Peng Tang

Candidate

Computer Science

Department

This thesis is approved, and it is acceptable in quality
and form for publication on microfilm:

Approved by the Thesis Committee:

, Chairperson

Accepted:

Dean, Graduate School

Date

APPLICATION OF NON-PARAMETRIC TEXTURE SYNTHESIS TO IMAGE INPAINTING

BY

PENG TANG

B.S., Computer Science, Tianjin Institute of Textile Science and Technology, 1993
M. S., Mathematics, Nankai University, 1999

THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science
Computer Science

The University of New Mexico
Albuquerque, New Mexico

May, 2004

Acknowledgements

Working on this thesis, I would like to extend my truly appreciation to the following persons for their great contributions of suggestions, ideas, corrections and explanations:

Lance Williams, Associate Professor of Computer Science, the University of New Mexico. For his endeavors as major professor, valuable discussions and ideas. I really appreciate that he spent a lot of holiday time to talk with me and help to correct my mistakes.

Joel Castellanos, Master's student, the Department of Computer Science, the University of New Mexico, Albuquerque, New Mexico. For his explanation of the Castellanos-Williams algorithm, and other discussions.

APPLICATION OF NON-PARAMETRIC TEXTURE SYNTHESIS TO IMAGE INPAINTING

BY

PENG TANG

ABSTRACT OF THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science
Computer Science

The University of New Mexico
Albuquerque, New Mexico

May, 2004

Application of Non-parametric Texture Synthesis to Image Inpainting

by

Peng Tang

B.S., Computer Science, Tianjin Institute of Textile Science and Technology, 1993

M. S., Mathematics, Nankai University, 1999

M.S. Computer Science, University of New Mexico, Albuquerque, New Mexico 2004

ABSTRACT

A non-parametric texture synthesis technique is applied to the problem of digital image inpainting. The technique described primarily handles homogeneous texture images. Based on the Castellanos-Williams algorithm, the method implicitly assumes a Markov random field model for textured image regions. The non-parametric sampling procedure for image inpainting utilizes a series of binary mask Gaussian Pyramid level textures. Texture is synthesized in a coarse-to-fine order. The laplacian pyramid transform is used in the implementation. Resolution hierarchy plays a critical role in the analysis, synthesis and sampling process for efficiency and flexibility. The degree of randomness is crucial in the sampling routine. Here, the sampling process randomly and uniquely chooses a pixel from the initial synthesis guess level, combining with the sampling with-replacement policy. Therefore, the sampling procedure generates a distribution that is very similar to the sample by running more than one time and taking into account the

neighborhood adjusting factor *k.* The combination of these contributes achievements to fulfill the best performance of image inpainting.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1:

Introduction

1.1 Texture

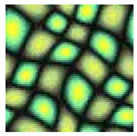
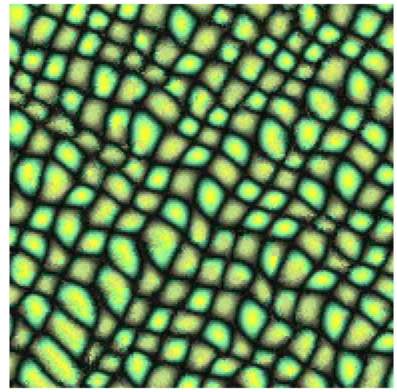
Texture is an attribute of an image, which is characteristic of appearance of a region with a statistically ergodic distribution. Textures can have a wide variety of characteristics, whether spatially homogeneous, or not, and may contain elements repeated either deterministically or randomly. From [1], we know that "there are three principal approaches used to describe texture: statistical, structural and spectral. Typically, the properties of a texture are computed from the grey level histogram of the texture region." Sometimes, the visual appearance of the surface of a fabric or a beautiful painting exhibits an essential texture quality. Some textures are illustrated in Figure 1.

(a) (b) (c) (d)

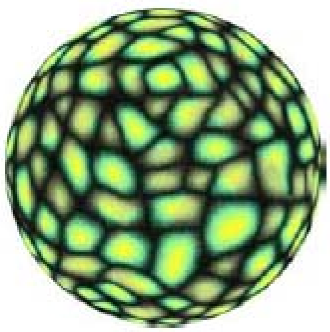
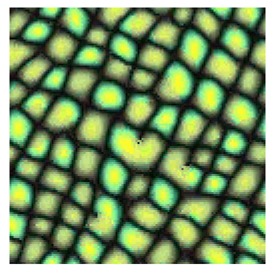
Figure 1: (a) A clover texture can be characterized statistically. (b) A real dinosaur skin example. (c) Texture of a piece of granite. (d) Texture formed by random arrangement of beans, which is isotropic texture at all scales.

1.2 Texture Synthesis

Texture synthesis is the construction of a new image that is different from the original, but it has the same visual appearance as the original. Texture synthesis is another way to create textures distinct hand drawings and images from cameras. Also, synthetic textures can be made of arbitrary size, and visual repetition is avoided. In both computer vision and graphics, texture synthesis has been being an active research field and is involved with many image-processing based applications. Image inpainting is one important application which has used texture synthesis techniques. Sometimes, inpainting has been referred to as hole filling or foreground removal [2]. Other image based applications include image and video compression and animation. Indeed, texture synthesis techniques can be applied anywhere that requires description, analysis and synthesis of an image where the pixels have an ergodic distribution. Figure 2 shows some examples of texture synthesis results from different approaches.

(a) (b)

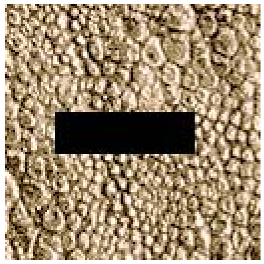



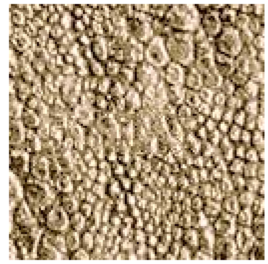
(c) (d)

Figure 2. (a) An artifical sample image. (b) Efros and Leung's synthesized result. (c) Castellanos-Williams result on surface of a sphere. (d) Our approach synthesized result.

1.3 Inpainting

There is no standard definition for image inpainting. Roughly speaking, the purpose of image inpainting is to remove an object from an image. The key point is how to make the image appear like the object never existed. In the literature, another application of image inpainting is described to repair a damaged picture. That means, for a scratched image, how can we restore so that it exhibits its undamaged appearance? Although, manual photographic restoration is an old topic. Digital image inpainting is a technique that restores an image by smoothly filling a hole in a purely automatic fashion. The basic inpainting method involves removing some foreground object or creating a hole in an image. Texture synthesis procedure will then use the neighborhood or background texture to fill in the hole. After synthesis, the picture exhibits its synthesized texture and the hole should be undiscernable. The synthesized texture used to fill the hole comes from the neighborhood or background area that surrounds the hole or damaged area. Figure 3 shows some inpainting examples. In figure 3, (a) and (b) are an example for homogeneous image with hole filling; (c) and (d) are an example for non-homogeneous texture and the object spans different homogeneous regions; (e) and (f) are an example for non-homogeneous texture and the object locates inside one homogeneous region.





(a) (b)

(c) (d)

(e) (f)

Figure 3: (a) and (b) are a real dinosaur skin image. (c) and (d) are inpainting examples in [7]. (e) and (f) are inapinting examples in [10].

1.4 Markov Random Fields – Non-parametric Sampling Methods

Markov Random Fields (MRF) are two-dimensional generalizations of Markov chains. They are used as texture models in many texture synthesis algorithms, which generate textures by a sampling process. MRFs have been shown to be a good approximation for a lot of texture types. Markov random fields capture the spatial structure of a texture by combining generic natural knowledge with a flexible formalism for specifying higher order interactions. Markov random fields models have been successfully used in many fundamental problems of image analysis, texture synthesis, and computer vision. The assumption underlying Markov random fields is that the probability distribution of spatial structure depends on the neighborhoods but is independent of the other parts of the image.

Texture synthesis can be accomplished by a non-parametric sampling process based on neighborhood search method. This method takes a given input sample image and grows a new texture by synthesizing one pixel at a time. In order to determine an output synthesis pixel value, the neighborhood of a synthesis pixel is compared with all neighborhoods of the input sample image. The most similar pixel is then assigned to the output synthesis pixel. Here, the neighborhood size is very important to the algorithm's performance. In order to synthesize the most similar texture possible, the neighborhood size should be large enough to model the local texture information exhibited in the input sample image. However, increasing the neighborhood size causes the algorithm to take more and more time to complete the synthesis so that the process becomes slow. In order to achieve a compromise, a multi-resolution synthesis method is presented, where texture structure can be captured using relatively small neighborhoods. Multi-resolution synthesis schemes are thus faster than single resolution schemes.

Chapter 2:

Related Work

2.1 Efros and Leung

In paper [2], the authors presented a non-parametric sampling technique for texture synthesis, “The texture synthesis process grows a new image outwards from an initial seed, one pixel at a time. A Markov random field model is assumed, ….., the method aims at preserving as much local structure as possible and produces good results for a wide variety of synthetic and real-world textures”[11]. Their method is a simple and efficient texture synthesis method that was based on Markov random fields. In the algorithm, a single pixel p is chosen as the unit of synthesis. Previously synthesized pixels in a square window around p are used as the context. In order to proceed with synthesis, the probability table for the distribution of p conditioned on all possible contexts is needed.

In their work, textures are modeled as a Markov Random Field (MRF). ”They 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. The neighborhood of a pixel is modeled 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.”[11]. In our implementation, the neighborhood of a pixel to be synthesized is also modeled as a square window around the pixel. The values of the pixels are updated one by one. The synthesized pixels are chosen randomly and the size of the neighborhood of a pixel is the only parameter that controls the randomness of the synthesized texture.

2.2 Wei and Levoy

In an extension to the Efros and Leung algorithm, Wei and Levoy developed a multi-resolution neighborhood search based algorithm in [4]. Texture is synthesized in a coarse-to-fine manner. The input consists of an example texture and a random noise image. The process modifies the noise image to make it look like the given example image. By using a search algorithm for best neighborhood match and a multi-level resolution pyramid, the procedure can synthesize as much texture as is needed. The initialization is the first step, then the synthesis process uses two different matching procedures for each Gaussian pyramid level:

1. Initialization: To start, level k of the synthetic Gaussian pyramid is populated with randomly chosen pixel values from level k of the analysis Gaussian pyramid. This initialization step is the source of randomness in the output texture. The other steps focus on the transformation of the output texture into something which resembles the input texture.
2. Multi-level neighborhood matching: The multi-level synthesis transforms each level from lower to higher resolutions, "such that each higher resolution level is constructed from the already synthesized lower resolution levels." [3]. For the first matching step, each pixel of the synthetic Gaussian level is visited in random order, like [2]. By means of an exhaustive search algorithm, the pixel with the most

similar neighborhood pixels is used to replace the pixel in the level being synthesized. However, in order to propagate information from coarse-to-fine scales, the first step neighborhood takes two levels of the Gaussian pyramid, it uses neighborhood pixels including level k and level k-1.

3. Single-level neighborhood matching: This synthesis process is to make sure the synthesized pixel retains as much local texture information as possible. The final value of the synthesis output pixel at level k is determined by repeating the non-parametric sampling procedure used in the first step. However, different than step 2 whose purpose is to project texture information from coarse-to-fine scales, the neighborhood that is used in step 3 consists only of pixels at level k in the analysis Gaussian pyramid.

2.3 Inpainting and Texture Synthesis

The goal of image inpainting is to fill a hole in an image so that the hole is undetectable by observers. In the literature, there are three kinds of works related to digital inpainting. First is the restoration of films [8], second is related to texture synthesis [9], the last is related to disocclusion [10]. For the first technique, it cannot be applied to still images or to films where the regions to be inpainted span multiple frames. For the third technique, [10] showed that the algorithm can be viewed as solving a set of differential equations. "The fill-in is done in such a way that isophote lines arriving at the regions' boundaries are completed inside. …, no limitations are imposed on the topology of the region to be inpainted." [10].

For the second technique, there are many different ways to do inpainting with texture synthesis. For example, [9] combines frequency and spatial domain information in order to fill a given region with a chosen texture. Inpainting with texture synthesis can roughly be separated into two parts: Computing the information of pixels in relation to the neighboring pixels; finding the optimal pixel with matched neighbors by estimating the distance measure. Traditional inpainting algorithm is still useful since it performs quite well for small scratches and runs relatively fast. The texture synthesis method brings up some extra overhead and runs in time proportional to the size of the image.

In recent years, pyramid-based multi-resolution texture synthesis has been adopted and used in many texture syntheses and inpainting algorithms. The Laplacian pyramid is the one that has been used widely, e.g., [4], [5]. A detailed explanation of the Laplacian pyramid transform will be presented in Chapter 4. Also, the Castellanos-Williams algorithm used in this thesis will be explained in detail in the next chapter. Finally, the similarities and differences between our method and Castellanos-Williams will also be discussed there.

Chapter 3:

Castellanos-Williams

3.1 Overview

Our work is mainly based on the Castellanos-Williams algorithm, so in this part, I will give a detailed explanation of the Castellanos-Williams method as it relates to our work. The Castellanos-Williams algorithm implemented in [11] focuses on to synthesis texture from rectangular to surface and assumes the sample textures are isotropic, so they do the triangulation on the sample texture in order to have a good mapping from the rectangular sample to triangular grids on the surface. In our implementation, the sample texture will be synthesized from rectangular to rectangular shape and we do not assume the sample texture is isotropic, so triangulation does not need in our method.

Although, the Castellanos-Williams algorithm for texture synthesis on surfaces is inspired by several multi-resolution methods [11], synthesis texture image by using non-parametric procedure to sample an implicitly defined Markov Random Field, the main inspiration came from [4]. [2] also influenced to some extent. However, Castellanos-Williams technique differs from other methods in some significant ways: The Castellanos-Williams method generates texture coarse-to-fine in a more efficient way by using a Laplacian pyramid transform; the non-parametric sampling procedure in [4] has been replaced with a routine which better preserves the first-order statistics of the input texture. Finally, they resample the input texture using a triangulated grid so that the geometry more closely resembles the subdivision surface on which the texture will be synthesized. This also simplifies the optimal pixel searching process of the texture synthesis procedure.

3.2 Coarse-to-fine Information Flow

Unlike [4] who calculate just the Gaussian pyramid pixels of the target texture, the Castellanos-Williams method computes both the Gaussian and Laplacian pyramid pixels. Unlike the Gaussian pyramid alone, the Laplacian pyramid can be inverted. That means, the original function can be reconstructed from the Laplacian pyramid and a single residual Gaussian value. In the Castellanos-Williams algorithm, they synthesize both Gaussian and Laplacian pyramids in order to propagate information from coarse-to-fine scales. The biggest difference between Castellanos-Williams and Wei-Levoy is that the Castellanos-Williams algorithm returns both Gaussian and Laplacian pyramid pixels as the synthesis process proceeds. Every time, one optimal Gaussian pixel is copied to the synthesis side, at the same time the four children of the corresponding Laplacian pyramid are copied simultaneously during the synthesis process.

In the Castellanos-Williams algorithm, the Laplacian pyramid transform plays an important role. We know for each Gaussian pyramid level, there is a corresponding Laplacian pyramid level which has twice the resolution. Suppose the Laplacian pyramid transform transforms the target input image into m levels, the first level is 1, the second is

2, and this continue until level m. We call the original Gaussian and Laplacian levels from the input texture the *analysis side*, the intermediate synthesis texture is the *pre-sampling Gaussian level*, and the post sampling Gaussian and Laplacian is the *post-sampling image*. The *pre-sampling* and *post-sampling* belong to *synthesis side*. The Castellanos-Williams algorithm first takes the coarsest Gaussian level k and uses it to initialize the initial *pre-sampling* synthesis level pixels by randomly choosing pixels from the input Gaussian level k. Then the choosy sampling procedure uses this Gaussian level and the initial *pre-sampling* synthesis level texture to do the exhaustive neighborhood matching process. Each time an optimal matched pixel is found, the RGB values of this pixel is copied to the *post-sampling* synthesis Gaussian level and simultaneously the four children of Laplacian level k-1 are copied to the corresponding Laplacian pyramid level. Finally, the algorithm will *project* the *post-sampling* Gaussian level k, and then add the projected Gaussian level and the corresponding Laplacian level. This becomes the next *pre-sampling* synthesis level texture, and is a better approximation to the target texture than the texture at the previous level. The algorithm repeats this procedure until it reaches the finest level. After the algorithm reaches the finest level, the method will do the sampling process by using the input texture as the analysis image and then sampling the first *pre-sampling* synthesis level texture, because there is no Laplacian pyramid pixels to be copied. When this step is done, the final output image will be the Castellanos-Williams texture synthesis output texture and will have the appearance of the input sample texture. During the whole algorithm, the exhaustive neighborhood matching will be very time consuming, the implementation described in [11] does all levels of synthesis in one whole program code. In my implementation, I have separated all levels of synthesis from one whole program code into level-based program code. For each level, the code runs independently, this strategy enhances the performance.

The Castellanos-Williams algorithm only randomizes the coarsest *pre-sampling* resolution synthesis with random values from the coarsest analysis Gaussian level, this randomization is transported through the entire algorithm to the finest resolution level. In contrast, [4] randomizes each level according to the input sample texture. Figure 4 demonstrates an example of Castellanos-Williams Texture Synthesis technique operating on a rectangular texture region.

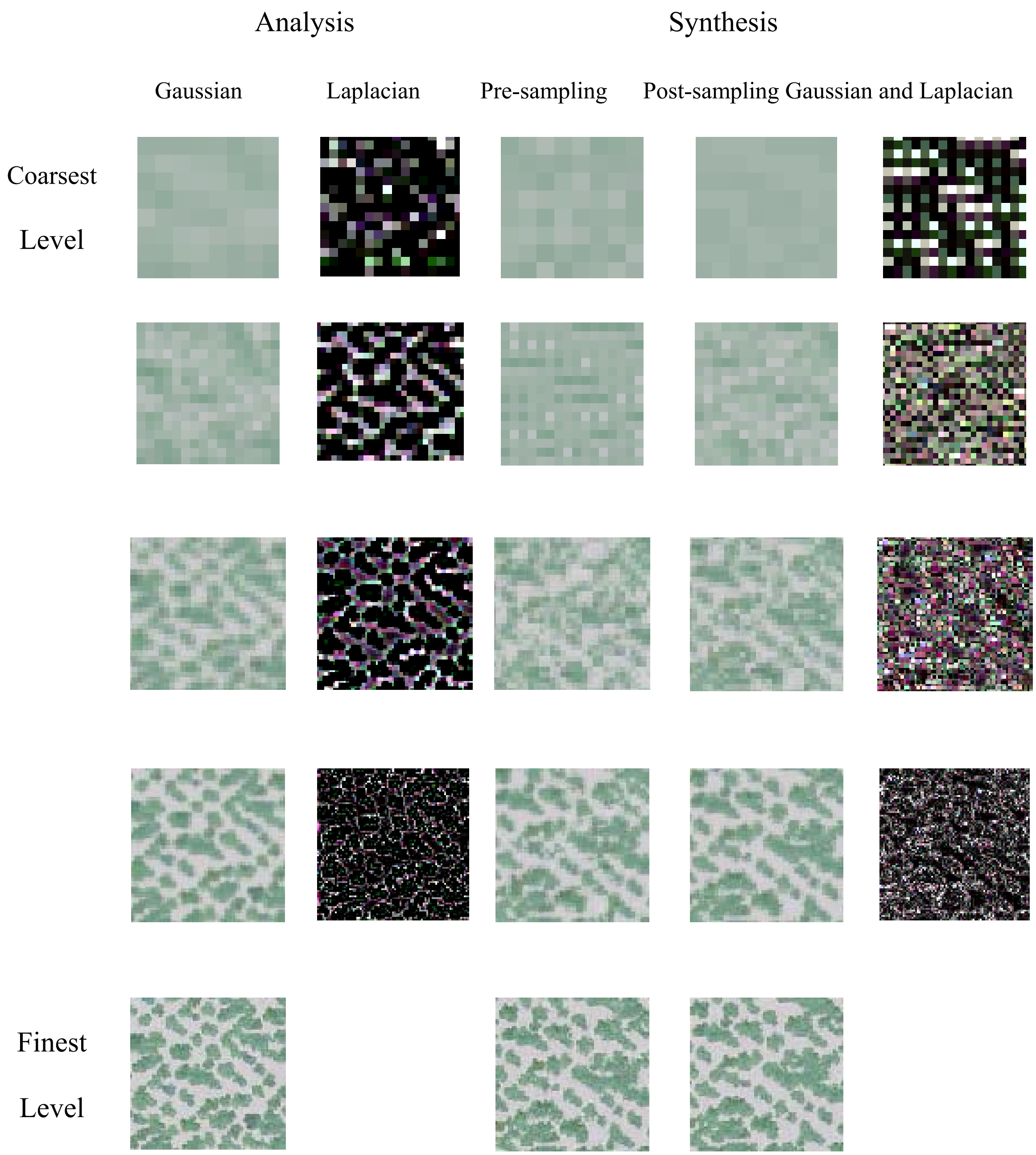


Figure 4: Texture analysis and synthesis. From top to bottom, the level from k to 1, the texture information proceeds from coarse-to-fine. In the first step, the *pre-sampling* texture is initialized with randomly chosen pixels from the coarsest Gaussian level. On the synthesis side, subsequent steps will use the previous *post-sampling* Gaussian and Laplacian level textures to generate the *pre-sampling* level texture for each step. The final

step use the original input as the analysis texture, and there is no Laplacian level in this step. The output will be the final texture synthesis result.

Chapter 4:

Modification of Castellanos-Williams Algorithm

4.1 Analysis

In the Castellanos-Williams algorithm, the main intention is to synthesize texture from raster image to a surface represented as a triangulation, so they have changed the traditional Laplacian pyramid transform and made it suitable for transforming surface. Also, when doing the neighborhood matching, they consider six rotational phases of the input sample each rotated 60 degree relative to the previous. Finally, the Castellanos-Williams method maps the square input pixels to triangular faces on surface. In addition, the Castellanos-Williams method can synthesize a bigger image than input sample because they use a scaling factor to adjust the synthesis texture scale corresponding to the input sample.

In our approach, the goal is to do inpainting using texture synthesis process. The mapping is from square input pixels to square output pixels. During the process, the hole in the texture will be filled in using texture from the surrounding region. In order to finish the inpainting process, the Castelanos-Williams synthesis technique plays an important role. However, inpainting is different than pure texture synthesis. In particular, we need to preserve the image size, so a scale factor isn't needed. Also, for the neighborhood matching procedure, in the Castellanos-Williams implementation, they do "wild-cards" for boundary neighborhood matches. Basically, they assume that if the target pixel doesn't have a full complement of neighborhood pixels, they use a fixed value to describe the missing neighborhood pixels on both the analysis and the synthesis sides, and when calculating the distance from the neighborhood, since on both sides they have the same pixel values for the missing pixels, the difference will be zero during the distance computation; the weighting process will be based on just the existing neighbors not the missing neighbors. Finally, the rotation step in the Castellanos-Williams implementation is not used in the inpainting algorithm since we do not assume that the sample texture is isotropic.

For the Castellanos-Williams implementation, the triangulation increases the computational workload considerably. The 2-distant neighborhood is approximately 6×6 neighborhood rectangular faces in size for the coarsest level Laplacian pyramid transform level and the sample is typically 8×8 in size. In inpainting, the neighborhood size is 7×7, so for a color image with 3 RGB values, one pixel has 48 neighbors with 3 RGB values. Consequently, there will be a 48(3)=144 dimensional neighborhood vector. The error distance computation will calculate the Euclidean distance between the two 144 dimensional neighborhood vectors of the analysis and synthesis target pixels. The 7×7 neighborhood showing the Euclidean distance from the center is as follows:

35 (3√2)	34 (√13)	33 (√10)	32 (3)	31 (√10)	30 (√13)	29 (3√2)
36 (√13)	15 (2√2)	14 (√5)	13 (2)	12 (√5)	11 (2√2)	28 (√13)
37 (√10)	16 (√5)	3 (√2)	2 (1)	1 (√2)	10 (√5)	27 (√10)
38 (3)	17 (2)	4 (1)	*	0 (1)	9 (2)	26 (3)
39 (√10)	18 (√5)	5 (√2)	6 (1)	7 (√2)	8 (√5)	25 (√10)
40 (√13)	19 (2√2)	20 (√5)	21 (2)	22 (√5)	23 (2√2)	24 (√13)
41 (3√2)	42 (√13)	43 (√10)	44 (3)	45 (√10)	46 (√13)	47 (3√2)

Table 1: 7×7 Neighborhood Illustration

The digital number from 0 to 47 is the 7×7 neighborhood order. The number inside the bracket shows the basic Euclidean distance *d* between the neighborhood pixel and the center synthesis pixel. These distances are used for Gaussian weighting of distance. Later, we will show where and how to use this distance parameter.

4.2 Details

In addition above discussion to the texture synthesis to image inpainting, there are additional steps to apply which need to be done. The hole is represented by the value zero. Consequently, we need to do a gray scale transformation of the image to take intensity from the range 0-255 to the range 1-255. We use the image processing software package called Gimp to remove the foreground object, and fill the hole grey scale value with zeros. The mask image is a binary image with two colors, zero denotes black pixel values and 255 denotes white pixel values. The hole is black with pixel value zeros. Outside the hole is white with pixel value 255s. The Laplacian pyramid transform is done on the mask binary image as well as on the input texture. Finally, from the coarsest level to the finest level, when doing texture synthesis at each step of the pixel matching, the value of the mask pyramid acts like a switch, deciding whether a pixel can be picked as the optimal matching pixel and be copied into the texture being synthesized.

4.2.1 Input Texture Pre-processing

For texture synthesis to be applied to inpainting, the first difficulty is that knowing how to detect whether a pixel in the sample image is a pixel inside or outside the hole. We know that the goal of texture synthesis by neighborhood matching is to synthesize a new texture from a sample input image. For synthesis only, you don't need to worry about whether or not to use a pixel in the sample image. The algorithm will automatically pick the pixel using the choosy sampling process by searching through the whole sample texture. By choosing the synthesis pixels from the whole sample, the output image will resemble the original input texture. However, an inpainting algorithm can not choose pixels that are inside the hole of the original texture. The goal of the algorithm is to choose pixels outside the hole of the sample texture as samples and fill in the hole with a texture so that the resulting image looks intact. For this reason, the mask pyramid is applied, and the sample texture needs to be processed. The preprocessing is simple. For a given gray scale sample texture, the pixel values are distributed between 0 and 255, the gray scale transformation is to multiply each RGB value with a fraction (254/255), then adding one to the result. After this transformation, the RGB values of the sample texture will be bounded within 1 to 255. Mathematically, if we suppose that the pixel value of the sample texture is *g(x)*, after the preprocessing pixel value is *f(x)*, then:

$$f(x) \quad = \quad g(x) \times \frac{254}{255} + 1$$

Using this formula, the input image has only been slightly changed.

4.2.2 Analysis and Synthesis

Pyramid-based texture analysis and synthesis techniques have been introduced and used in many image processing applications, e.g., [4, 5, 11,]. Typically, image or texture is decomposed into a set of subbands. The subband transform makes an image to a particular pyramid type, which defines the characteristic of the texture. The subbands are calculated by convolving and downsampling by a factor of two. This produces a series of images with different size, which are called an image pyramid.

The Laplacian pyramid transform can be the basis for an effective pyramid based texture synthesis method. There are two basic operations: *reduce* and *project*. The *reduce* operation uses a low pass filter. The image is convolved with a Gaussian kernel and then subsampled by a factor of two in the width and height dimensions. The *project* operation upsamples the image with a factor two in each dimension. A full level of the pyramid consists of two images, G_k (a low pass image) and L_k (a high pass image):

$$\begin{aligned} G_k &= reduce(G_{k-1}) \\ L_k &= G_{k-1} - project(G_k) \end{aligned}$$

Tle Laplacian pyramid transform is a linear and reversible transformation. The upper level G_{k-1} image can be reconstructed from G_k and L_k:

$$G_{k-1} = L_k + project(G_k)$$

So the original image can be constructed as:

$$Original\ image = L_1 + project(G_1)$$

The above Laplacian pyramid transform representations G_k and L_k are also described as Gaussian pyramid level and Laplacian pyramid level. Figure 5 is an example of Gaussian and Laplacian pyramid levels of Laplacian pyramid transform.

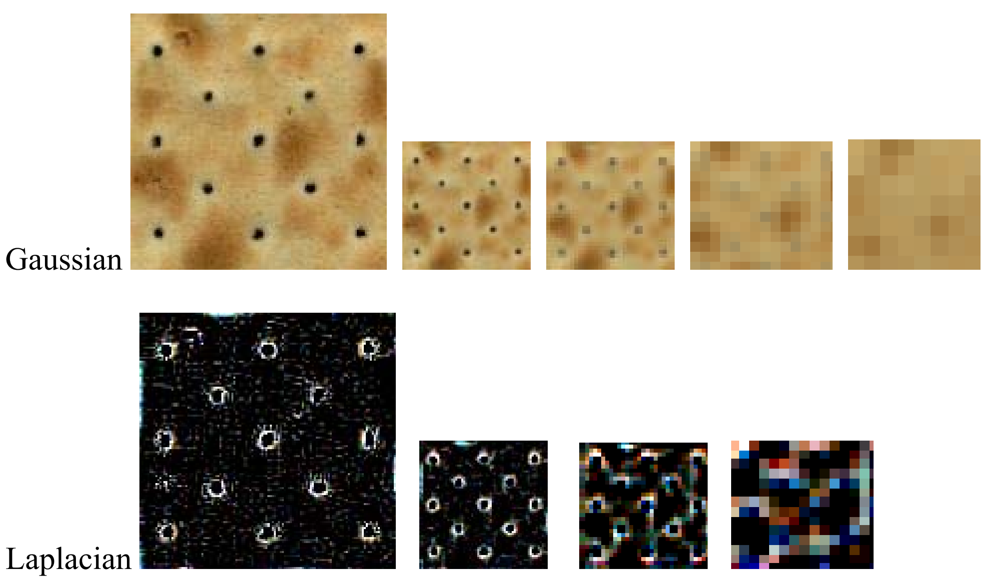


Figure 5: Example of Gaussian and Laplacian pyramid levels

4.2.3 Neighborhood Matching Details

Texture is an important attribute of images, especially for computer graphics, it increases the visual realism of images. Texture synthesis is a useful way to produce a texture image of an arbitrary size. Neighborhood matching is the crucial part in texture synthesis. The first step for neighborhood matching is to define the size of the neighborhood. In our implementation, the 7×7 neighborhood is defined as indicated by the pattern in Table 1. The spiral ordering from 0 to 47 around the central pixel enumerates the 48 neighbors for a simple pixel in the image. The basic Euclidean distance of a neighbor pixel is calculated according the Euclidean distance computation formula and the pixel's coordinates in the image representation, which assumes two adjacent pixels' Euclidean distance is unit. So if $\boldsymbol{P_1}(x,y)$ and $\boldsymbol{P_2}(u,v)$ are two points in an image, the basic Euclidean distance will be as follows:

$$d(P_1, P_2) = \sqrt{(x-u)^2 + (y-v)^2}$$

All the basic Euclidean distances of the 48 neighbors are calculated and shown in Table 1.

As shown in Table 1, a 7×7 neighborhood is comprised of 48 neighborhood pixels. For a color image, each pixel has 3 RGB values. Consequently, there will be 3(48) = 144 values per neighborhood. If using an array to express these values, it can be treated as a 144 dimensional vector. The localized texture is captured by weighting the error distance through a Gaussian function between the two vectors of analysis side and synthesis side neighborhood. The neighborhood weighting scale factor is defined as:

$$w_i = e^{-\frac{1}{2} \times \frac{{d_i}^2}{s^2}}$$

where, d_i is the basic Euclidean distance, σ is the Gaussian function parameter, in our implementation, σ is 4.0. The function of the weighting scale factor is to weight the 48 neighbors of the two pixels between analysis and synthesis pyramid level.

During the implementation of inpainting, in order to avoid generating seam on the boundary of the hole, there is another adjusting factor *k*, which is used to adjust the neighborhood error distance according to whether a neighborhood pixel is located inside the hole or outside the hole. When computing the error distance of the two vectors, first we need to determine the neighborhood pixel position by using the binary mask pyramid image that has the same size as the analysis level image. If a random sampling pixel in the synthesis step is chosen, the coordinates of the pixel are transmitted to the *ErrorDistance()* procedure. The procedure uses the corresponding mask level image to compare the neighborhood pixels' values in the coordinates with 255. If the neighborhood pixel values are not 255, it is inside the hole. If the pixel values are 255, it is outside the hole. After getting the neighborhood pixel position, it works as a switch. If the pixel is inside the hole, the adjusting factor one is used during the Euclidean error distance computation; if it is outside the hole, the adjusting factor *k* is used during the Euclidean error distance computation corresponding to each dimension within the 144 dimensional vector. The Euclidean error distance computation between the two vectors is to make subtraction for the corresponding pixel RGB values, then squares it and sums all the results and does square root on the final result. The working equation is as follows:

$$Errordistance = \sqrt{\sum_{i=1}^{n} a_i \cdot (X_{Ai} - X_{Si})^2 w_i}$$

$a_i \in \{1, k\}$ is an indicator variable. If a pixel is inside of the hole, a_i takes value 1; if a pixel is outside of the hole, a_i takes value k. It is decided by the neighbor position in the mask level image whether a pixel is inside or outside of the hole. X_{Ai} denotes the *i-th* coordinate of the analysis neighborhood vector; X_{Si} denotes the *i*-th coordinate of the synthesis neighborhood vector. And, w_i is the neighborhood weighting scale factor as defined in previous equation. By using the adjusting factor k, we have found there is a big improvement on the synthesis texture of inpainting. This method makes the boundary of the hole to be visual appearance seamless in a significant way.

The sampling routine calculates the final distance of the two vectors by multiplying a factor of $1.005^{(C[j]-Cmin\ )}$, here the base is different from the Castellanos-Williams's method. *c[j]* represents the number of times that pixel *j* in the analysis Gaussian pyramid has been selected and copied to the synthesis *post-sampling* pyramid level. c_{min} represents the minimum number of times that any pixel in the analysis Gaussian pyramid level has been selected. The pixel *j* of the target Gaussian pyramid level is selected such that the final distance is a minimum by exhaustive searching through traversal every pixel in the given Gaussian pyramid level. The random order is performed in the sampling procedure, which plays an efficient sampling process. In order to enhance the algorithm performance, we have been running the random sampling procedure with different times and different *k* values. Theoretically, the more times running for the sampling, the better the results for inpainting. In our practice, many results are performed by running 4 times and *k* equals 8 under the combination of σ equals 4.0 and c_{min} base is 1.005. However, the time consuming is exponential increasing with the texture size increasing.

4.2.4 Coarse-to-fine Information Flow with Mask Pyramid

For inpainting implementation, the mask pyramid plays an important role on the texture synthesis process. The mask pyramid is a binary Gaussian pyramid level, all of the pixel values are 0 or 255. Zeros denote the black pixels that consist into the hole. 255s denote the white pixels that consist into the outside part of the hole. In order to computer the error distance of the two vectors, we need to determine the neighborhood pixel position by using the binary mask pyramid level image. If a random sampling pixel in the synthesis step is chosen, the coordinates of the pixel are transmitted to the *ErrorDistance()* procedure. The procedure uses the corresponding mask level image to compare the neighborhood pixels' values in the coordinates with 255. If the neighborhood pixel values are not 255, it is inside the hole. If the pixel values are 255, it is outside the hole. If the pixel is inside the hole, the adjusting factor one is used during the Euclidean error distance computation; if it is outside the hole, the adjusting factor *k* is used. The Euclidean error distance computation between the two vectors is according to the *Errordistance* equation. Figure 6 is an example of the coarse-to-fine information flow with mask pyramid level.

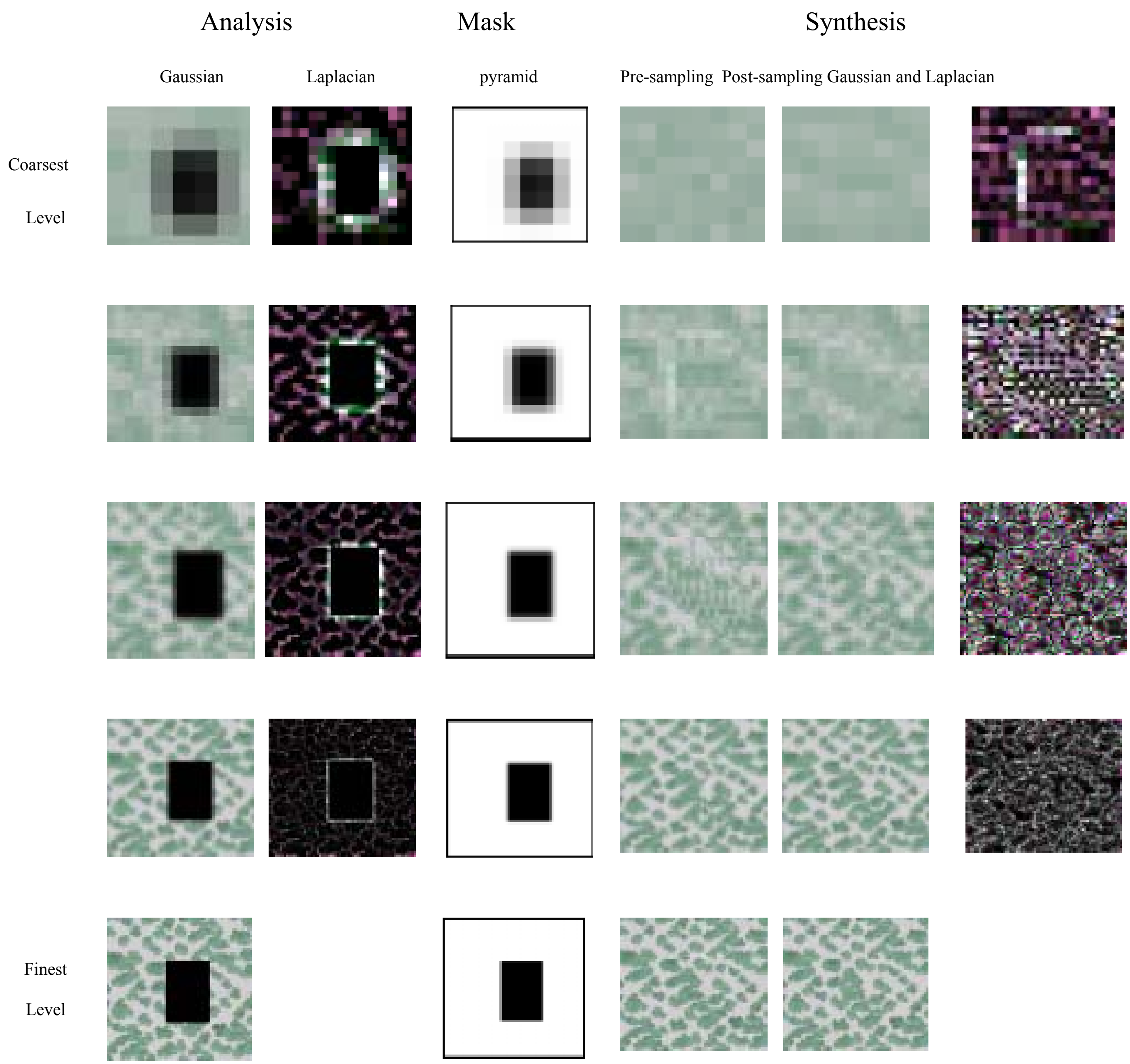


Figure 6: Coarse-to-fine information flow with mask pyramid

4.2.5 Algorithms

All algorithms' descriptions of the above texture synthesis technique to inpainting are exhibited as follows:

Readppm(): An algorithm that reads a PPM ASCII image information and stores pixel values into a data file without the PPM head structure.

Makeoriginalmask(): An algorithm that calculates the original mask image according to the input image data file information.

Adjust(): An algorithm that adjusts the original input texture according to the image preprocessing requirements.

Reduce(): An algorithm that accomplishes the Laplacian pyramid transform level by level by acting on the input texture data file and the original mask file with Gaussian convolution.

Project(): An algorithm that executes upsampling and Gaussian convolution for the given Gaussian pyramid level image.

Laplacian(): An algorithm that does transaction for generating Laplacian pyramid level image.

DoConvolution(): A function that calculates the Gaussian convolution for given Gaussian kernel.

Swap_indexes() *and Random_perm*(): Functions that generate the random order synthesis pixels.

Cminimum(): A function that finds the minimum number of a pixel in the given analysis Gaussian pyramid level has been selected and used in the *post-sampling* texture.

ErrorDistance(): A function that computes the error distance of the two neighborhood vectors with the neighborhood weighting and adjusting factors.

NeighborFind(): A functions that finds the neighborhood pixels of the analysis side.

Sampling(): This is the main procedure of the texture synthesis algorithm, which finds the optimal pixels for filling in the hole.

The formalizations of texture synthesis and sampling algorithms are shown in the following tables: Table 2 and Table 3. In our implementation, the whole texture synthesis algorithm has been divided into five (for 128×128 image) or six (for 256×256 image) or seven (for 512×512 image) individual steps. With the increasing of the input texture size, the later steps will take more time to finish sampling process. Among all the synthesis steps, the first step is to do synthesis with the coarsest level texture, and randomly choosing pixels from the coarsest Gaussian level texture to fill in the initial *pre-sampling* level texture. The sampling process uses the coarsest Gaussian level as analysis input texture and the outputs are the *post-sampling* Gaussian and Laplacian level textures. The last step is to finish the final synthesis process, which uses the original input texture as the analysis input and the previous step output as the *pre-sampling* texture, there is no Laplacian level image in this step. The inputs of the intermediate steps are the corresponding Gaussian and Laplacian level image on analysis side and the *pre-sampling* level texture from the previous step.

Texture synthesis algorithm pseudocode summary.

TextureSynthesis(I_0)

Begin

$Level \leftarrow log_2(size(I_0))$

$G_{M0} \leftarrow MakeOriginalMask(I_0)$

For i = 0 to *Level* do

$\langle G_{Ai}, L_{Ai} \rangle \leftarrow LaplacianPyramidTransform(I_0)$

$G_{Mi} \leftarrow LaplacianPyramidTransform(G_{M0})$

End

$G_{\mathrm{Glevel}} \leftarrow Random(G_{Alevel})$

For j = *Level* to 0 do

$\langle G_{Sj}, L_{Sj} \rangle \leftarrow Sampling(G_{Aj}, L_{Aj-1}, G_{\mathrm{Gj}}, G_{Mj})$

$G_{Gj-1} \leftarrow Project(G_{Sj}) + L_{Sj-1}$

End

return(G_{S0})

End

Table 2: Texture synthesis algorithm

Sampling algorithm pseudocode summary

Sampling(G_A, L_A, G_{Gk}, G_M)

Begin

Cmin $\leftarrow$ MAX_INT

MinimumDistance $\leftarrow$ MAX_FLOAT

Sampling_num $\leftarrow$ 0

While (*Sampling_num* < 4) do

For $i = 1$ to size(G_A) do

Cmin $\leftarrow$ *Cminimum*(*c[]*)

RandomRow = *Random*(Height(G_A))

RandomCol = *Random*(Width(G_A))

Ω_S $\leftarrow$ *NeighborFind*(G_{Gk}, *RandomRow, RandomCol*)

For j=1 to *size*(G_A) do

If j is not in hole determine by using G_M Then

Ω_A $\leftarrow$ *NeighborFind*(G_A, j)

$d \leftarrow \mathrm{ErrorDistance}(\Omega_A, \Omega_S)1.005^{(c[j]-Cmin)}$

If d < *MinimumDistance* Then

Optimal $\leftarrow j$

MinimumDistance $\leftarrow d$

End

c[*Optimal*] $\leftarrow$ c[*Optimal*] + 1

End

G_S (*RandomRow, RandomCol*) $\leftarrow$ G_A(*Optimal*)

$L_S(RandomRow*2, RandomCol*2) \leftarrow L_A(Optimal*2)$

End

End

End While

$G_G \leftarrow Project(G_S) + L_S$

Return G_G

End

Table 3: Sampling algorithm

Notations:

Symbol	Meaning
I_0	original input texture sample
G_A	analysis Gaussian pyramid level
L_A	analysis Laplacian pyramid level
G_G	*pre-sampling* synthesis pyramid level
G_M	mask pyramid level
Ω_S	synthesis neighborhood vector
Ω_A	analysis neighborhood vector
G_S	*post-sampling* Gaussian pyramid level
L_S	*post-sampling* Laplacian pyramid level
c[]	array for storing the number of pixels have been selected
RandomRow	the row number of the synthesis pixel
RandomCol	the column number of the synthesis pixel
Optimal	the best match pixel position
d	the Euclidean distance of the two neighborhood vectors
size()	giving the total number of pixels of an image
Cmin	the minimum number of times of a pixel has been selected

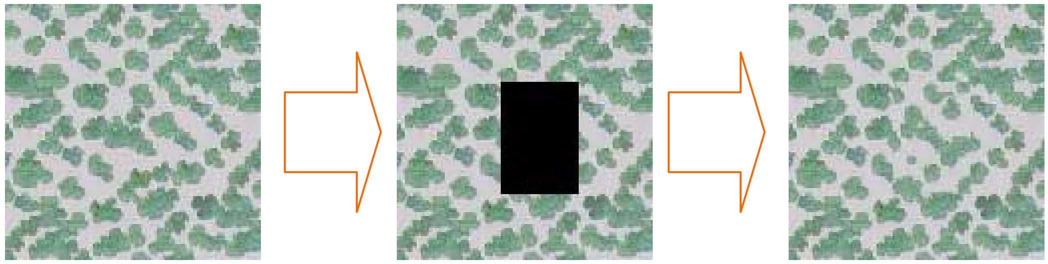
Chapter 5:

Results

All of the results shown in this chapter use four pyramid levels of resolution for 128×128 samples, five levels of resolution for 256×256 samples. The 7×7 neighborhoods are used in the sampling process at all levels. Some of the samples are provided by Joel Castellanos [11]. Some of the samples are downloaded from NASA's web site [13]. Some of them are taken from Peterson Multimedia Guides: North American Birds. Here I want to thank all of the contributors for distributing the images that help me a lot.

In our implementations, some holes are rectangular shapes that were taken on purpose. Others are made according the natural objects geometry shape. Rectangular holes are more difficult to do inpainting than the natural geometry holes because they often generate seams on the boundary of the holes.

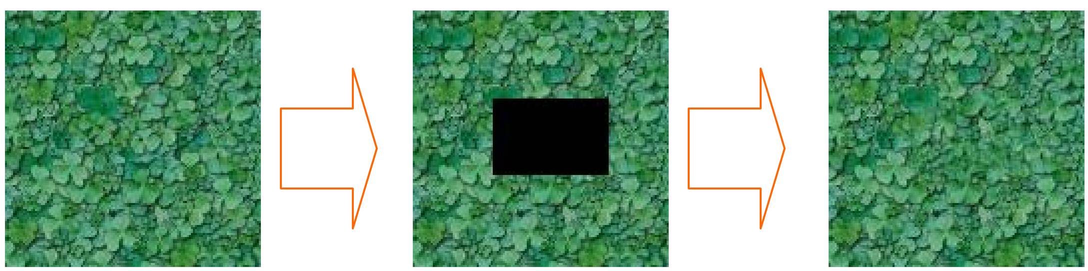
The sample presented in Figure 7 is a clover texture that has been purposely made a rectangular hole, then the sampling process is used to fill in the hole with surrounding texture information.



(a) (b) (c)

Figure 7: (a) Original sample texture. (b) Sample texture with hole. (c) Our inpainting algorithm result.

Figure 8 is another clover texture with a rectangular shape hole and has been synthesized by using surrounding pixels to fill in the hole.

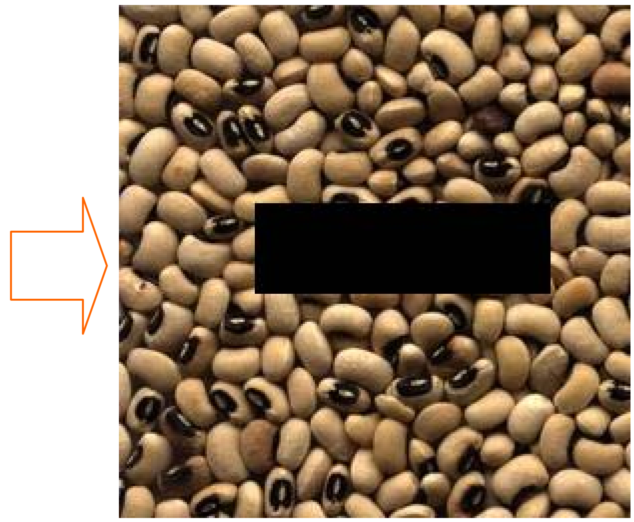


(a) (b) (c)

Figure 8: (a) Original sample texture. (b) Sample texture with hole. (c) Our inpainting algorithm result.

Figure 9 is a 256×256 texture of randomly arrangement beans, which is isotropic at all scales. The hole is also a rectangular shape.

(a)

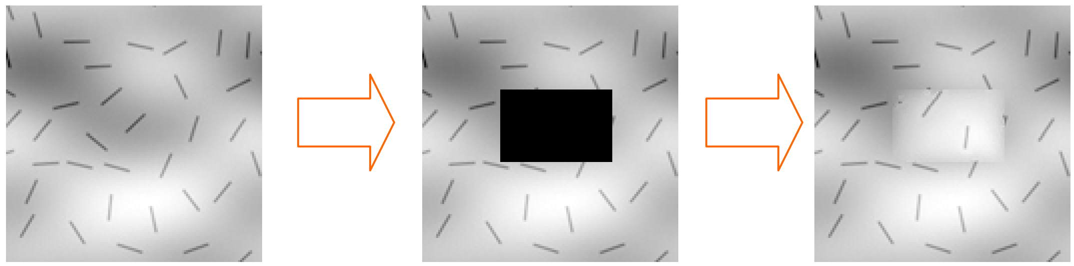


(b)

(c)

Figure 9: (a) Original sample texture. (b) Sample texture with hole. (c) Our inpainting algorithm result.

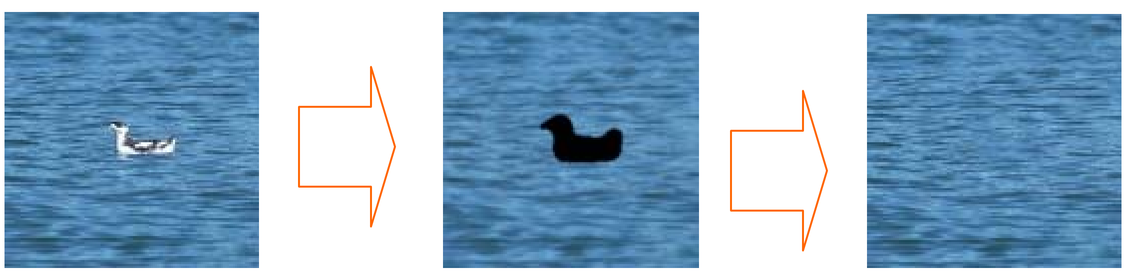
The bar sample shown in Figure 10 is not good. In general, our method fails with such textures. The algorithm has been trying to closely match the original, however the seam on the boundary is clearly appearing. What we understand is that the sampling method is too greedy and results in loosing the first-order statistics. For the boundary, we may need other strategies to handle it. This will be a part of our future work.



(a) (b) (c)

Figure 10: (a) The original image. (b) The image with a hole. (c) Our synthesized result.

For the above results, the holes are rectangular shapes that can be made arbitrary inside the image region. Followings are some examples that apply the algorithm to natural holes for inpainting.

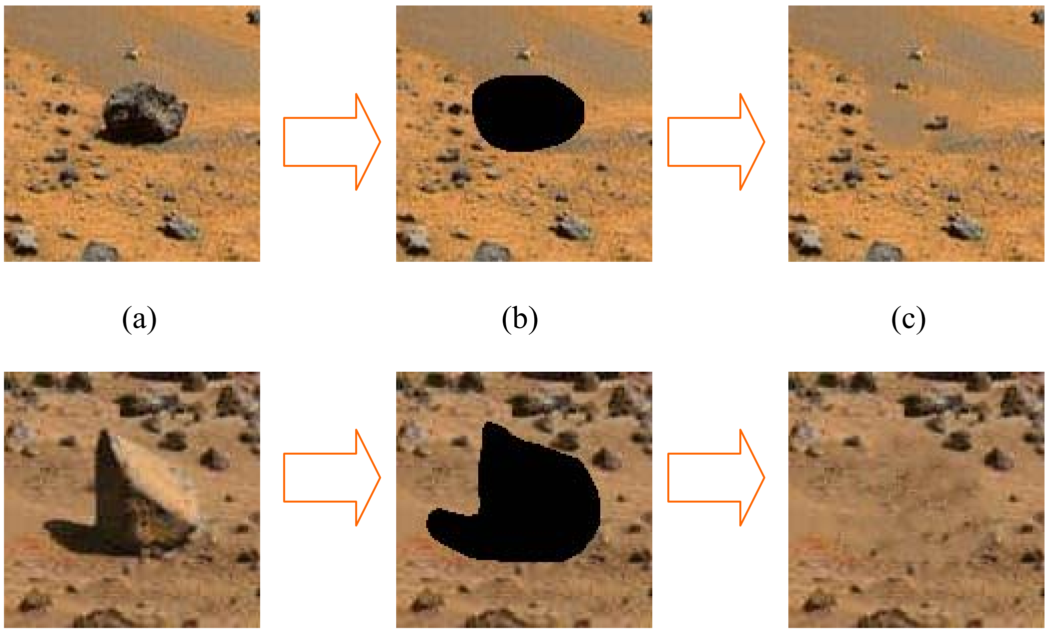


(a) (b) (c)

(d) (e) (f)

Figure 11: (a) and (d) are the original textures. (b) and (e) are the hole textures. (c) and (f) are the inpainting results.

There are some sample images downloaded from NASA's web site, which are showing the surface of Mars. We performed the inapinting algorithm on these images for removing some objects such as rocks. The performance is good.



(a) (b) (c)

(d) (e) (f)

Figure 12

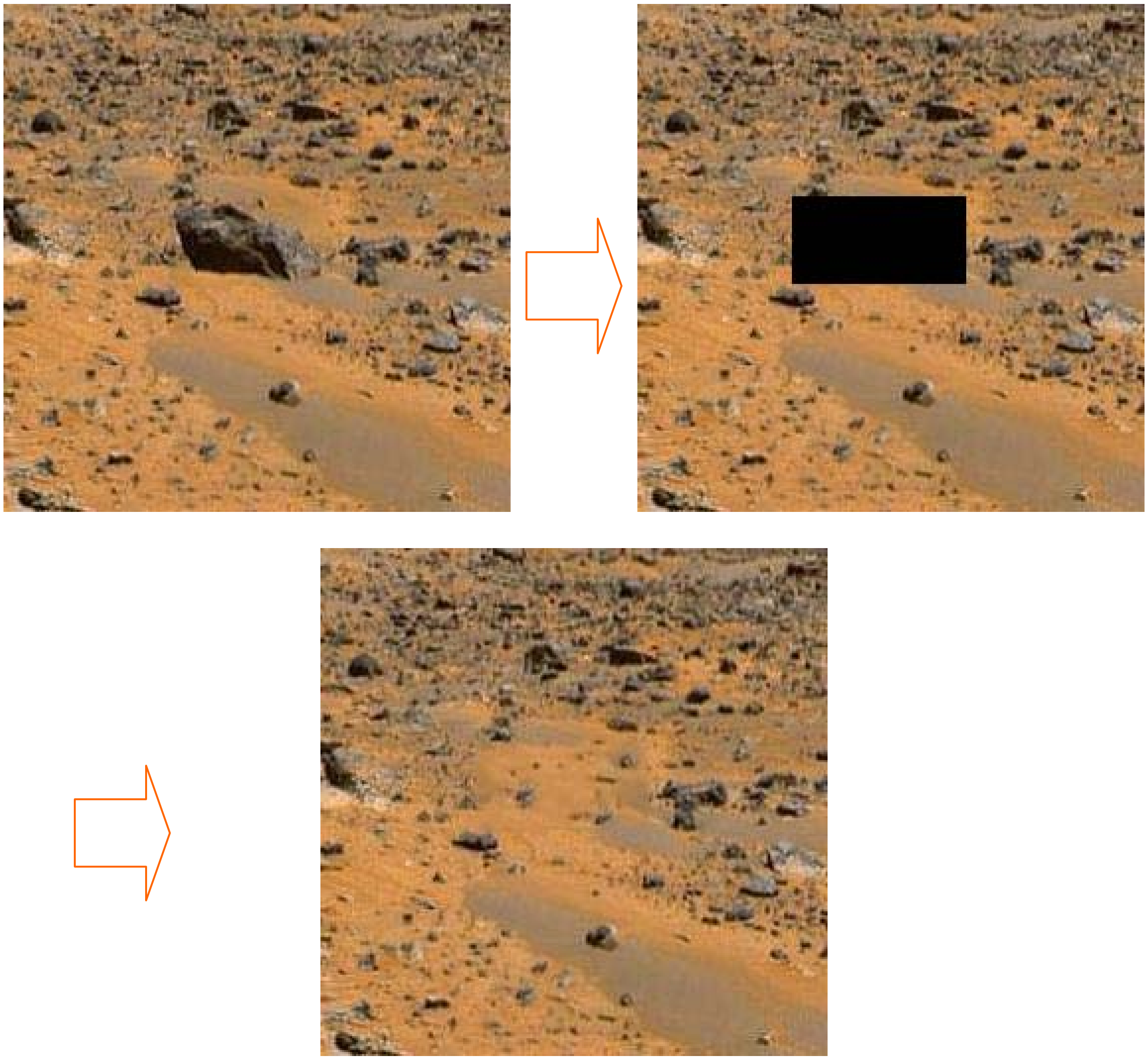


Figure 13

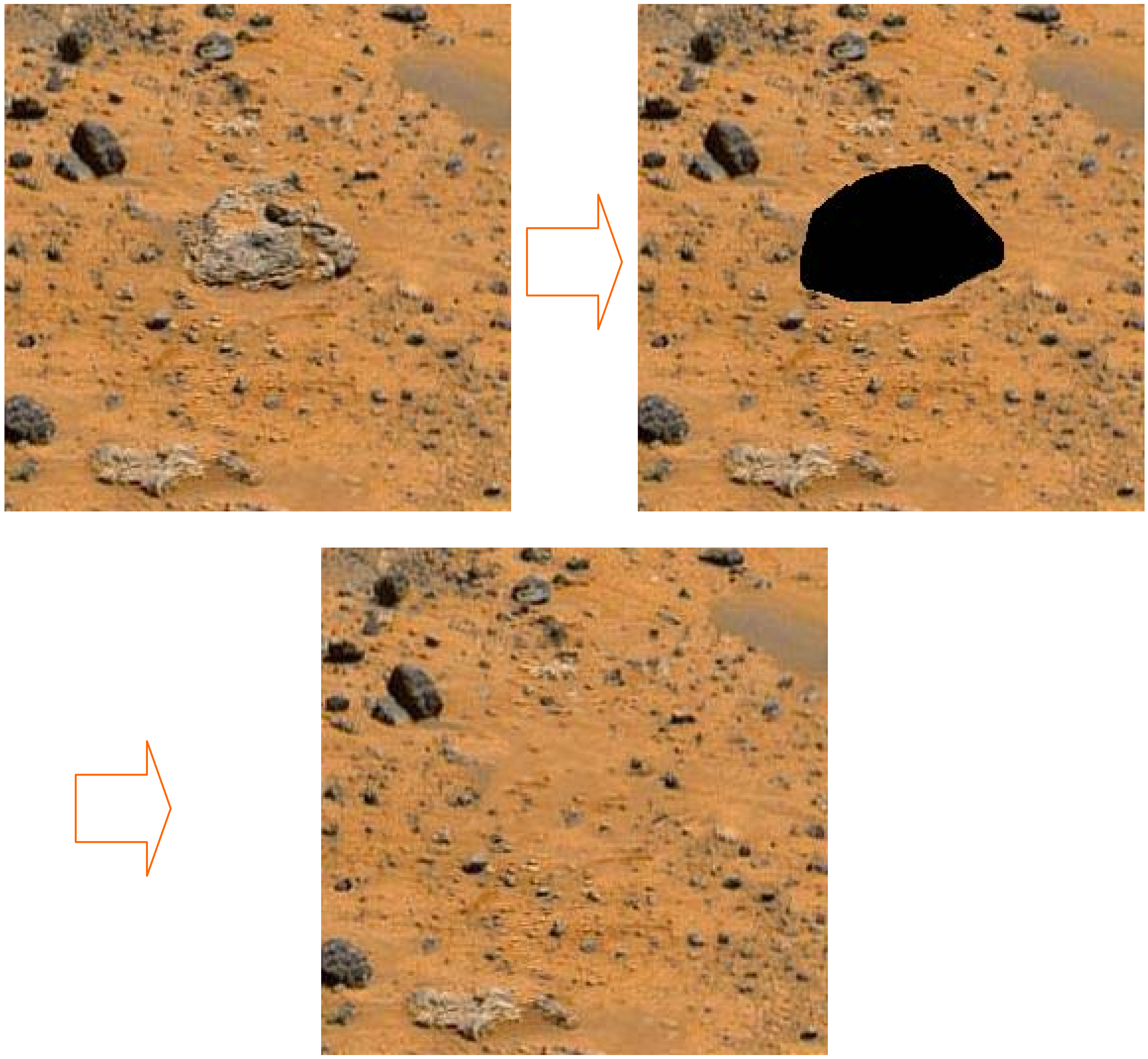
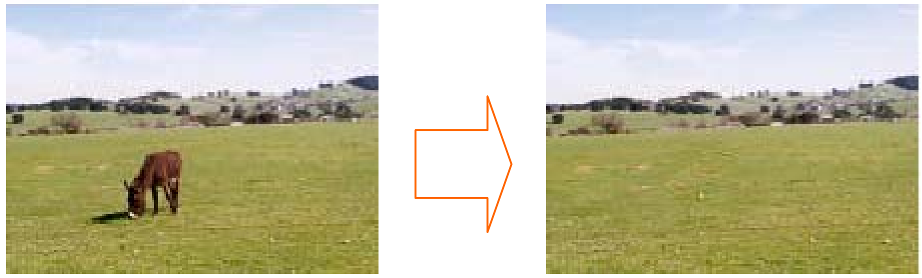


Figure 14

There is another example that compares Wei's result [12] with our result in Figure 13. For this example, the difficulty is that the original texture contains different texture region, however, our approach still achieves a very good result.



(a) Original sample image (b) Li-Yi Wei's result

(c) Our hole image (d) Our inpainting result

Figure 15

Except above experimental results, following is an example that the hole spanning two different texture regions, and the hole is filled in by just one part of the texture region through our algorithm. For this kind of situation, texture segmentation is needed for inpainting through our algorithm.

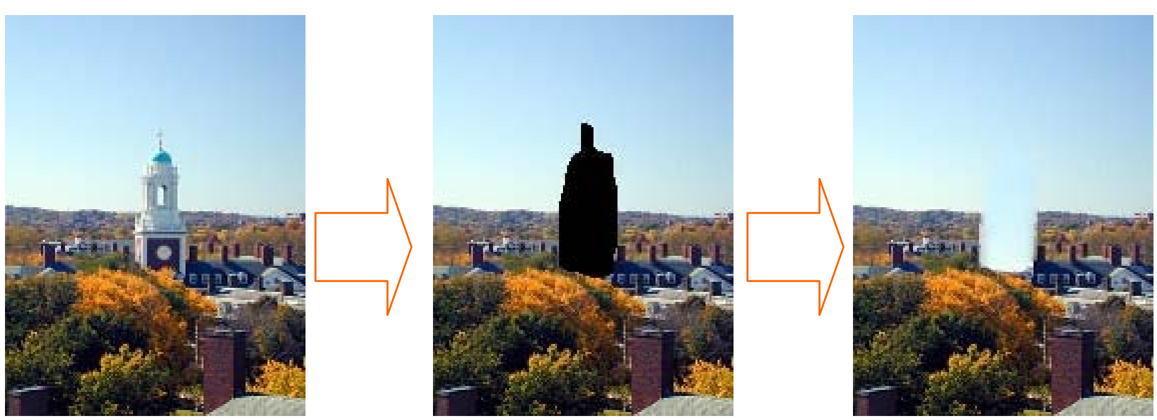


Figure 16: Hole spanning two different textures example.

For the result showed in Figure 10, we know that it is not good. In order to have a good understanding what reason results into the negative inpainting result. We separate the high and low frequency of the texture, so we have two different textures bar and cloud. Through experiments, we find that the low frequency contained in the original image contributes more on the negative result.

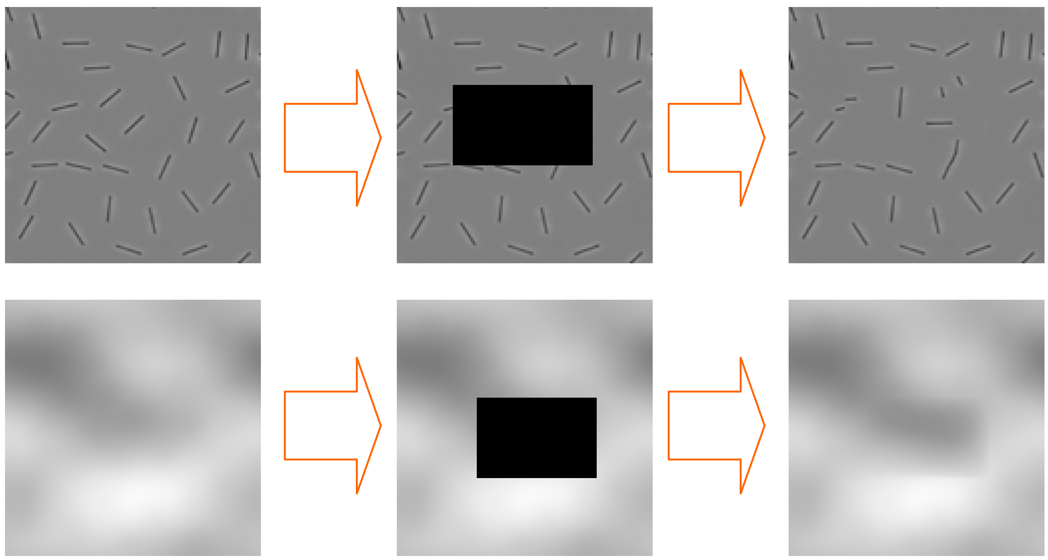


Figure 17: Barcloud low and high frequency inpainting comparison.

Chapter 6:

Complexity Analysis

For the performance, the most computationally work of the algorithm is the sampling procedure. We usually use a neighborhood size 3×3 for the coarsest level sampling procedure, and 7×7 for the rest of the levels sampling procedure. The Laplacian pyramid transform transforms any texture to the coarsest level which is 8×8 size image. So for a 128×128 sample texture, there will have four levels of Laplacian pyramid, five levels of Laplacian pyramid for a 256×256 sample texture, and six levels of Laplacian pyramid for a 512×512 sample texture, and so on. During our experiments, we have separate each level as an individual process. For different levels, the time complexity heavily depends on the texture size in that level. Also, we know for inpainting the hole or real object size will determine the pixels that are synthesized by using sampling procedure. So the hole or object size is the realistic reason that determines the time complexity. However, through our experiments, we find that for the same level, even for different textures, the running time approximately tends to the same value. In order to have an precision demonstration of the algorithm complexity, we take all the running examples and each level running time. The coarsest level running time is approximately 2 seconds, the level adjacent to the coarsest level running time is 3 seconds. Table 4 shows a detailed explanation for each level running time with a 128×128 sample texture. Table 5 shows the running time for 256×256 sample textures.

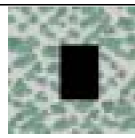
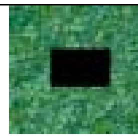
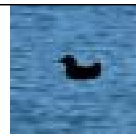
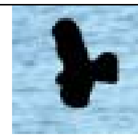
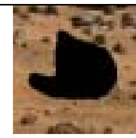
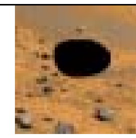
Levels						
4	2s	2s	1s	3s	3s	2s
3	2s	2s	2s	3s	3s	2s
2	18s	19s	16s	18s	22s	18s
1	328s	342s	254s	328s	381s	299s
0	5450s	5968s	4121s	4260s	7117s	4762s

Table 4: 128×128 sample textures running time illustration.

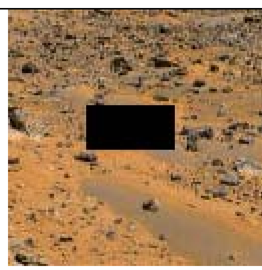
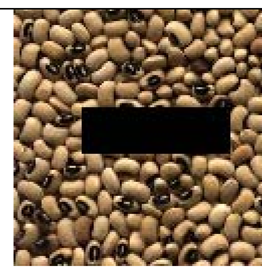
Levels			
5	2s	3s	4s
4	2s	3s	5s
3	18s	18s	22s
2	328s	295s	437s
1	6957s	4762s	10084s
0	56910s	23901s	79581s

Table 5: 256×256 sample texture running time illustration.

All of the results experiments above is implemented in Visual C++6.0, running on a 2.0GHz Intel Celeron CPU with 512MB RAM, and usually the CPU usage is about 95~96%.

Chapter 7:

Main problems and Future Work

There are many different techniques that use texture synthesis to do image inpainting for computer-based image process. In our implementation, the most difficult thing is how to deal with the seams. We have found that some inpainting results are good for some samples, and some results are acceptable. However, there still have some textures are very difficult for inpainting by using this non-parametric texture synthesis method. For example, a texture that includes different texture regions and the hole spans different regions. By using the *k* factor to adjust the neighborhood distance and sampling more than one time have been proven to be a strong improvement on the inpainting performance. But the seam is remaining as the biggest problem.

For our experiments, we mostly chose the homogeneous texture as our input image. From above examples, we can see that many textures are homogeneous because we firstly think this algorithm is not suitable for non-homogeneous texture. However, we finally found this technique also does a great job on some non-homogeneous texture images as long as the hole locates in one homogeneous region. This has been proven in Figure 12. But for a general non-homogeneous texture image and the hole spans among different homogeneous region remaining as an unsolved problem for our technique.

There is another problem related to the time complexity. Our algorithm is deeply time consuming with the increasing of the image size. For our practice, a 256X256 texture will take more than one day to finish the final step synthesis on a 2.0GHz Celeron Pentium IV with 512 MB RAM.

Our future work will focus on how to solve the seam problem and perform inpainting technique to general non-homogeneous texture. The goal is to produce a seamless texture synthesis technique to inpainting that works on general texture image. For this reason, texture segmentation will be a part of the work. Also, speeding the algorithm is another important part of the future work.

Chapter 8:

Conclusion

Texture is an important attribute of images. Texture synthesis has many applications in image processing. Inpainting is an old and hot topic in image restoration or removing unwanted objects from an image. The technique we demonstrated in this thesis is an automatic non-parametric texture synthesis algorithm applying to image inpainting. Comparing with other methods of inpainting, this technique works at a flexible position. It could be applied to some situation that requires flexible and good performance for small still texture inpainting.

From Efros and Leung, Wei and Levoy to Castellanos-Williams, they all have influence on this thesis, although this work directly stands up on the Castellanos-Williams idea for texture synthesis.

The extension of the application for this method is over more general categorized texture image. Surely there will have more challenge on the future work.

REFERENCES

[1] http://www.hyperdictionary.com/dictionary/texture

[2] Alexei A. Efros and Thomas K. Leung. Texture Synthesis by Non-parametric Sampling. *IEEE International Conference on Computer Vision*, Corf, Greece, September 1999.

[3] Li-Yi Wei and Marc Levoy. Fast Texture Synthesis using Tree-structured Vector Quantization. *In SIGGRAPH* 2000, 479-488.

[4] Li-Yi Wei and Marc Levoy. Texture Synthesis Over Arbitrary Manifold Surfaces. *In SIGGRAPH* 2001, 355-360.

[5] David J. Heeger and James R. Bergen, Pyramid-Based Texture Analysis/Synthesis. *In SIGGRAPH '95*, pages 229-238, 1995.

[6] Paul Billault. Texture Synthesis Algorithms. *Ecole Polytechnique, Promotion X-98*.

[7] Hamilton Chong. Image Inpainting and Texture Synthesis. Appears in: *http://www.people.fas.harvard.edu/~hchong/Spring2002/cs276r/*

[8] A.C. Kokaram, R.D. Morris, W.J. Fitzgerald, P.J.W. Rayner. Interpolation of missing data in image sequences. *IEEE Transactions on Image Processing* 11(4), 1509-1519, 1995.

[9] A. Hirani and T.Totsuka. Combing Frequency and spatial domain information for fast interactive image noise removal. *Computer Graphics*, pp.269-276, *SIGGRAPH* 96, 1996.

[10] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester. Image Inpainting. *Computer Graphics, SIGGRAPH* 2000, July 2000, 417-424.

[11] Joel Castellanos. Automatic Synthesis of Isotropic Textures on Surface from Sample Images. Master Thesis, University of New Mexico, Albuquerque, 2003.

[12] http://mrl.nyu.edu/~hertzman/topics00/MLCG2.pdf

[13] http://marsprogram.jpl.nasa.gov/