In other words, textures are usually referred to as visual or tactile surfaces composed of repeating patterns, such as a fabric.

In the domain of visual information processing, the word texture has a wider meaning. However, since a majority of natural surfaces consist of repeating elements; this narrower definition of texture is still powerful enough to describe many surface properties. Since natural textures may contain interesting variations or imperfections, certain amount of randomness over the repeating patterns should be allowed. For example, a honeycomb texture is composed of hexagonal cells with slight variations of size and shape of each cell. The amount of randomness can vary for different textures, from stochastic (sand beach) to purely deterministic (a tiled floor). Textures can be obtained from a variety of sources such as hand-drawn pictures or scanned photographs.

The goal of texture synthesis can be stated as follows: Given a texture sample, synthesize a new texture that, when perceived by a human observer, appears to be generated by the same underlying process. Thus synthesized output texture is perceptually similar to the input texture but also ensures that the result contains sufficient variation. This cannot be achieved by simply tiling the given input texture several times. The blockiness in the result is clearly perceivable. *See Fig 1.1*

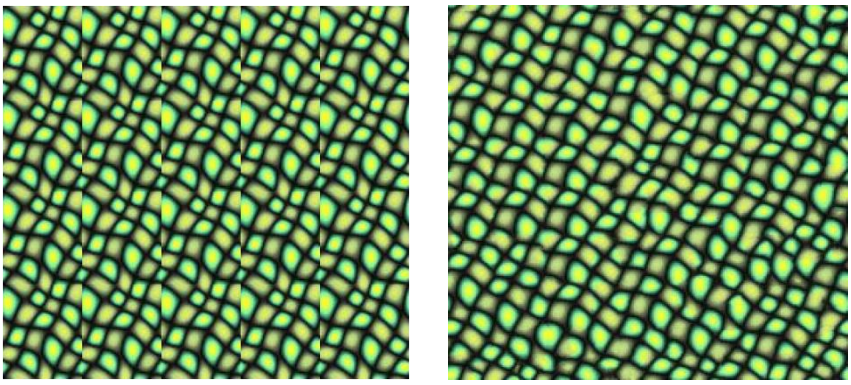


Figure 1.1. (Left) Texture produced by tiling (Right) Texture produced by synthesis

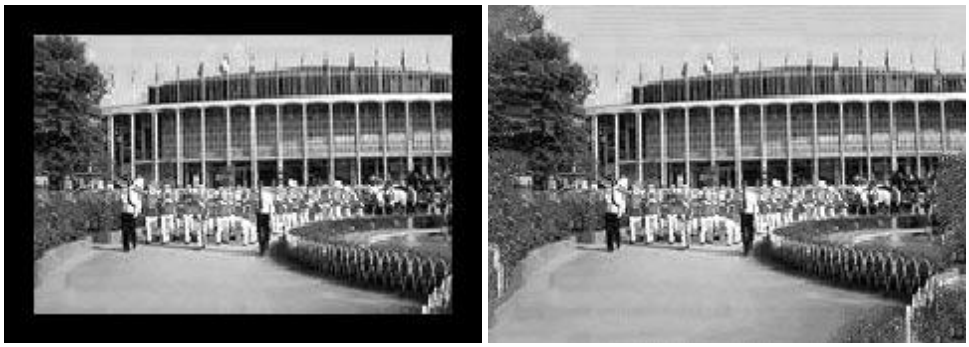
Hence problem at hand can be formulated as,

- Given a sample texture synthesize a new texture that looks like the input
- The synthesized texture can be of arbitrary size specified by the user.
- It should not have visible artifacts such as seams, blocks and misfitting edges.
- Also it should not repeat i.e. the same structures in the output image should not appear at multiple places.

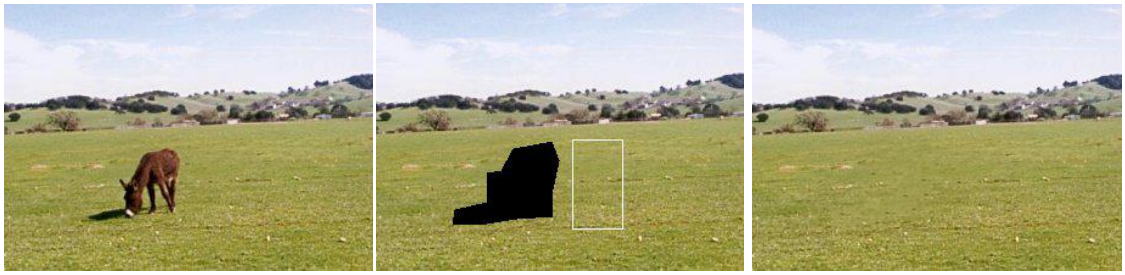
II. Potential Applications of Texture synthesis

Other than large surface creations, there are many other areas where texture synthesis can be applied as a solution. Like hole-filling, Image-video compression, foreground removal etc. Some of these applications are presented here.

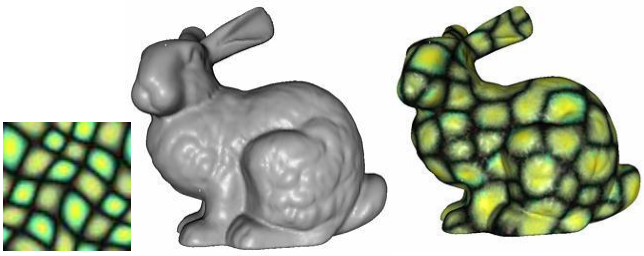
1. Occlusion Fill-in or image extrapolation



2. Foreground Removal



3. Surfacing objects or texture transfer



III. Literature Survey

The first paper in this area was published in 1993 by Popat and Picard [2]. But the idea had been around for many years. In his 1948 article, Claude Shannon mentioned an interesting way of producing English-sounding written text using n-grams. The idea is to model language as a generalized Markov chain: a set of n consecutive letters (or words) make up an n-gram and completely determine the probability distribution of the next letter (or word). One can then repeatedly sample from this Markov chain to produce English-sounding text. The algorithm actually generated some interesting results like "I spent an interesting evening recently with a grain of salt". Popat extended this idea to two dimensions for texture synthesis.

This genre of synthesizing texture falls under the category of pixel-based synthesis using markov random fields. Other than this, major approaches that have made a large impact are pyramid based texture synthesis methods and recently patch based synthesis methods. Recent work in this domain is concentrating more on making available procedures computationally efficient and real-time rather than refining the procedures. The details about publications in the area of texture synthesis can be found at [1]

IV. Approaches to be explored

Out of many algorithms proposed and presented, here we have tried to implement three of them. The reasons behind choosing those papers are, three vary in the fundamental techniques they use and they are milestones of the work done in this domain. [3] is pixel-based method whereas [5] is a method which works on multi-resolution pyramid. And [6] uses patch based approach as opposed to pixel-based approach.

Here, we have given complete implementation details of the algorithms.

1) Pixel – Based Methods

Pixel-based algorithms synthesize a new texture pixel by pixel, with the value of each new pixel determined by its local neighborhood. Among various methods proposed methods based on Markov Random Fields give most pleasing results with moderate computation speed.

Markov Random Field methods model a texture as a realization of a local and stationary random process. That is, each pixel of a texture image is characterized by a small set of spatially neighboring pixels, and this characterization is the same for all pixels.

The image is stationary if, under a proper window size, the observable portion always appears similar. The image is local if each pixel is predictable from a small set of neighboring pixels. *See fig 4.1.*

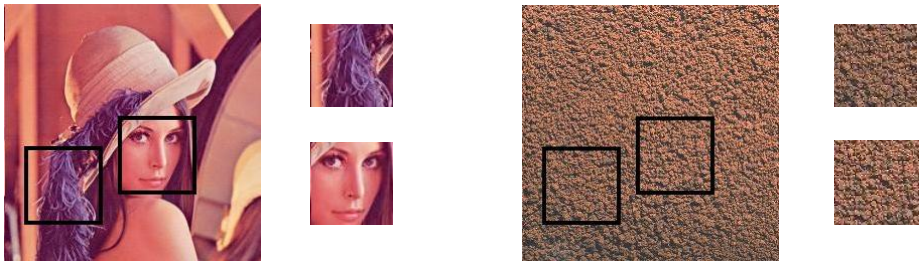


Figure 4.1 (a) Non-texture and (b) Texture images
Texture as a spatially local and stationary process

Different regions of a texture are always perceived to be similar (b1,b2), which is not the case for a general image (a1,a2). In addition, each pixel in (b) is only related to a small set of neighboring pixels. These two characteristics are called stationarity and locality, respectively.

Here, algorithm which operate on pixel based approach with MRF proposed by [3] is described.

1. Non-parametric Synthesis using markov random field [3]

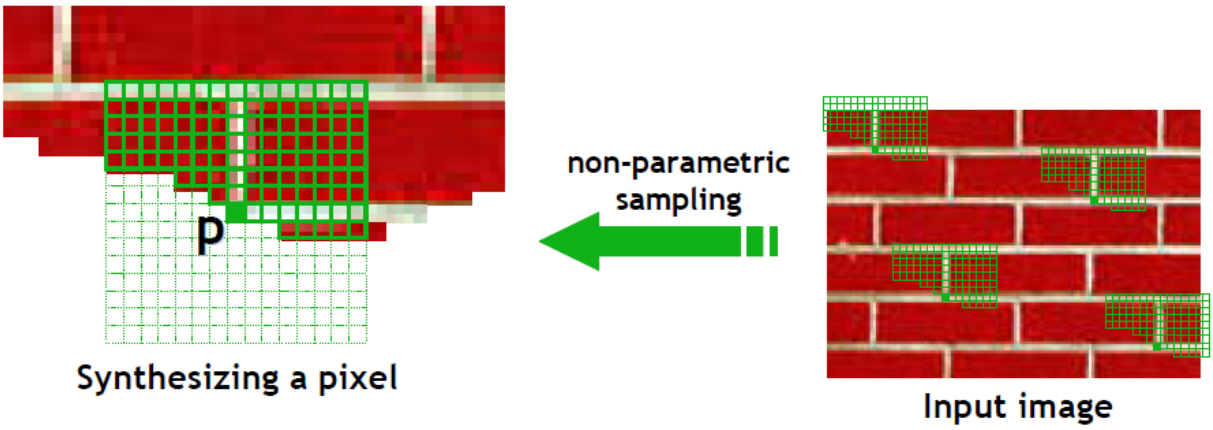


Figure 4.2 Neighborhood search operation

- First, the output texture is seeded from a portion of the input, like a 3 × 3 patch. Starting from this seed, it generates new pixels outward in a spiral fashion.
- For each output pixel under investigation, a fixed-size window (with size picked by the user) centering on the pixel is intersected with already synthesized pixels. Window size is carefully chosen considering the unit size of texture.
- This collection of intersection pixels is then searched throughout the input image to find the most similar N candidates.
- Finally, a random candidate is drawn uniformly from this candidate set, and the input pixel centered on this neighborhood is copied to the target pixel.
- This process is repeated for each not-yet-synthesized output pixel in a spiral fashion until the entire output is synthesized.

Pseudo Code

Main Function

```
function GrowImage(SampleImage,Image,WindowSize)
  while Image not filled do
    progress = 0
    PixelList = GetUnfilledNeighbors(Image)
    foreach Pixel in PixelList do
      Template = GetNeighborhoodWindow(Pixel)
      BestMatches = FindMatches(Template, SampleImage)
      BestMatch = RandomPick(BestMatches)
      if (BestMatch.error < MaxErrThreshold) then
        Pixel.value = BestMatch.value
        progress = 1
      end
    end
    if progress == 0
      then MaxErrThreshold = MaxErrThreshold * 1.1
  end
  return Image
end
```

Subroutine to find matching neighborhood

```
function FindMatches(Template,SampleImage)
  ValidMask = 1s where Template is filled, 0s otherwise
  GaussMask = Gaussian2D(WindowSize,Sigma)
  TotWeight = sum i,j GaussiMask(i,j)*ValidMask(i,j)
  for i,j do
    for ii,jj do
      dist = (Template(ii,jj)-SampleImage(i-ii,j-jj))^2
      SSD(i,j) = SSD(i,j) + dist*ValidMask(ii,jj)*GaussMask(ii,jj)
    end
    SSD(i,j) = SSD(i,j) / TotWeight
  end
  PixelList = all pixels (i,j)where SSD(i,j)<=min(SSD)*(1+ErrThreshold)
  return PixelList
end
```

Implementation Details

It is clear from the pseudo code that these methods use exhaustive searching. Loops in MATLAB are not the best way to implement such procedures as they tend to make the program extremely slow. Here, the given details try to eliminate as many looping structures as possible by either using inbuilt functions or by restructuring the problem.

- First, we initialize output image by placing a small patch (seed) from the input image. Either we can use 'imcrop' to crop the patch or using simple looping it can be done.
- The image is then grown in spiral fashion around the patch. For such a growth the pixel of interest should be chosen from the new boundary in each of the iteration. We need to search for all unfilled pixels which are neighbors of filled pixels. We can implement this without looping using 'imdilate' and 'find'
- After pixel-of-interest for current iteration is chosen, neighborhood mask creation is simple. In the mask, put 1 for all valid neighbors and 0 for yet to synthesize/ unfilled neighbors.
- The most exhaustive loops are the ones for distance calculation. Authors have suggested SSD – sum of square differences as a distance measure. To see how we can compute SSD without/ with minimal looping we have to restructure it.

$$D(x,y) = \sum_{i=-n}^{n} \sum_{j=-n}^{n} \left(S(x+i, y+j) - T(i,j)\right)^2$$

$$D(x,y) = \sum_{i=-n}^{n} \sum_{j=-n}^{n} \left(S(x+i, y+j)\right)^2 - 2 * S(x+i, y+j) * T(i,j) + (T(i,j))^2$$

$$D(x,y) = \sum_{i=-n}^{n} \sum_{j=-n}^{n} \left(S(x+i, y+j)\right)^2 - \sum_{i=-n}^{n} \sum_{j=-n}^{n} 2 * S(x+i, y+j) * T(i,j) + \sum_{i=-n}^{n} \sum_{j=-n}^{n} T(i,j)^2$$

It can be observed that the second term in the image is nothing but twice the correlation. So instead of looping we can implement correlation using 'imfilter'. Last term is constant for all x and y within iteration and hence has to be computed only once. The first term can also be computed with a little trick without using loops. Compute squared sample image at once, and use imfilter 'average' with filter kernel that equal to mask. The only thing to take care of is multiplying by the averaging factor.

2) Pyramid based Texture Analysis-Synthesis [5]

Like pixel based technique, this also requires only example texture and generates output texture. The main concept which makes this method distinguished is pyramid, giving multi-resolution approach. The method is explained below briefly followed by explanation of each sub-procedure.

- The pyramid-based texture analysis/synthesis technique starts with an input (digitized) texture image and a noise image (typically uniform white noise).
- The algorithm modifies the noise to make it look like the input texture.
- It does this by making use of an invertible image representation known as an image pyramid; along with histogram matching that matches the histograms of two images.
- As it iterates the noise histogram begins to match the input texture image.
- We stop the iteration when significant match is there or stop it critically after some threshold iterations.

We shall explain in detail the procedures mentioned above, beginning with image pyramids.

Image Pyramids:
An image pyramid is a type of subband transform in which the basis/projection functions are translated and dilated copies of one another (translated and dilated by a factor or 2j for some integer j).The subbands are computed by convolving and subsampling. For each successive value of j, the subsampling factor is increased by a factor of 2. This yields a set of subband images of different sizes (hence the name image pyramid) that correspond to different frequency bands. The pyramid used is the Laplacian pyramid (a radially symmetric transform).

Laplacian Pyramid.
The Laplacian pyramid is computed using two basic operations: reduce and expand. The reduce operation applies a low-pass filter and then subsamples by a factor of two in each dimension. The expand operation upsamples by a factor of two (padding with zeros in between pixels) and then applies the low-pass filter. In the end, we get a collection of pyramid subband images consisting of several band pass images and one leftover low pass image. These images have different sizes because of the subsampling operations; the smaller images correspond to the lower spatial frequency bands (coarser scales).

Histogram Matching:
Histogram matching is generalisation of histogram equalisation. The algorithm takes an input image and coerces it via a pair of lookup tables to have a particular histogram. The two lookup tables are: 1) the cumulative distribution function of an image and 2) the inverse cumulative distribution function of image. The CDF is a lookup table that maps from the interval [0,256] to the interval [0, 1]. The inverse CDF is a lookup table that maps back from [0, 1] to [0,256]. It is constructed by resampling (with linear interpolation) the cdf so that its samples are evenly spaced on the [0, 1] interval.

Texture-Matching procedure
This is the main procedure. It modifies an input noise image so that it looks like an input texture image.

- First, match the histogram of the noise image to the input texture.
- Second, make pyramids from both the (modified) noise and texture images.
- Third, loop through the two pyramid data structures and match the histograms of each of the corresponding pyramid subbands.
- Fourth, collapse the (histogram-matched) noise pyramid to generate a preliminary version of the synthetic texture.

Matching the histograms of the pyramid subbands modifies the histogram of the collapsed image. In order to get both the pixel and pyramid histograms to match we iterate, rematching the histograms of the images, and then rematching the histograms of the pyramid subbands.

3) Patch – based Texture Synthesis [6]

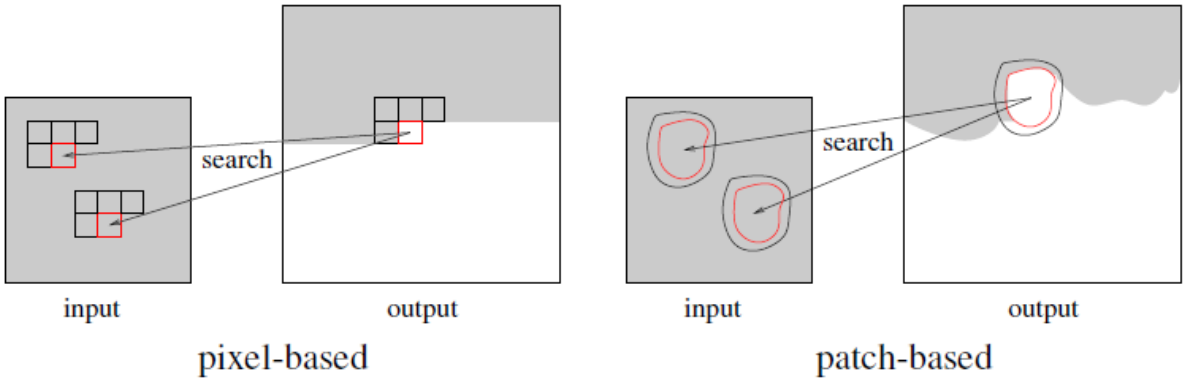


Figure 4.4. Pixel based vs. Patch-based Synthesis

Figure 4.4 illustrates the basic idea of patch-based texture synthesis. In some sense, patch-based synthesis is an extension from pixel-based synthesis. The quality and speed of pixel-based approaches can be improved by synthesizing patches rather than pixels. Patch-based synthesis is very similar to pixel-based synthesis, except that instead of copying pixels, we copy patches. As illustrated in Figure 4.4, to ensure output quality, patches are selected according to its neighborhood, which, just like in pixel-based synthesis, is a thin band of pixels around the unit being copied.

The major difference between the two approaches is that,
In pixel-based algorithm, the copy is just a copy. However, in patch-based algorithms, the issue is more complicated as a patch, being larger than a pixel, usually overlaps with the already synthesized portions, so some decision has to be made about how to handle the conflicting regions. By using patches with irregular shapes, this approach took advantage of the texture masking effects of human visual system and works surprisingly well for stochastic textures. The methods in this genre vary in the way they handle the overlap region.

The complete quilting algorithm is as follows:

1. Initialize the upper-left corner of the output image with a randomly selected patch from the input image.
2. Working left-to-right, top-to-bottom in the output image, repeat the following:
    a) Select the next patch to add from the input image from among the best-fit patches.
    b) Calculate the error surface between this new patch and its area of overlap with already processed patches.
    c) Calculate the minimum cost path through the error surface to determine the patch boundary and then add the new patch to the image.

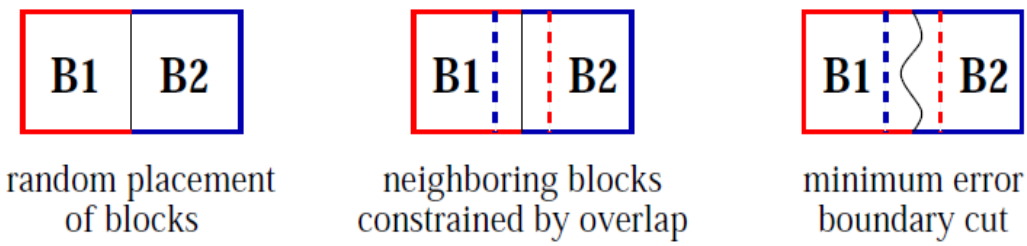


Figure 4.5. Patch-based Synthesis

Step a: Patch selection
During each iteration of processing we need to select the best-fit patch to paste onto the current area of texture we are processing according to some error measure. The procedure for doing this is to perform a brute-force exhaustive search of all possible patches in the input image, calculate the error for each patch, retain all patches that meet the best-fit criteria, and, finally, randomly select one of the retained patches.

Step b: Calculate the error surface
The error surface is calculated is based on the L2 norm.

Step c: Determine minimum-cost boundary cut
Once the error surface is calculated, all of the information necessary to identify the minimum-cost boundary cut to use in order to paste the new patch into the output image is present. We want to make the cut between two overlapping blocks on the pixels where the two textures match best (that is, where the overlap error is low). This can easily be done with dynamic programming (Dijkstra's algorithm can also be used).

The minimal cost path through the error surface is computed in the following manner. If *B*1 and *B*2 are two blocks that overlap along their vertical edge (Figure 4.5) with the regions of overlap *B*1 and *B*2 , respectively, then the error surface is defined as *e* =(*B*1 - *B*2)^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:

$$E_{i,j} = e_{i,j} + \min(e_{i-1,j-1}, e_{i-1,j}, e_{i-1,j+1}) \ (i = 2 \ldots N)$$

In the end, the minimum value of the last row in *E* will indicate the end of the minimal vertical path though the surface and one can trace back and find the path of the best cut. Similar procedure can be applied to horizontal overlaps. When there is both a vertical and a horizontal overlap, the minimal paths meet in the middle and the overall minimum is chosen for the cut. The minimum error boundary cut can then be determined by searching the last (N^{th}) row of the overlap patch, identifying the minimum value, and tracing backwards. The formula for calculating the minimum cost path for a horizontal strip of the overlap patch, working left to right, is readily derivable with minor modifications to the i and j subscripts.

For patches with both vertical and horizontal overlap (i.e. most patches), it is observed that the horizontal and vertical minimum cost paths will, by necessity, meet in the middle and the overall minimum can be chosen as the stopping point for the vertical and horizontal trace backs.

The size of the block is the only parameter controlled by the user and it depends on the properties of a given texture; the block must be big enough to capture the relevant structures in the texture, but small enough so that the interaction between these structures is left up to the algorithm. In our experiment the width of the overlap edge (on one side) was 1/6 of the size of the block. The error tolerance was set to be within 0.1 times the error of the best matching block.

V. Results and Discussion

Out of the three methods we explored, we got good quality outputs for pixel-based and patch based approach. The quality of output is the best for pixel-based as it gives seamless and perceptually same texture but with a big disadvantage of long running time, making it of no practical use. One more key factor governing synthesis in pixel based approach is window size. Ideally the window size should cover basic texture element. For lesser size we get distortion in synthesis process. Refer figure 1. In some natural textures when the basic elements are of varying size, it is difficult to determine the optimum window size. In such cases pixel-based method might fail. Refer figure 8.

Laplacian pyramid based approach works well with isotropic and stochastic textures when we copy the low pass bands as they are and do histogram matching for high pass levels. (Refer to figure 9). There is a scope of improvement in pyramid based method by using steerable pyramid as well as by taking into consideration second order statistics like auto-correlation. But still this approach cannot be expected to produce desired outputs for structured or semi-structured textures.

Patch based method, in most cases give very good output almost equally good as pixel-based with significant improvement in run time landing it useful for many practical problems(Refer figure 10). The problems that can arise with this method are, for big patch size visible artifacts if no good overlap is found and for small patch size texture distortion.

VI. Conclusion

In this report we have given an introduction to textures and application of texture synthesis. We have also discussed about the various methods used for texture synthesis in the literature review section. Texture synthesis algorithms based on pixel, pyramid and patch have been discussed in detail. We then examined the results obtained from each of the algorithm and the observation we made from the results. With patch based algorithm we can produce high quality synthesised texture images very quickly. But still there are many more possible extensions that could improve the results on semi-structured textures. We would be able to produce better results if we can model the texture elements accurately rather than image based approach.

VI. References

1. Dr. Rupert Paget. A brief history of non-parametric texture synthesis. http://www.texturesynthesis.com/history.htm

2. K. Popat and R.W. Picard. Novel cluster-based probability model for texture synthesis, classification, and compression. In Visual Communications and Image Processing, pages 756–68, 1993.

3. Efros and T. Leung. Texture synthesis by non-parametric sampling. In International Conference on Computer Vision, volume 2, pages 1033–8, Sep 1999.

4. Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In Proceedings of ACM SIGGRAPH 2000, Computer Graphics Proceedings, Annual Conference Series, pages 479–488, July 2000.

5. D. J. Heeger and J. R. Bergen. Pyramid-Based texture analysis/synthesis. In R. Cook, editor, SIGGRAPH 95 Conference Proceedings, Annual Conference Series, pages 229–238. ACM SIGGRAPH, AddisonWesley, Aug. 1995.

6. Nealen, A. and Alexa, M. 2003. Hybrid texture synthesis. In Proceedings of the 14th Eurographics Workshop on Rendering (Leuven, Belgium, June 25 - 27, 2003). ACM International Conference Proceeding Series, vol. 44. Eurographics Association, Aire-la-Ville, Switzerland, 97-105.

7. Wei, L. 2007. Part I: fundamentals. In ACM SIGGRAPH 2007 Courses (San Diego, California, August 05 - 09, 2007). SIGGRAPH '07. ACM, New York, NY, 1. DOI= http://doi.acm.org/10.1145/1281500.1281612

8. A.A. Efros and W.T. Freeman, Image Quilting for Texture Synthesis and Transfer, SIGGRAPH 01, Los Angeles, California, August, 2001.

Pyramid Based Texture Synthesis

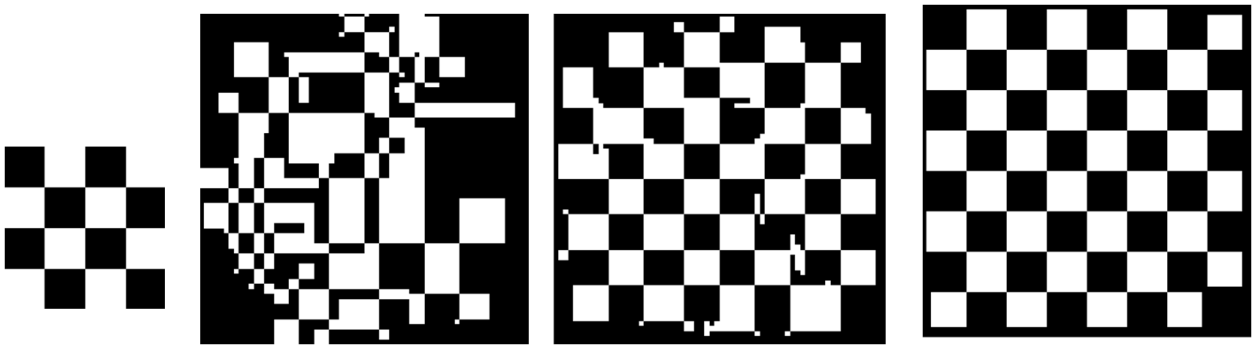


Figure 1. Sample checker board image and synthesis results for window size 5, 15 and 25

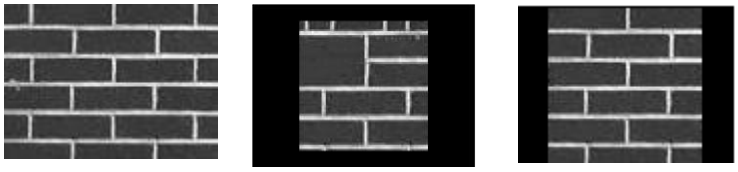


Figure 2. Sample brick wall image and synthesis results for window size 5 and 15

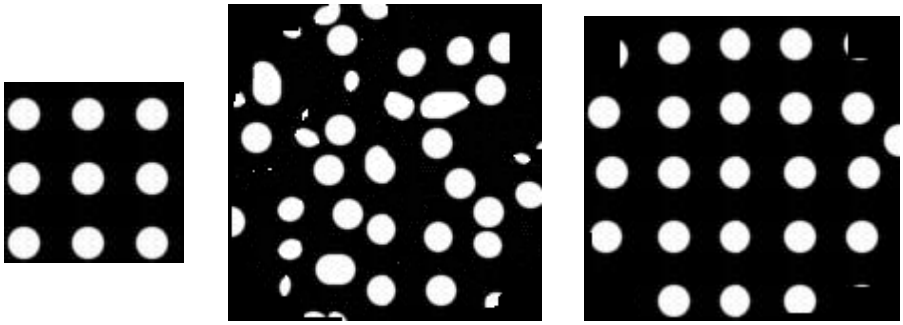


Figure 3. Sample polka dots image and synthesis results for window size 5 and 10
[Run Time details : Input: 90 x 90 , Window-size :10, output texture: 166x166 , Time taken**: 5096.47 seconds**]

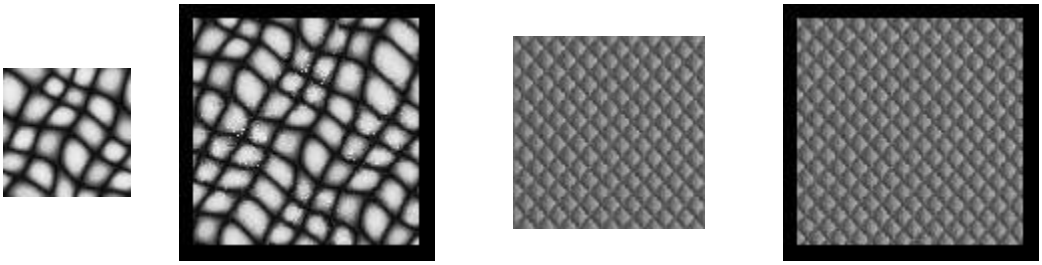


Figure 4. Semi-structured and structured textures

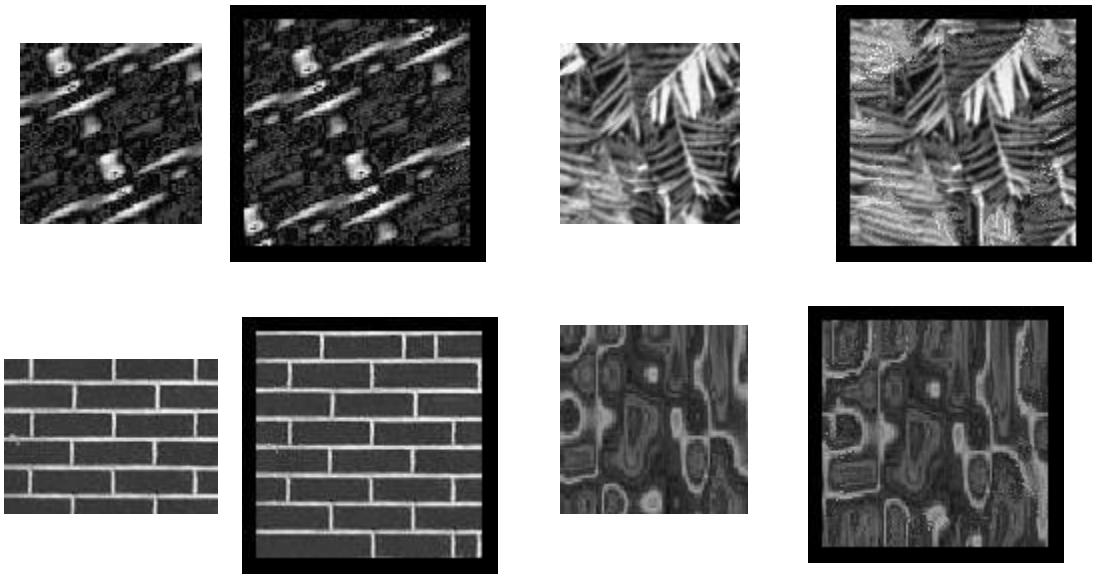


Figure 5. Some more grayscale textures covering a wide variety

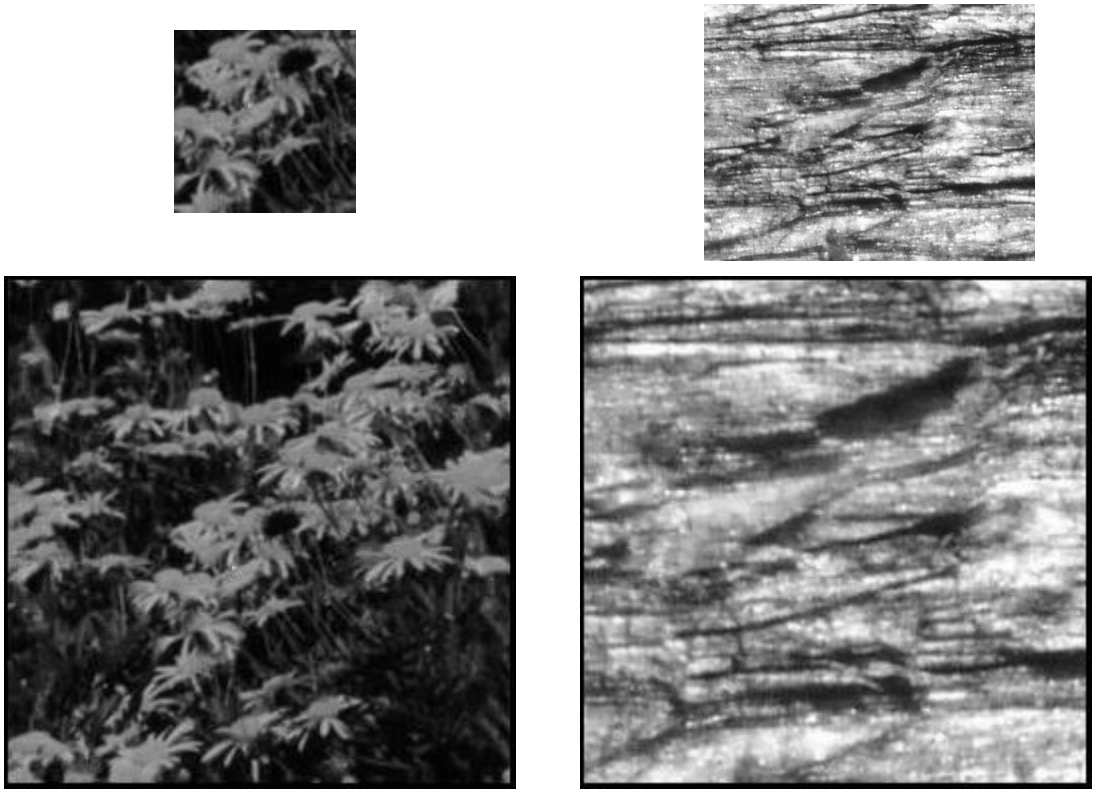


Figure 6. Larger synthesis examples

Figure7. Color Texture Synthesis Examples

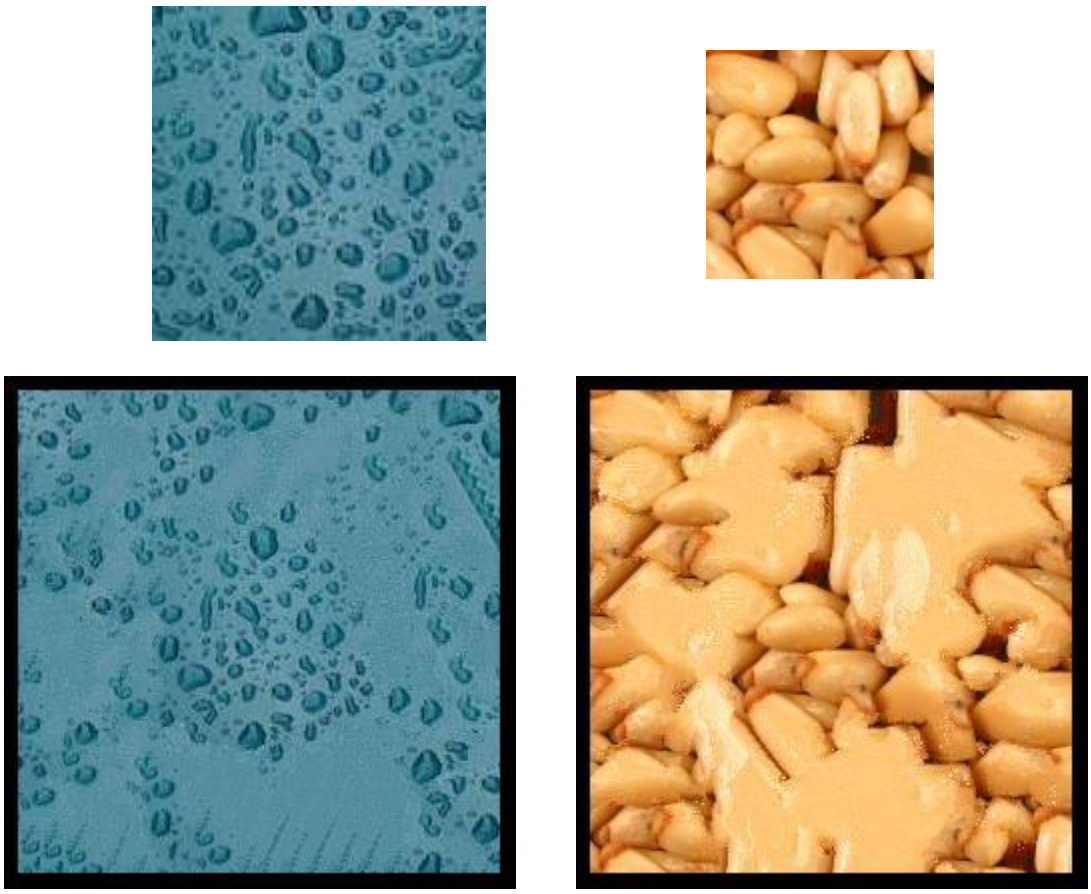
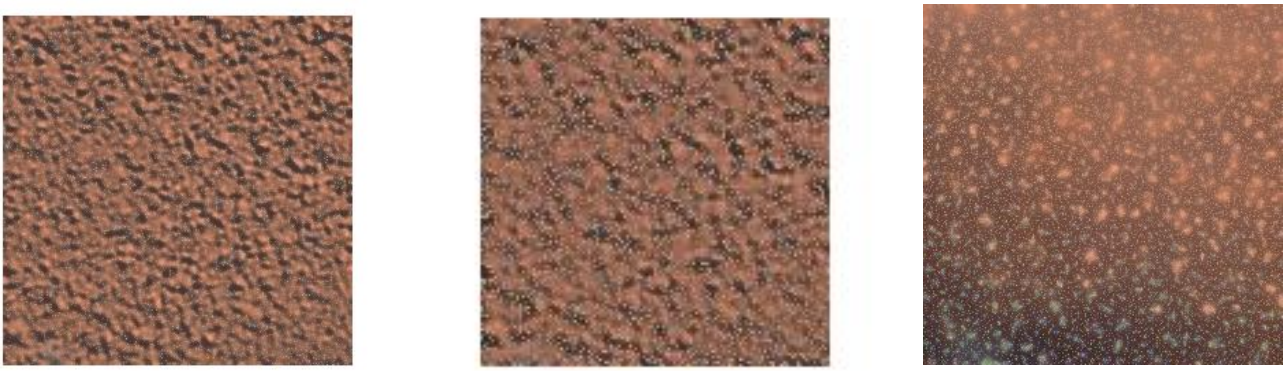


Figure8. Failure Examples

Pyramid Based Texture Synthesis

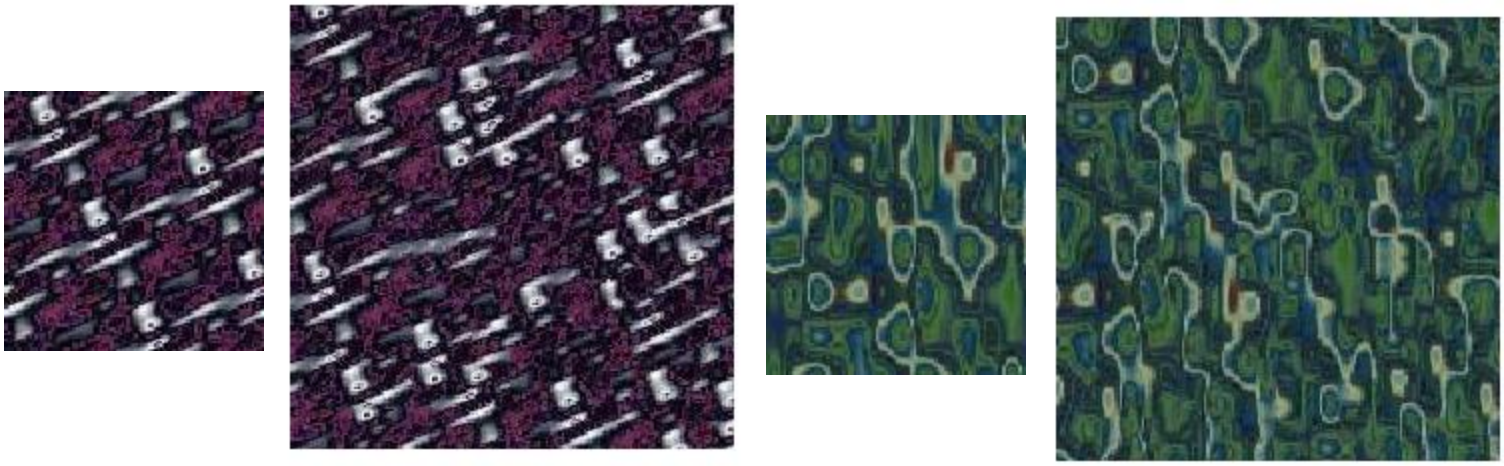


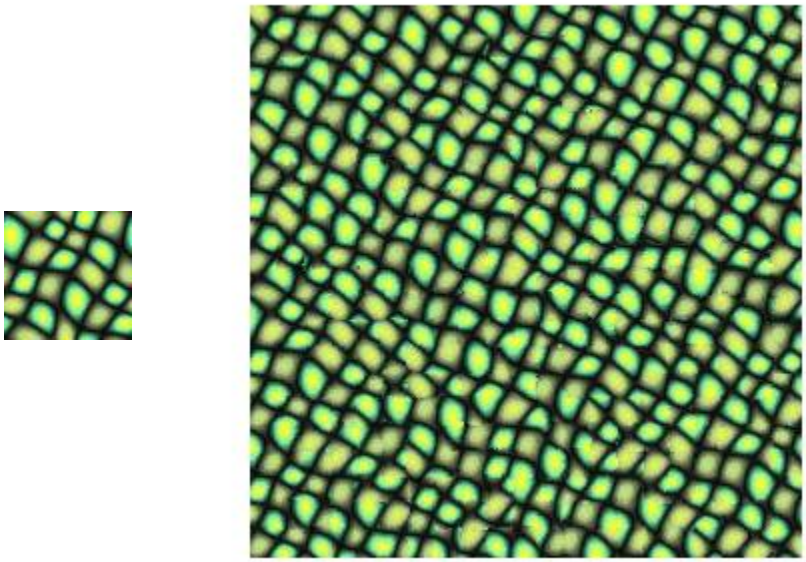
Sample *Noise generated texture by matching top two levels* *Noise generated texture by matching full pyramid*

Figure 9

Patch based synthesis results

Figure 10





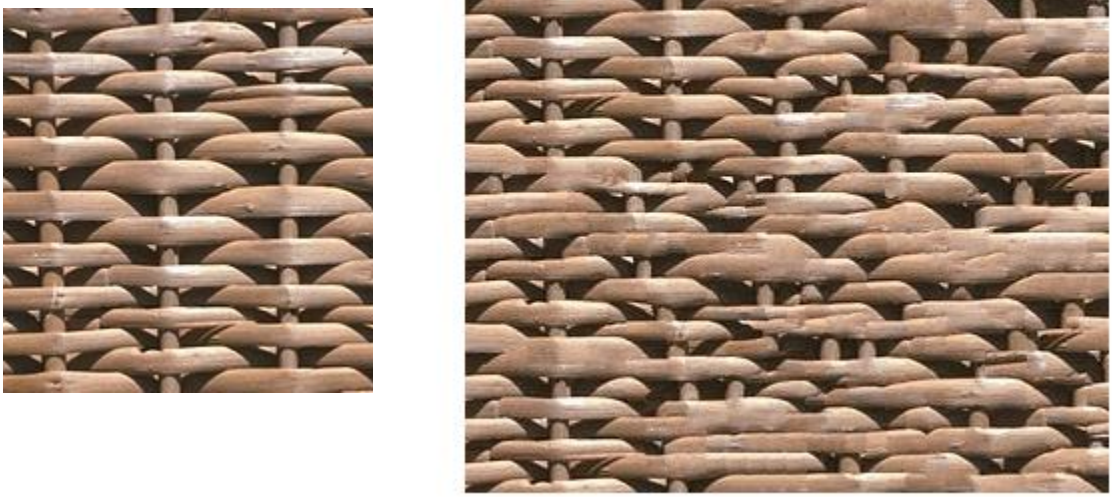


Figure 11.More Results