Fast Algorithms For Fragment Based Completion In Images Of Natural Scenes

2004

Siddharth Rajkumar Borikar

University of Central Florida

Find similar works at: https://stars.library.ucf.edu/etd

University of Central Florida Libraries http://library.ucf.edu

Part of the Computer Sciences Commons, and the Engineering Commons

STARS Citation

Borikar, Siddharth Rajkumar, "Fast Algorithms For Fragment Based Completion In Images Of Natural Scenes" (2004). Electronic Theses and Dissertations. 82.
https://stars.library.ucf.edu/etd/82

This Masters Thesis (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of STARS. For more information, please contact lee.dotson@ucf.edu.
FAST ALGORITHMS FOR FRAGMENT BASED COMPLETION IN IMAGES OF NATURAL SCENES

by

SIDDHARTH RAJKUMAR BORIKAR
B.E. University of Mumbai, 2000

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in the School of Computer Science in the College of Engineering and Computer Science at the University of Central Florida Orlando, Florida

Spring Term
2004

Major Advisor: Dr. Sumanta Pattanaik
ABSTRACT

Textures are used widely in computer graphics to represent fine visual details and produce realistic looking images. Often it is necessary to remove some foreground object from the scene. Removal of the portion creates one or more holes in the texture image. These holes need to be filled to complete the image. Various methods like clone brush strokes and compositing processes are used to carry out this completion. User skill is required in such methods. Texture synthesis can also be used to complete regions where the texture is stationary or structured. Reconstructing methods can be used to fill in large-scale missing regions by interpolation. Inpainting is suitable for relatively small, smooth and non-textured regions. A number of other approaches focus on the edge and contour completion aspect of the problem. In this thesis we present a novel approach for addressing this image completion problem.

Our approach focuses on image based completion, with no knowledge of the underlying scene. In natural images there is a strong horizontal orientation of texture/color distribution. We exploit this fact in our proposed algorithm to fill in missing regions from natural images. We follow the principle of figural familiarity and use the image as our training set to complete the image.
To my parents
ACKNOWLEDGEMENTS

I realize dearth of words when I express my sincere gratitude to Dr. Sumanta Pattanaik. His guidance and support, both academically and personally, has been pivotal to my career. I am very grateful to him for providing me the opportunity to pursue this work.

Very special thanks to Dr. Kanad Biswas for guiding me through my entire thesis work as my co-advisor. He has been a source of incessant inspiration; I wish to thank him from the bottom of my heart for all he has done for me.

I would like to thank my committee member Dr. Ratan Guha for his time to read my thesis and invaluable suggestions.

A person who deserves a special mention is Mangesh Nijasure who has been a great buddy and a mentor as and when required. Sincere thanks to all my friends especially Darshan, Gautami, Suyog, Kartik, Prasad and Amit for being there when I needed them the most.
TABLE OF CONTENTS

LIST OF FIGURES vii

CHAPTER ONE: INTRODUCTION 1

CHAPTER TWO: RELATED WORK 7

CHAPTER THREE: ALGORITHM (PIXEL BASED APPROACH) 26

  Algorithm 1.1 27
  Algorithm 1.2 30
  Algorithm 1.3 31
  Algorithm 1.4: Multi Scale Completion 33
  Algorithm 1.5 35
  Algorithm 1.6: Linear Interpolation Based Completion 36
  Algorithm 1.7: User Defined Placement (Interactive) 37
  Algorithm 1.7 (a) 38

CHAPTER FOUR: ALGORITHM II (FRAGMENT BASED APPROACH) 44

  Algorithm 2.1: Grid Algorithm 44
  Correction for Self-Replacement 47
  Correction for Center line and Boundary Irregularities 49
  Tilted Orientations 50

CHAPTER FIVE: CONCLUSIONS 62

  Computations in [Drori03] 62
  Computations in our method 65
  Timing Estimates 68
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Texture Mapping</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>Texture Synthesis</td>
<td>2</td>
</tr>
<tr>
<td>1.3</td>
<td>Texture cloning</td>
<td>3</td>
</tr>
<tr>
<td>1.4</td>
<td>Image inpainting</td>
<td>3</td>
</tr>
<tr>
<td>1.5</td>
<td>Image Completion</td>
<td>5</td>
</tr>
<tr>
<td>1.6</td>
<td>Images with their Fourier transforms</td>
<td>6</td>
</tr>
<tr>
<td>2.1</td>
<td>Image inpainting</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>Input image and the result after texture warping.</td>
<td>10</td>
</tr>
<tr>
<td>2.3</td>
<td>Super Resolution</td>
<td>11</td>
</tr>
<tr>
<td>2.4</td>
<td>Result of texture synthesis.</td>
<td>13</td>
</tr>
<tr>
<td>2.5</td>
<td>Failures of [Heeger95] – Textures of hay and marble</td>
<td>13</td>
</tr>
<tr>
<td>2.6</td>
<td>Image Replacement</td>
<td>15</td>
</tr>
<tr>
<td>2.7</td>
<td>[Efros99] Method</td>
<td>16</td>
</tr>
<tr>
<td>2.8</td>
<td>Sample image</td>
<td>17</td>
</tr>
<tr>
<td>2.9</td>
<td>Other textures.</td>
<td>17</td>
</tr>
<tr>
<td>2.10</td>
<td>Input texture sample and partially filled noise image</td>
<td>18</td>
</tr>
<tr>
<td>2.11</td>
<td>Good and bad results of Wei-Levoy algorithm</td>
<td>20</td>
</tr>
<tr>
<td>2.12</td>
<td>Synthesis result of Ashikhmin algorithm.</td>
<td>20</td>
</tr>
<tr>
<td>2.13</td>
<td>Filling in the missing regions with the texture</td>
<td>22</td>
</tr>
<tr>
<td>2.14</td>
<td>Image Completion [Drori03].</td>
<td>24</td>
</tr>
<tr>
<td>3.1</td>
<td>Image $I$</td>
<td>27</td>
</tr>
<tr>
<td>3.2</td>
<td>Unknown matte $\Omega$</td>
<td>27</td>
</tr>
</tbody>
</table>
CHAPTER ONE:
INTRODUCTION

The filling of missing information is a very important topic in image processing, with applications including image coding and wireless image transmission (e.g. recovering lost blocks), special effects (e.g. removal of objects), and image restoration (e.g. scratch removal). The object of inpainting is to rebuild the missing or damaged portion of an image in order to make it more legible and to restore its unity. The removal of portions of an image is an important tool in photo-editing and film post-production. In image processing textures play a very important role because one can control the texture even if there is no control over the structures in the image.

Figure 1.1 Texture Mapping

Textures are inherent part of any image and are used widely in computer graphics to represent fine visual detail and produce more realistic looking images. Texture mapping is a powerful

1 http://astronomy.swin.edu.au/~pbourke/texture/texturemapping/
technique to add realism to computer generated scene. In its basic form, texture mapping lays an image (the texture) onto an object in a scene. Texture synthesis attempts to generate large textures based on small texture samples, small textures are processed to get larger textures with similar visual appearance. Efficient texture synthesis generates textures which can be used for any surface on any object. Texture editing allows manipulating the existing texture to add more effect to it as per its use. Texture synthesis can also be used for completing missing regions in the image. Potential benefits of texture synthesis methods include the ability to create large and/or tilable textures from a small sample in a direct manner. Some methods require user input to select the texture to indicate large-scale properties of the texture appearances. There are editing tools available to create textures which require a sufficient degree of sophistication from the user. Attempts have been made in texture synthesis to generate the textures automatically without any user interaction.

Figure 1.2 Texture Synthesis [Wei00]
Other applications of texture synthesis include occlusion fill-in, image denoising, foreground removal and compression. Texture editing operations also have been automated to some extent, requiring very less user input and skills. An example of texture cloning is shown in Figure 1.3.

Figure 1.3 Texture cloning [Brooks02]

Texture synthesis methods have been applied in inpainting. Image inpainting is a technique in which the image is modified or filled in an undetectable manner. Inpainting is used to restore old paintings by filling in the gaps or cracks formed in it caused due to deterioration. This kind of filling in of missing information is an important topic in image processing with applications in image coding and wireless image transmission (e.g., recovering lost blocks), special effects (e.g., removal of objects).

Figure 1.4 Image inpainting. [Bertalmio00]
Recent work in this area tries to automate the filling of missing part using the available information from their surroundings. Work on digital inpainting has been done focusing on applications like restoration of films and texture synthesis. Image completion is another area related to texture synthesis. Inpainting techniques were naturally extended from paintings to images and films. Image inpainting and image completion are not the same; they differ in the area that is to be filled or completed. In image completion regions are large and consist of textures, large scale structures and smooth areas. In image completion the region that is to be removed is specified by the user. It is generally some foreground element that is not required in the image. After removing the foreground element, the area is to be filled so that the image looks naturally complete.

Removal of portions of an image is an important tool in photo editing. Various methods like clone brush strokes and compositing processes are used to complete the image, user skill is required in such methods. Texture synthesis can also be used to complete regions where the texture is stationary or structured. Reconstructing methods can be used to fill in large-scale missing regions by interpolation. Inpainting is suitable for relatively small, smooth and non-textured regions. There are a number of approaches related to image completion; most of them focus on the edge and contour completion aspect of the problem.

Our approach focuses on image based completion, with no knowledge of the underlying scene. In natural images there is a strong horizontal orientation of texture/color distribution. We have tried to exploit this fact in our proposed algorithm to fill in missing regions from a natural image.
We follow the principle of figural familiarity and use the image as our training set and complete the image based on example.

**Figure 1.5 Image Completion [Drori03]**

The assumption of horizontal orientation in natural images is also supported by taking the Fourier transform of such images. The result of Fourier transforms of some of the images is shown in Figure 1.6. As can be observed from the transforms, there is a distinct vertical line at the center. This indicates that the color/texture in the image is horizontally oriented.

The organization of the thesis is as follows. In chapter 2, we discuss some of the work related to the problem of image completion. In chapters 3 and 4, we present our algorithms on image completion. In chapter 5, we make a computational comparison of our algorithm with [Drori03].
Figure 1.6 Images with their Fourier transforms [freefoto]
A major goal of computer graphics is reproducing the visual realism of the real world and hence textures are commonly employed when rendering synthetic images. Textures on the surfaces of objects make the computer renderings of objects more realistic. Textures describe a wide variety of surface characteristics such as terrain, plants, fur and skin. These textures can be obtained from a variety of sources such as hand-drawn pictures or scanned photographs.

Textures have often been classified into two categories, deterministic textures and stochastic textures. A deterministic texture is characterized by a set of primitives and a placement rule (e.g., a tile floor). A stochastic texture, on the other hand, does not have easily identifiable primitives (e.g., granite, bark, sand). Many real-world textures have some mixture of these two characteristics (e.g. woven fabric, wood grain, plowed fields).

Texture mapping is a technique for adding the appearance of surface detail by wrapping or projecting a digitized texture image onto a surface. Problems arise in texture mapping if the there is a difference in size of the surface and texture or there is no direct mapping between the texture and the topology of the surface so the texture may be unnaturally mapped or distorted.

A solution to the problems in texture mapping is to generate synthetic textures. Because synthetic textures can be made any size, visual repetition is avoided. Texture synthesis can also
produce tileable images by properly handling the boundary conditions. Potential applications of texture synthesis are also broad; some examples are image denoising, occlusion fill-in and compression (as only the sample texture needs to be preserved). The goal of the texture synthesis is to create other sample textures which appear to be generated by the same underlying stochastic process from a texture which is defined as some visual pattern with a stationary distribution on a 2-D plane.

Image inpainting is a method for repairing damaged pictures or removing unnecessary elements from pictures. Inpainting has its applications in restoration of damaged paintings and photographs and also could be applied in image filling problems. The objective of inpainting is to reconstitute the missing or damaged portions of the work, in order to make it more legible and to restore its unity. Inpainting has found its use in photography and film.

Figure 2.1 Image inpainting. [Bertalmio00]

The purposes remain the same: to revert deterioration (e.g., cracks in photographs or scratches and dust spots in film), or to add or remove elements (e.g., removal of stamped date and red-eye from photographs).
Bertalmio et al. [Bertalmio00] have tried to devise an algorithm for the basic techniques used by professional restorators in image restoration. The basic idea is to smoothly propagate information from the surrounding areas in the isophotes (lines of equal gray values) direction. The user has to provide the region to be inpainted.

The global picture determines how to fill in the gap, the purpose of inpainting being to restore the unity of the work. The structure of the area surrounding is continued into the gap, contour lines are drawn via the prolongation of those arriving at. The different regions inside, as defined by the contour lines, are filled with color, matching those of regions at the boundary. The small details are painted in other words, “texture” is added.

The authors have represented the inpainting concepts in mathematical and algorithmic language and presented a method for digital inpainting with least user input. The user only has to mark the regions that are to be painted. By using multi resolution techniques the algorithm saves time.

Brooks et al. [Brooks02] have presented a mechanism to edit textures. It allows the user to perform replicated texture editing operations. A single editing operation at a given location would cause global changes, affecting all similar areas of the texture.

The editing system used neighborhood self-similarity within a texture to edit the original texture itself. Changes made to a particular pixel by the user were made to affect all pixels that exhibit similar local neighborhoods at multiple scales. Self-similarity based painting altered the color or brightness of similar pixels to that which the user selected. A pixel is selected, referred as
selection point, which acted as a base for the replication. The local circular neighborhood of the chosen selection point was then compared against that of every other pixel’s neighborhood in the image. The current painting color was then applied to the selected pixel but also to a subset of all pixels in the image: those that have local neighborhoods whose difference from the selected pixel are within a certain threshold.

The algorithm changes (edits or warps) the texture as defined by the user. The pixel values are changed as per the user input and the result is an automatically generated texture with limited user input.

![Figure 2.2 Input image and the result after texture warping. [Brooks02]](image)

Freeman et al. [Freeman02] have discussed a method to increase the resolution of images. Constructing polygon models for complex, real-world objects is difficult. Image based rendering (IBR), a complementary approach for representing and rendering objects, uses cameras to obtain rich models directly from real-world data. When the image is enlarged, since the original information captured is less, one gets a blurry result. The authors have developed a training set for a learning-based approach for enlarging images. In a training set, the algorithm learns the fine
details that correspond to different image regions seen at a low-resolution and then uses those learned relationships to predict fine details in other images.

Training set based methods could be studied for its applications in texture synthesis and image filling. Texture synthesis could generate large training sets but the number of textures to be considered could be limited as per the need. The same would apply for image filling.

Figure 2.3 Super Resolution [Freeman02]

Heeger et al. [Heeger95] have described a method for synthesizing images that match the texture appearance of a given digitized sample. The focus of their paper is on the synthesis of stochastic textures. Some theories of texture discrimination are based on the fact that two textures are often difficult to discriminate when they produce a similar distribution of responses for set of (orientation and spatial-frequency selective) linear filters. In this method, the textures are synthesized by matching distributions (or histograms) of filter outputs. The approach is based on the principle that all of the spatial information characterizing a texture image can be captured in the first order statistics of an appropriately chosen set of linear filter outputs.
The inputs to the algorithm are texture image (digitized) and a noise image (typically uniform white noise). The noise is modified to look like the input texture. It is done by making use of an invertible image representation known as an image pyramid; along with a function to match the histograms of two images. The operations involved in the algorithm are - histogram matching, pyramid generation and texture matching.

Histogram matching is a generalization of histogram equalization. The input image is coerced via a pair of lookup tables to have a particular histogram. The two lookup tables are: (1) the cumulative distribution function (cdf) of one image, and (2) the inverse cumulative distribution function (inverse cdf) of the other image. An image’s histogram is computed by choosing a binsize, counting the number of pixels that fall into each bin, and dividing by the total number of pixels. An image’s cdf is computed from its histogram simply by accumulating successive bin counts. 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. These two lookup tables are used by the match-histogram function to modify an image (im1) to have the same histogram as another image (im2).

The input noise image is modified by the match-texture function so that it looks like an input texture image. The histogram of the noise image is matched to the input texture. Pyramids are constructed from both the (modified) noise and texture images. The histograms of each of the corresponding pyramid subbands are matched by looping through the two pyramid data structures. Finally, the (histogram-matched) noise pyramid is collapsed to generate a preliminary
version of the synthetic texture. Matching the histograms of the pyramid subbands modifies the histogram of the collapsed image. The steps, rematching the histograms of the images, and then rematching the histograms of the pyramid subbands, are iterated to get both the pixel and pyramid histograms to match.

The algorithm requires that the input texture be homogenous, if the repeated patterns are small enough the algorithm works well but in case of high frequency textures the algorithm fails to capture the correlation over large distances.

![Image](image1)

**Figure 2.4 Result of texture synthesis. [Heeger95]**

![Image](image2)

**Figure 2.5 Failures of [Heeger95] – Textures of hay and marble [Heeger95]**
The work done by Igehy et al. [Igehy97] is another approach for image filling which uses the texture synthesis algorithm described in the paper by Heeger et al. [Heeger95]. In the algorithm the noise is converted to an image using the original image and a synthetic texture which is derived from the target. The algorithm is completely based on [Heeger95], except that with every iteration of the algorithm, the original is composited back into the noise image according to the mask using a multi-resolution compositing technique that avoids blurring and aliasing. Matching is done on the newly composited noise image. The resulting image becomes synthetic texture at pixels where the mask is 1, and original texture at pixels where the mask is 0.

Using fractional values in the mask yields a smooth transition from real to synthetic texture from the point of view of texture discrimination. Since the non-masked regions of the original image are chosen to already match the target texture, the sub-band matching does not modify these regions too much. Thus, the mask values of a region determine how much of the original texture’s characteristics are incorporated into the sub-band matching, allowing for smooth texture transition.

In this algorithm the target texture in the image has to be specified. It has a limitation that the texture synthesized does not capture the correlation over large distances, so only textures with high frequency would give good results.
Efros et al. [Efros99] have modeled the texture as a Markov Random Field (MRF). 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. Based on this principle the neighborhood of the pixel is considered. The neighborhood of a pixel is modeled as a square window around that pixel. The size of the window is a free parameter and is defined by the user depending on the texture. To decide the size of the window, 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 is taken on the scale of the biggest regular feature.

For each pixel the neighborhood is defined, depending on the perceptual distances from all the occurrences of similar neighborhoods, a set is defined. And from the histogram of the center pixel values of the neighborhoods in the set a value for the current pixel is estimated.
Figure 2.7 [Efros99] Method. Given a sample texture image (left), a new image is being synthesized one pixel at a time (right). To synthesize a pixel, the algorithm first finds all neighborhoods in the sample image (boxes on the left) that are similar to the pixel’s neighborhood (box on the right) and then randomly chooses one neighborhood and takes its center to be the newly synthesized pixel. [Efros99]

For synthesizing the texture, the texture is generated in layers outward from a 3-by-3 window taken randomly from the sample image (in case of hole filling, the synthesis proceeds from the edges of the hole). Now for any point p to be synthesized only some of the pixel values in the neighborhood are known (i.e. have already been synthesized). So to solve this problem a heuristic is applied to normalize the error.

One problem of this 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 texels (or the same texels but differently
illuminated) making it hard to find close matches for the neighborhood context window. The results are presented for different window sizes.

Figure 2.8 Sample image (left), synthesized textures with neighborhood windows of sizes 5, 11, 15 and 23 respectively. [Efros99]

Figure 2.9 Other textures. [Efros99]

The major challenges in texture synthesis as posed in the paper by Wei et al. [Wei00] are-1) modeling- how to estimate the stochastic process from a given finite texture sample and 2) sampling- how to develop an efficient sampling procedure to produce new textures from a given model. They claim that both the modeling and sampling parts are essential for good texture
synthesis because the plausibility of generated textures will depend primarily on the accuracy of the modeling, while the efficiency of the sampling procedure will directly determine the computational cost of texture generation.

The inputs consist of an example texture patch and a random noise image with size specified by the user. The algorithm modifies this random noise to make it look like the given example. The two major components of the algorithm are a multiresolution pyramid and a simple searching algorithm. This searching is a new concept which was not used yet, we have also applied a similar search operation in our algorithm.

As in the case of Efros et al. their algorithm is also based on the modeling of textures by Markov Random Fields which models the texture as a realization of local and stationary random process. Each pixel is characterized by a small set of spatially neighboring pixels. To preserve the locality of the texture, the texture is generated pixel by pixel taking into account the neighborhood of the pixel.

Figure 2.10 Input texture sample and partially filled noise image [Wei00]
The algorithm raster scans the random noise image and transforms the pixels to the ones in the input texture. Only in the case of first few rows and columns the noise values are used but as the pixels are covered more and more pixels with assigned values become available and in the end synthesis look natural. In case of textures with large scale textures a multiresolution approach. This saves computation time as large neighborhoods are avoided and large structures in the texture can be represented more compactly.

The key advantages of the algorithm are quality and speed: the quality of the synthesized textures are equal to or better than those generated by previous techniques, while the computation speed is faster than those approaches that generate comparable results to their algorithm.

The algorithm described by Ashikhmin [Ashikhmin01] is similar to the one by Wei et al. in which the searching operation is used. The algorithm by Wei et al. uses the L2 norm to calculate the distances between two neighborhoods. Since the L2 norm is averaging of neighborhood pixels the edges, corners and other high level features in the image are not captured. This could lead to smoothed out edges and the human visual system being very sensitive to such details, these anomalies are easily recognized.

The Wei-Levoy algorithm starts a new search at each pixel. This algorithm considers positions of the already assigned pixels in its search for the current pixel. They have assumed that pixels from the input sample that are appropriately “forward-shifted” with respect to pixels already used in synthesis are well-suited to fill in the current pixel.
Figure 2.11 Good and bad results of Wei-Levoy algorithm [Ashikhmin01]

Figure 2.12 Synthesis result of Ashikhmin algorithm. Better than the previous algorithm in which the edges are smoothed out. [Ashikhmin01]

The selection is made only between the candidate pixels; the suitable candidate pixel is chosen to fill the current pixel only if it is completely inside the image. They have also added a user control to guide the filling process. The user can specify certain high level texture that will be used in the search algorithm. The user provides an image that shows how the resulting texture should look.
Bertalmio et al. [Bertalmio03] have tackled the problem of image filling. The applications of image filling are in image coding and wireless image transmission (e.g., recovering lost blocks), special effects (e.g., removal of objects), and image restoration (e.g., scratch removal).

Various algorithms that have been proposed in the literature fill these regions with available information from their surroundings. In cases of texture synthesis the information required for texture generation is from the input image. Since most image areas are not pure texture or pure structure, this approach provides just a first attempt in the direction of simultaneous texture and structure filling-in.

The basic idea of the algorithm in [Bertalmio03] is that the original image is first decomposed into the sum of two images, one capturing the basic image structure and one capturing the texture (and random noise). The first image is inpainted following the work by Bertalmio et al. [Bertalmio00], while the second one is filled-in with a texture synthesis algorithm following the work by Efros et al. [Efros99]. The two reconstructed images are then added back together to obtain the reconstruction of the original data. In other words, the general idea was to perform structure inpainting and texture synthesis not on the original image, but on a set of images with very different characteristics that are obtained from decomposing the given data. The decomposition is such that it produced images suited for these two reconstruction algorithms.

The algorithm worked fine for well defined structures in the image but, in case of natural images the structures do not have well defined edges so the results might not be correct. Also for large unknown regions the algorithm might not give plausible results.
The work done by Drori et al. [Drori03] comes closest to the work reported in this thesis. In the paper [Drori03] the missing regions are iteratively filled using the known image as the training set. The goal is to complete the unknown regions in an image based on the known regions, given the inverse matte and the source image. Self-similarity in the input image is required for this algorithm to give correct results.

The unknown region is approximated and a search is applied for the best fragment. The completion process consists of compositing image fragments. The confidence map is calculated at every level which guides the completion process. Initially the inverse matte defines the confidence map. The values at the low confidence regions are approximated by applying smoothing process. The confidence at the unknown areas is low but the region next to it has high confidence. The confidence values increase as the completion continues, the completion stops when the average confidence level of the image is approximately one.

The algorithm begins from the coarser image and ends with a fine image. At each level, the low confidence regions are approximated. The approximation is a rough classification of the pixels to
some underlying structure that agrees with the parts of the image with high confidence. The region is completed at fragment level. Iteratively at each level, a fragment is selected that is to be filled called the ‘target fragment’.

A search is applied for the most suitable fragment which has familiar details taken by example from a region with high confidence; this is called as the source fragment. The search extends over all the positions in five scales and eight orientations. The source and the target fragments are then composited into the image. The size of the fragment is defined adaptively depending on the image and the area to be filled. As the unknown regions are completed the confidence of the image increases and the iterations stop when it is almost one. At each level from low resolution to high resolution the confidence of the image has to converge to one for the completion process to complete.

The approximation is made with the assumption that the unknown area is smooth and so the data from the known regions in the vicinity is used to approximate the values. Approximation involves the up sampling and down sampling of the image using a local kernel at multiple resolutions.
Figure 2.14 Image Completion [Drori03]. Input image, inverse matte defining the element that is to be removed and the result.

The confidence map $\beta$ is calculated by assigning each pixel a value. It is one for all the known pixels and for unknown pixels it is calculated over the neighborhood around it. The fragment with the highest confidence value is chosen from the low confidence areas as the target fragment.

The distance between the source and the target fragments is calculated over the neighborhood. A search is applied to get the position, scale and orientation of the source fragment that minimizes the function:

$$ r = \arg \min_{s=Sr(i), t=T(i), j \in N} \sum (d(s,t)\beta_s \beta_t + (\beta_t - \beta_s)\beta_t) $$
$d(s, t)$ denotes the similarity between the features. The measure is L1 and the features are color and luminance for all pixels and gradients for known pixels. The fragments, source $s$ and target $t$, are compared by considering the corresponding pixels in them.

Depending on the spatial frequency the size of the neighborhood is defined, size is inversely proportional to the spatial frequency. Regions with high frequency are completed with small fragments and regions with low frequencies are completed with large fragments.

They have an elaborate method for merging the source and the target fragments. They construct a Laplacian pyramid for the source and target images and a Gaussian pyramid for the binary masks. At each scale the images in both the pyramids are multiplied and then added to get the Laplacian pyramid for the merged image. The composited image is then reconstructed from the pyramid. The inverse matte is again calculated for the image and the iteration continues till the confidence of the image converges to one. The completion process can be regarded as a ‘push background’ process rather than the ‘pull foreground’ process associated with image matting.

The algorithm requires extensive computation for every fragment at every scale. It takes a lot of time to complete a small portion of the image. In the next chapter, we present our approach to this problem which attempts to solve it in much less time with good results.
Our work to image completion follows the principle of figural familiarity. The missing areas are filled with familiar details taken by example from rest of the image as in [Drori03]. As outlined in chapter 2, the search for appropriate source image fragment is quite exhaustive as it extends over all possible pixels of source image, eight orientations and over five scales. The approach does not make use of any knowledge of the scene and is computationally expensive.

However, as observed in most images of natural scenes e.g. Figure 3.1, there is a strong horizontal orientation of texture/color distribution. This prompts us to limit the search only along horizontal direction and thereby reduce the search complexity extensively. We now proceed to develop an algorithm for image completion based on horizontal continuity consideration.

Let $I$ denote the source image and $\Omega$ denote the occluding figure. The image completion problem is to replace $\Omega$ by selecting appropriate fragments from the source image $I$. As a first step, a user defined matte is created to indicate the occluding portion of $\Omega$. This can be done either through image segmentation and/or using a suitable matting tool. All pixels in $\Omega$ are referred to as target pixels and the remaining pixels are termed as source pixels.
Algorithm 1.1

Based on the principle of horizontal continuity we first mark out a horizontal search strip. The search will be limited to this zone. For example, in a 256x256 image, the strip is defined as 20 pixels above and below the unknown region. So the domain to be searched is reduced considerably. The unknown matte is either hard coded as a rectangular area in the image or defined by another image.
2. Let $I_t$ be the seed-block of $m \times m$ (m=5 in this case) pixels in the target area surrounding the pixel $p \ (i, j) \in \Omega$.

3. We proceed to find a set of blocks $I_{source}$ of $m \times m$ size in the search strip of the source image (outside the matte).

4. We find the distance $d_k (I_{target}, I_{source})$ for all possible source candidates $I^k_{source}$ using the averaging norm. To calculate the distance, the average R, G, B value at each known pixel inside the strip is calculated. Average at pixel $p$ is the average of pixels in the $m \times m$ block $I_p$ around ‘p’. The distance is calculated as the mean square difference of the average at the pixels.
\[ d_k (I_{\text{target}}, I_{\text{source}}) = \sqrt{((\text{average}_{\text{red}} (p_{\text{target}}(i, j)) - \text{average}_{\text{red}} (p_{\text{source}}(i, j)))^2 + ((\text{average}_{\text{green}} (p_{\text{target}}(i, j)) - \text{average}_{\text{green}} (p_{\text{source}}(i, j)))^2 + ((\text{average}_{\text{blue}} (p_{\text{target}}(i, j)) - \text{average}_{\text{blue}} (p_{\text{source}}(i, j)))^2)\]

5. We then replace the central pixel of \( I_{\text{target}} \) by the central pixel of \( I_{\text{source}} \) for which the distance is minimum.

6. If a number of \( I^k_{\text{source}} \) patches are found to be at equal distance from the \( I_{\text{target}} \) then we randomly pick one of the patches.

7. The algorithm repeats by considering the target pixel next in the scan order in \( \Omega \).

As the unknown matte starts filling up, the averaging process gets better as inside the 5x5 block more pixels are now known.

Figure 3.5 Result of Algorithm 1.1

To illustrate the algorithm, we apply it in fig 3.5 (a) where the unknown matte is shown by white patch. The result of algorithm 1.1 is shown in Figure 3.5 (b). It is observed that the performance
of the algorithm is very poor. This can be attributed to the fact that inside the matte, the average is almost same so it ends up filling with same source pixel most of the time.

**Algorithm 1.2**

Now we improve upon the algorithm by placing more weight on pixels to the left of the target pixel, as the most likely candidates are bound to be found on left of the matte. So a new $I_t$ is proposed as shown in Figure 3.6.

![Pixel p](image)

Figure 3.6 New block to find average.

The algorithm is applied on the image as shown in Figure 3.7.

![Image](image)

Figure 3.7 Result of Algorithm 1.2
The final image looks odd because the region is filled from the source pixels only on the left side of the matte. Also since for calculating the average at the target pixel, we consider the pixels from the top and the bottom rows and then begin completing the image in scanline order we get the completed missing region dominated by the color or texture from the top-left pixels.

To give due importance to source pixels on either side of the matte, we propose the next version of the algorithm as follows.

**Algorithm 1.3**

It is observed that in many natural scenes the color distribution is totally different on the left side and the right side of the matte. To provide horizontal continuity on both the sides, the algorithm is now improved by filling the matte simultaneously from two sides, instead of filling it in the standard scanline order. For each scanline in the matte, filling proceeds by filling $pl(i, j)$ from the leftmost pixel and $pr(i, j)$ from the rightmost pixel moving towards the center of the matte. Accordingly we choose different search strips for the left and right parts. Since the pixels are filled from the nearest horizontal neighbors, the height of the search strip is reduced to 2 pixels above and below the target pixel and 40 pixels to the left or right of the pixel. The average value at each known pixel in the image is calculated. Average at pixel $p$ is the average of pixels in an $m \times m$ window around $p$. The average at the target pixels in the left and right part inside the unknown region is calculated from the source pixels to the left and right part of the unknown region respectively.
Figure 3.8 Search strip for the left and right pixels.

(a) Average Left Pixel                (b) Average Right pixel

Figure 3.9 Blocks to find averages are different for the left and right pixels
On applying algorithm 1.3 to the same image, the results are shown in Figure 3.10. A distinct dividing line now appears in the middle of the unknown region where the left and the right parts meet. Also since the images are horizontally oriented target pixels at the top and the bottom rows of the unknown region are filled from source pixels at the top and bottom part of the strip, which results in irregularities in the image.

Algorithm 1.4: Multi Scale Completion

It is observed that the adjacent pixels in unknown region $\Omega$ are mapped (filled) by same source pixel because the distance of the target pixel from a particular source pixel is almost same. The reason can be attributed to the fact that all pixels in the matte area have zero values in the beginning. To overcome this problem we follow a multilevel approach using a pyramid structure.
1. The pyramid is obtained, starting with highest resolution image and downsampling it with a simple kernel like averaging.

2. Four levels of pyramid are generated.

3. The matte at the topmost level of the pyramid is filled using algorithm 1.3.

4. The image is now upsampled and again algorithm 1.3 is applied at this level.

The advantage is that, the matte area now is not totally blank but it contains relevant portions of the source image as far as color/texture distribution is concerned.

Example steps for a 256x256 image-

- Reduce the image to 64x64. Apply the algorithm 1.3 with smaller search strip.
- Resize the image to 128x128. Apply the algorithm with a bigger search strip.
- Resize the image to original size (256x256). Apply the algorithm with the original search strip.

5. Instead of individual pixels, blocks of 5x5 pixels are replaced. Along with the source pixel the neighboring pixels are also copied.

6. Depending on the image, the different variations to this algorithm that can be tried are-

- Reducing the number of vertical pixels in the strip.
- Reducing the number of horizontal pixels.
- The number and positions of pixels considered to calculate the average of the target pixel are changed.
The result of the algorithm is as shown in Figure 3.11. The distinct line at the center of the matte is clearly visible.

Algorithm 1.5

Previously, the average of pixels in an m x m block around the source and target pixel was considered to calculate the distance between the source and the target fragments. As a result, sometimes the same pixel was used as a source for the complete row in the unknown area (Figure 3.11). This would give incorrect results in case of regions where the area around it is not smooth.

In this method, we change the averaging operation in algorithm 1.4. To calculate the distance we do not calculate the average. We calculate the difference between the source and the target fragments by taking the sum of the mean squared difference between the corresponding pixels in the fragment. The formula is given in Equation 1.
\[ d_k(I_{\text{target}}, I_{\text{source}}) = \sqrt{\sum_{\text{neighborhood}} (p_{\text{source}}^{\text{red}} - p_{\text{target}}^{\text{red}})^2 + (p_{\text{source}}^{\text{green}} - p_{\text{target}}^{\text{green}})^2 + (p_{\text{source}}^{\text{blue}} - p_{\text{target}}^{\text{blue}})^2} \]

The pixel or neighborhood with the least value is the nearest source to the target.

Figure 3.12 Result of Algorithm 1.5

The results are as shown in Figure 3.12. There still exits a divide between the left and right regions.

Algorithm 1.6: Linear Interpolation based Completion

In many natural scenes, the sky is the dominating part of the image. The sky consists mostly of smooth areas and patches of clouds. In such cases very little gain is obtained by searching for suitable textures as it is found that the areas on the left and right side of the matte have the same/similar color distribution. Here one can simply make use of linear interpolation between
the two points across the scanline. The same technique can be applied to other parts of the image where we have little texture area and the smooth areas dominate.

![Figure 3.13 Linear Interpolation Results.](image)

**Algorithm 1.7: User Defined Placement (Interactive)**

In some natural images the color and texture are similar in the horizontal direction. We can exploit this similarity in image filling. And with some user input the process can be completed in much less time.

The portion to be removed is defined in the matte, as is the case in all-previous cases. The user input that is required in this method is a ‘horizontal offset’ and a direction. The user chooses an area in the image that is to be used as source and approximately gives a pixel count as offset by which the matte should be moved. The unknown matte is moved in the user specified direction (i.e. horizontally left or right) by the number of pixels specified by the user as offset. Once the matte is in the user specified position the pixels at that area are copied and then placed in the missing region. Since the image is horizontally similar we get the filled part of the image blending easily with the neighborhood without any complex computations.
The problem with the method is that the user has to provide the two critical parameters. User needs to have some intuitive feeling about the size of the pixel so that he can specify the offset. If the image looks unnatural, the offset or the direction has to be changed and this is done till the image looks plausibly filled. This algorithm could be used as a step for further processing.

**Algorithm 1.7 (a):**

We can extend the algorithm 2.3 and apply the replacement in cases where the missing region is large. When the missing region is large only an offset and direction would not suffice as there is not enough known area to fill the region.

We then divide the missing region in horizontal strips and fill individual strips as an independent missing region. We apply the algorithm 2.3 for every strip. This requires extensive user input but the results are good.
In the figure 3.16 below we present the pseudo-codes for algorithms 1.3 and 1.4.

Algorithm 1.3:
The pixel replacement algorithm is presented below.

Function PixelAlgorithm

(Input image of any size is read in RGB format.)

1 $I \leftarrow$ Image

(The matte is read as input.)

2 $U \leftarrow$ Regular Matte

(For each row of pixels in the image, search for the pixels on the boundary of the matte. Set the pointers to the pixels on the left and right boundary of the matte i.e. set lptr and rptr to the left and right boundaries of the matte respectively.)
3 For each row, r, in the image

3.1 Initialize lptr = 1; rptr = image width

3.2 Scan row wise and until a pixel on the matte is found lptr = lptr+1 and rptr = rptr-1

(Fill the row of the matte pixel by pixel from the left and the right side simultaneously. Each pixel on the matte is passed to the function ReplacePixel. The function searches for the source pixel and replaces the unknown pixel with the source pixel.)

3.3 Until the pixels between lptr and rptr are filled

3.3.1 \( I = \text{ReplacePixel}(I, U, r, \text{lptr, side}) \)

3.3.2 lptr = lptr + 1

3.3.3 \( I = \text{ReplacePixel}(I, U, r, \text{rptr, side}) \)

3.3.4 rptr = rptr – 1

4 Return \( I \)

Function ReplacePixel (I, U, row, ptr, side)

(Set search limits for every pixel depending on the side of the matte at which it lies. The limits are decided by the user depending on the image. hLim is the number of pixels in the row to be considered in the search strip in the horizontal direction. The direction depends on the side of the pixel. vLim is the number of rows above and below the pixel row to be taken in the search strip.)

1 Search Strip = (HorizontalRange, VerticalRange)

   HorizontalRange = number of pixels to be considered in the horizontal direction in the row of ptr.
VerticalRange = number of rows to be considered for search.

1.1 If side = left

    HorizontalRange = 1 to (ptr – hLim)

    If side = right

    HorizontalRange = (ptr+1 to last pixel in the row)

1.2 VerticalRange = (row-vLim) to (row+vLim)

(Find the nearest pixel to ptr. The distance between two pixels is found by taking the difference between the average of pixels in a block of size m x m around them. The pixel in the search strip with the least distance from ptr is taken as the source pixel.)

2 For every pixel in the search strip

    2.1 Calculate the distance from ptr.

\[
    d_k(I_{\text{target}}, I_{\text{source}}) = \sqrt{((\text{average}_{\text{red}}(p_{\text{target}}(i,j)) - \text{average}_{\text{red}}(p_{\text{source}}(i,j)))^2 + ((\text{average}_{\text{green}}(p_{\text{target}}(i,j)) - \text{average}_{\text{green}}(p_{\text{source}}(i,j)))^2 + ((\text{average}_{\text{blue}}(p_{\text{target}}(i,j)) - \text{average}_{\text{blue}}(p_{\text{source}}(i,j)))^2}
\]

2.2 Source Pixel \(S_p = \text{minimum (distance (P, ptr)) where P \in \text{Search Strip}}\).

(Replace ptr with the source pixel.)

3 ptr = \(S_p\)

(Return the changed image.)

4 Return \(I\).
Algorithm 1.4:

Function MultiLevel

(*Input image of any size is read in RGB format.*)

1 $I \leftarrow$ Image

(*The matte is read as input.*)

2 $U \leftarrow$ Regular Matte

(*Create a n-level pyramid of the image and the matte by downsampling them. Starting from the top level, at every scale in the pyramid apply algorithm 1.3 and try to fill the missing region. Upsample the completed image and matte at a level and use it as input image and matte for the next level. Since there is some value assigned to the pixels in the matte at every level, the average around the pixels can be calculated with more accuracy. Hence, the pyramid approach gives better results.*)

PyramidImage(I) – Holds different levels of images of different resolutions.

PyramidImage(U) – Holds different levels of the matte of different resolutions.

3 PyramidImage(I) = downsample(I)

        PyramidMatte(U) = downsample(U)

4 For each level in the PyramidImage and PyramidMatte, starting from the top level-

    4.1 $I =$ current level of the image

        $U =$ current level of the matte
4.2 Apply the algorithm 1.3.

4.3 $I = \text{Upsample} (I)$

$U = \text{Upsample} (U)$

5 Return $I$

Figure 3.16 Pseudocode

As can be observed from the results of the algorithms presented so far, the completed region appears as if filled randomly from the nearby pixels, the texture of the surrounding area is not considered at all. In the next chapter we propose algorithms which complete the image by filling a small area in it instead of filling individual pixels i.e. by filling small blocks in the matte.
In the earlier chapter we have presented the approach to complete the missing part of the image pixel by pixel in scanline order. The results obtained are good for images with majority of high frequency content. But otherwise the completed region appears as if filled randomly from the nearby pixels, and it does not capture the texture information of the source image. To utilize the texture component of the surrounding area, we now propose to complete the unknown part with blocks of pixels instead of individual pixels. This will help to capture the appropriate texture content. We place a grid over the complete image with each cell of size \( m \times m \). The search is now made to find the appropriate block from the source to fill up the \( m \times m \) block of the matte. The algorithm and its variations are detailed below.

**Algorithm 2.1: Grid Algorithm**

In all the previous algorithms, the search for the appropriate pixel is made by making a match between the current pixel in the matte and some area in the source image. However, to have a seamless joint across the border of the matte, it would be more appropriate to locate the pixel \( p_a \) most similar to the pixel lying on the left boundary of the matte, and replace the pixel on its right by the pixel on the right of \( p_a \).

To reduce the search complexity the search can be made over areas larger than a pixel size (Figure 4.1). We propose to consider \( m \times m \) blocks for searching and replacement. For this
purpose we put a grid on whole of the image with grid size of $m \times m$. On each horizontal row we pick up the nearest $m \times m$ block \('B_b'\) lying to the left of the matte. We now search for a similar block \('B_s'\) on the left side of \('B_b'\) whose characteristics match closely with that of \('B_b'\). Then we simply replace the block on the immediate right of \('B_b'\), i.e. \('B_{br}'\), by the block, \(B_{sr}\) to the immediate right of \('B_s'\). The logic for doing this is that the figural familiarity between \('B_s'\) and \('B_{sr}'\) will be reflected across the blocks \('B_b'\) and \('B_{br}'\).

When the next horizontal row of blocks is considered, we again find the block nearest to the boundary and find the closest match as before. The replacement block now could be the block right of this block or the block below \('B_{sr}'\). It is observed that in many cases they turn out to be the same blocks. If it is not so we can make a selection and replace it with the block that matches the best between the two blocks.

A similar procedure is used to start filling up the blocks along the right boundary of the matte. For the block \('B_b'\), next to the right boundary, we locate the most similar block \('B_s'\) by searching along the horizontal row in the right part of the source image. Now we replace the block \('B_{bl}'\) immediate left to \('B_b'\) with the block \('B_{sl}'\) which is immediate left of \('B_s'\). Again at each stage of filling the other blocks inside the matte, care is taken to ensure that replaced blocks match sidewise as well as with the blocks on the top. This helps in maintaining the figural continuity across the matte area.
Figure 4.1 Search and Replace. Sample grid of m x m is shown inside the image. The grid blocks are defined as above.

The method will result in uniformly filling up the matte row under consideration from the left and the right sides. However, it is quite likely that while blocks from left and right side of the matte are filled in properly, there may be a mismatch where the left and the right blocks meet at the centre of the matte. It is because the source image on the left side of the matte may not agree totally with the right side of the matte.

One possible solution to this problem would be a texture interpolation for middle three or five blocks, or a random displacement of the blocks by two or three pixels at the centre of the matte.
Correction for Self-Replacement:

We fill the unknown region by filling in the grid blocks that fall inside it. In most of the images the portion to be removed is one single large body with a convex shape. Considering a grid row, the unknown region is continuous in such shapes. In some images, the shape of the missing region is concave which could lead to some regions with known values that lie between the unknown regions. This causes discontinuities in the missing area along a row of the grid. In such cases, there could be some blocks with known values of pixels between the blocks that contain pixels with unknown values.

We apply the search procedure on the image we move along the grid row, filling in the blocks in the matte as we proceed. When we come across a row where there are such discontinuities, the result of search for source blocks might be some blocks that are themselves unknown. This means that the block in the matte is being filled by another block in the matte itself. This gives completely incorrect results as the blocks are filled with wrong source blocks. And as we proceed further, the error is propagated along the row.

We use the following method to handle such images. During the first pass, we keep track of the blocks with unknown values that are incorrectly filled by other blocks with unknown values. To keep track, we assign a particular error value to the pixels that are incorrectly filled. This error value that is assigned is generally a matte color (for example bright red color in the Figure below) that is missing from the original image. In the next pass, we create another matte with the error values as the missing regions. Now, the input image to the second pass is the output image of the first pass which has the error pixels in it. Using this matte and the image as input to the
algorithm we run the second pass of the algorithm. Since majority of the original missing blocks are filled in the first pass, there are not many missing regions in the second pass. This reduces the chances of the pixels being filled by pixels with unknown values again.

![Image](image1)

**Figure 4.2 Self Replacement.** The red areas in (c) are the regions where the elephant is filled from the matte itself. Iteration is applied to fill up the incorrectly filled region.

The number of passes has to be decided dynamically. The number of passes depends on the shape of the missing region in the image. This iterative method yields better image completion results.
**Correction for center line and boundary irregularities:**

While blocks from left and right side of the matte are filled in properly, it results in mismatch where the left and the right blocks meet at the centre of the matte. It is because the source image on the left side of the matte may not agree totally with the right side of the matte.

For the removal of the conspicuous central line between the blocks filled from the left and right side we have applied certain techniques depending on the image. In some cases such a line can be observed along the boundary of the missing region also. This is due to the fact that the blocks that are filled after the search may not be continuous in color and texture with the block immediately next to it.

The various steps that we suggest here are based on linear interpolation. It is similar to the algorithm 1.6. In cases where a line appears at the boundary of the unknown region, certain number of pixels from both sides of the boundary is selected. Linear interpolation is applied along these pixels. In the same way, for the line appearing at the center of the missing region some pixels on both sides of the line interpolated.

In some images the area immediately next to the missing region is smooth but the contrast between the left and right sides of the missing region is large. In such cases applying the search gives unnatural results. This is because the smoothness of left side would propagate in the missing region and same on the right side. But when they meet at the center a dividing line is clearly formed. So instead of applying the search the pixels in the missing area are linearly
interpolated along the row. This gives smoother and better results than the results by performing
the search. The steps have to be applied depending on the structure and the texture of the image.

In some cases the blocks picked up for replacement are algorithmically correct but because of the
structure of the image the completion looks unnatural. So for some images the matte has to be
modified to include some of the known part to make up for its structure.

![Figure 4.3 Extended Matte](image)

**Tilted Orientations:**

In natural images of mountains and similar slopes the image does not have perfect horizontal
orientation. If the missing region encompasses such a slope then there is certain unnaturalness in
the algorithmically filled part. In the middle of the missing region a line is formed where the left
and right parts. For example, in case of a mountain slope, the mountain would fill one half and
other half by the sky and a straight line would divide the two regions. The slope of the mountain
is lost in the final part and abrupt region appears in the filled region.
To solve this problem, we have used multi colored mattes (Figure 4.4). In regions of slopes we use multicolored mattes; each color specifies the area from where it is to be filled, say, green colored matte should be filled from right only and blue colored matte should be filled from left only. Results are shown in Figure 4.5 and Figure 4.6.
Figure 4.5 Tilted Orientations [mountainlake]

Figure 4.6 Colored Matte
In some cases, the matte is very close to the edge of the image. Since there is not enough information on both the sides of the matte, filling from only one side is preferred. So to specify the side from which the matte should be filled, colored matte can be used (Figure 4.7).

Figure 4.7 Matte completed from the left portion
In the figure below we present the pseudo-codes for the above described algorithms.

Algorithm 1.2:

Function GridAlgorithm

(Input image of any size is read in RGB format.)

1 $I$ ← Image

(The matte is read as input.)

2 $U$ ← Regular Matte

3 Create a grid $G$ with block size $m \times m$ on the matte.

(A grid is defined over the image to make the search process systematic and fast. Search is applied over blocks which saves computation time and gives better results. Generally, a size of 10 x 10 blocks is taken for most images.)

4 For each row in the grid on the image

(Search for the blocks on the boundary of the matte. For each row in the grid look for boundary of the matte until the left and right boundaries are found, perform the following steps.)

4.1 Start from the leftmost block of the grid on the image -

- If the left boundary of the matte is in the block then select the block as starting block of the row, $b_{uleft}$

- Else move to the next block towards right

4.2 Start from the rightmost block of the grid on the image -
- If the right boundary of the matte is in the block then select the block as ending block of the row, $b_{\text{uright}}$

- Else move to the next block towards left

4.3 The position of the left and right blocks is noted in $(\text{gridRow}, \text{gridColLeft})$ and $(\text{gridRow}, \text{gridColRight})$ respectively.

(Fill the blocks from left and right sides of the regular matte. The block in the matte that is to be filled is passed to a function ReplaceBlock. The function searches for the suitable match, fills the block and returns the image with the filled block. Start filling $b_{\text{uleft}}$ and $b_{\text{uright}}$ block at the boundary of unknown matte simultaneously.)

4.4 Until all the blocks between $b_{\text{uleft}}$ and $b_{\text{uright}}$ are filled, perform 4.4.1 and 4.4.2-

4.4.1 For $b_{\text{uleft}}$

$I = \text{ReplaceBlock} (I, U, \text{gridRow, gridColLeft, left, iteration})$

$\text{gridColLeft} = \text{gridColLeft} + \text{blockWidth}$

$b_{\text{uleft}} = \text{block at gridColLeft}$

4.4.2 For $b_{\text{uright}}$

$I = \text{ReplaceBlock} (I, U, \text{gridRow, gridColRight, right, iteration})$

$\text{gridColRight} = \text{gridColRight} - \text{blockWidth}$

$b_{\text{uright}} = \text{block at gridColRight}$

(If ‘iteration’ returned from the function ‘ReplaceBlock’ is true then there are some blocks in the filled region that are filled from other blocks inside the matte. This gives incorrect result, so we perform another iteration of the algorithm with different matte. Create the unknown matte
with the error color as the missing region. Error value is some color value that is missing in the original image I. ‘Iteration’ as true for the algorithm to run the next iteration.)

5 If iteration = true

5.1 \( U \) = New matte with the error value as the missing region.

5.2 \( I \) = output image of the previous iteration.

5.3 Perform step 4.

(Return the completed image)

6 Return \( I \).

Algorithm (Tilted Orientations):
The algorithm presented below handles the images with the mattes having tilted structures in them that are to be filled. The user specifies the tilted area in the blue and the green mattes.

Function TiltedOrientation

(Input image of any size is read in RGB format.)

1 \( I \) \( \leftarrow \) Image

(The mattes are read as input. The colored mattes are taken to handle the case of tilted objects in the regular matte.)

2 \( U \) \( \leftarrow \) Regular Matte

3 \( B \) \( \leftarrow \) Blue Matte

4 \( G \) \( \leftarrow \) Green Matte
(A grid is defined over the image to make the search process systematic and fast. Search is applied over blocks which saves computation time and gives better results. Generally, a size of 10 x 10 blocks is taken for most images.)

5 Create a grid G with block size m x m on the matte.

(Search for the blocks on the boundary of the matte. For each row in the grid look for boundary of the matte until the left and right boundaries are found, perform the following steps.)

6 For each row in the grid on the image

6.1 Start from the leftmost block of the grid on the image -

- If the left boundary of the matte is in the block then select the block as starting block of the row, \( b_{u\text{left}} \)
- Else move to the next block towards right

6.2 Start from the rightmost block of the grid on the image -

- If the right boundary of the matte is in the block then select the block as ending block of the row, \( b_{u\text{right}} \)
- Else move to the next block towards left

6.3 The position of the left and right blocks is noted in (gridRow, gridColLeft) and (gridRow, gridColRight) respectively.

(Fill the blocks from left and right sides of the regular matte. The block in the matte that is to be filled is passed to a function ReplaceBlock. The function searches for the suitable match, fills the block and returns the image with the filled block. Fill the image using the matte. Start filling \( b_{u\text{left}} \) and \( b_{u\text{right}} \) block at the boundary of unknown matte simultaneously.)
6.4 Until all the blocks between \( b_{\text{uleft}} \) and \( b_{\text{uright}} \) are filled, perform 6.4.1 and 6.4.2-

6.4.1 For \( b_{\text{uleft}} \)

\[
I = \text{ReplaceBlock} \left( I, U, \text{gridRow}, \text{gridColLeft}, \text{left}, \text{iteration} \right)
\]

\[
\text{gridColLeft} = \text{gridColLeft} + \text{blockWidth}
\]

\[
b_{\text{uleft}} = \text{block at gridColLeft}
\]

6.4.2 For \( b_{\text{uright}} \)

\[
I = \text{ReplaceBlock} \left( I, U, \text{gridRow}, \text{gridColRight}, \text{right}, \text{iteration} \right)
\]

\[
\text{gridColRight} = \text{gridColRight} - \text{blockWidth}
\]

\[
b_{\text{uright}} = \text{block at gridColRight}
\]

(If ‘iteration’ returned from the function ‘ReplaceBlock’ is true then there are some blocks in the filled region that are filled from other blocks inside the matte. This gives incorrect result, so we perform another iteration of the algorithm with different matte. Create the unknown matte with the error color as the missing region. Error value is some color value that is missing in the original image \( I \). ‘Iteration’ as true for the algorithm to run the next iteration.)

7 If iteration = true

7.1 \( U = \) New matte with the error value as the missing region.

7.2 \( I = \) output image of the previous iteration.

7.3 Perform step 6.

(For the blue matte)

8 Change the unknown matte to get the area of the blue colored matte.

8.1 Apply step 6.

(Fill the area from left side only)
8.2 For the blocks between $b_{\text{uleft}}$ and $b_{\text{uright}}$

$$I = \text{ReplaceBlock}(I, B, \text{gridRow}, \text{gridCol}, \text{left})$$

(For the green matte)

9 Change the unknown matte to get the area of the green colored matte.

9.1 Apply step 6.

(Fill the area from right side only)

9.2 For the blocks between $b_{\text{uleft}}$ and $b_{\text{uright}}$

$$I = \text{ReplaceBlock}(I, G, \text{gridRow}, \text{gridCol}, \text{right})$$

(Return the completed image)

10 Return $I$

Function $\text{ReplaceBlock}(I, U, \text{gridRow}, \text{gridCol}, \text{side})$

(ReplaceBlock takes as input the block in the matte that is to be filled. The search strip for the block is defined and search is applied to get the suitable block. The target block used is the block on the left or right side of the input block; it depends on the side of the input block.)

(Select the target block.)

1 Input block = The $m \times m$ sized block at the position (gridRow, gridCol) on the grid.

$T_b = \text{Target block}$

1.1 If side=left, $T_b$ is on left of the input block, $b_{\text{uleft}}$.

1.2 If side=right, $T_b$ is on the right of the input block, $b_{\text{uright}}$. 


(Set search limits for the target block.)

2 HorizontalLimit = number of blocks to be considered in the horizontal direction in the row of the input block.

VerticalLimit = number of rows to be considered for search.

2.1 If side=left

   The blocks in the horizontal direction which should be selected are in the limit-
   HorizontalLimit = First block to Block before target block on left

If side=right

   HorizontalLimit = Block next to target block on the right to LastBlock

2.2 VerticalLimit = the row of the input block

(Once the limits are set, find the nearest block to the target. For every block in the search strip calculate the distance from the target. The distance between two blocks is calculated as the L2 norm, i.e. mean squared difference between the corresponding pixels in the blocks. The block with the least distance is considered as the source block $S_b$.)

3 For every block in the search strip

   3.1 Calculate the distance between the source and the target blocks, using Equation 3.1.

   3.2 Find the block with the least distance.

4 The block with the least distance is considered as the source block $S_b$.

(Depending on the side of the input block the replacement block is selected.)

5 $R_b = $ replacement block
5.1 If side = left, R_b is the block on the right of S_b

If side = right, R_b is the block on the left of S_b

(If R_b is in the matte, then the selected replacement block is not correct so replace the block with some color value (for example, red) to signify the error in the fill process. Error value is some color value that is missing in the original image I. Set iteration as true for the algorithm to run next iteration.)

6 If R_b is in the matte-

6.1 R_b = Error color.

6.2 Iteration = true

(Place R_b on the input block, i.e. fill the block in the matte with R_b.)

6.3 Input block = R_b

(Return the changed image.)

7 Return I

Figure 4.8 Pseudo Code

The above presented algorithms give satisfactory completed images. The time required to complete the images is very less as compared to the algorithm described in [Drori03]. In the next chapter, we present the comparison of computational requirement between our algorithm vis-à-vis the algorithm by Drori et al. in [Drori03].
CHAPTER 5: CONCLUSIONS

In this work, we have presented a technique for image filling. We have shown that even by using a restricted area of the source image, the matte area can be filled with background texture. In this chapter we make a comparison of computational requirement vis-à-vis another approach [Drori03] used for image filling which is closest to our approach. The authors in [Drori03] fill the image matte by applying extensive search over all positions and eight orientations in the image. Also, it considers the texture frequency of the image to decide the size of the fragment. As we shall show below, their method is very time consuming.

Our algorithm is orders of magnitude faster than [Drori03]. Our search is limited to a very small and restricted area in the image. The algorithm takes advantage of the fact that all the natural images have horizontal distribution of texture and color. Therefore the search is made over a small region i.e. a horizontal strip next to the missing region.

Computations in [Drori03]:

The authors use a multi level approach. They consider the image pyramid and perform the operations at each level starting from the coarsest image. The operations at each level are detailed below.
Approximation

In this step, they generate approximate values for the missing region. This approximated image is used in further steps. Approximation is achieved by downsampling a few levels and then upsampling back to the original size the image using a kernel

\[ Y_{t+1}^l = (Y_t^l \alpha + C)(*K_\epsilon \downarrow)^i(*K_\epsilon \uparrow)^i \]

where,

- \( l \) – Current level
- \( t \) – Iteration
- \( Y \) – Approximation output
- \( C \) – Image
- \( K_\epsilon \) – Kernel

Using a kernel of the size \( m \) and 3 levels, the number of operations in this step are \( 6mn^2 \).

Calculation of the confidence map over the approximated image

\[ \beta_i = \begin{cases} 1 & \text{if } \alpha_i = 1 \\ \sum_{j \in N} g_j (\alpha_j)^2 & \text{otherwise} \end{cases} \]

The operation is done on the whole of the image. If the size of the matte is \( M \), then the number of operations in this step is \( mM \), as on the remaining part of the image, \( \beta_i \) is 1.

Calculation of the level set

\[ \beta_i = \begin{cases} 0 & \text{if } \beta_i > \mu(\beta) \\ \beta_i + \rho[0,\sigma(\beta)] & \text{otherwise} \end{cases} \]

The number of operations in this step is \( M \) (size of the matte).
Searching

This is the most expensive operation in the algorithm. It involves the calculation of distance between the fragments using the following formula –

\[
\begin{align*}
    r &= \arg \min_{s=Sr(i), t=T(i), i \in N} \sum (d(s, t)\beta_s\beta_t + (\beta_t - \beta_s)\beta_t) \\
    &= \arg \min_{s=Sr(i), t=T(i), i \in N} \sum (d(s, t)\beta_s\beta_t + (\beta_t - \beta_s)\beta_t)
\end{align*}
\]

If \(d(s, t)\) is L2 norm then the computations involved to find the distance between the pixels in the neighborhood are 8. Therefore, the complete equation involves 13 operations. Over the neighborhood of size \(m\), there would be 13m operations.

This formula is applied over a large number of fragments to find the best match. It is applied over all the positions in the fragments at other levels and all the orientations. For the search at other levels, since the size reduces by \(n/4\) with each level the computations involved at the highest resolution level (given image) are (considering 4 levels) –

\[
13m \times 8 \times (1 + \frac{1}{4} + \frac{1}{16} + \frac{1}{64}) = 8840mn^2 / 64
\]

Considering the computations at all the four levels for searching, the total computations in searching operation are –

\[
(104mn^2 / 64) + (520mn^2 / 64) + (2184mn^2 / 64) + (8840mn^2 / 64)
\]

\[
= 11648mn^2 / 64
\]

For a two level approach, the computations involved are –

\[
= 624mn^2 / 4.
\]
Compositing

The Laplacian and Gaussian pyramids of the image and the matte are generated to merge the source and target fragments. The following operation is carried at each level of the pyramid. The number of levels in the pyramid is 3, the number of computations involved are \(3n^2m\) (\(m\) is the size of the kernel).

The steps after approximating the image are performed continuously until the image converges i.e. the confidence level in the image averages to 1. So the operations are performed ‘iteration’ number of times (assuming every level requires ‘iteration’ number of passes to converge).

Total Computational Requirement

Assuming the algorithm does 4 iterations on an average for the confidence level of the image nears 1, the total number of computations (at highest level) =

\[6mn^2 + \{mM + M + (11648mn^2 / 64) + 3mn^2\} \times 4\]

For a typical image of size 256 X 256, a typical matte of 50 X 75 and a neighborhood of 3 X 3, the total number of operations (at the highest level) turn out to be -

\[= 440158704\]
\[= 4.4 \times 10^8\]

Computations in our method:

As we have indicated a number of variations of our approach in chapters 3 and 4, we consider the most extensive one which is the Grid algorithm that handles tilted orientations (multi colored
matte) and is computationally the heaviest. The computations involved in our algorithm are as follows –

**Traversing**

Traverse the image to get the matte region. The operations involved are checking every block in the grid over the image. The computations involved are the number of blocks \( n_b \) in the grid. \((m \text{ is the size of the block in the grid)}\)

\[
n_b = \frac{n^2}{m}
\]

**Searching**

For all the blocks in the matte search is applied for the suitable block. Searching involves finding a source block in the search strip to replace the block in the matte.

Let the size of the search strip be denoted by \( \text{size} = (n_{\text{Rows}} \times n_{\text{Cols}}) \). \( n_{\text{Rows}} \) is the number of rows in the search strip and \( n_{\text{Cols}} \) is the number of blocks in each row in the search strip. 

\( n_{\text{Cols}} \) does not exceed half the number of blocks in a row in the grid.

Typically, \( n_{\text{Rows}} = 3 \) and \( n_{\text{Cols}} = \frac{n}{(2m)} \).

Therefore size of a search strip, \( \text{size} = \frac{3n}{(2m)} \).

**Distance calculation using the L2 norm**

\[
d_k(I_{\text{target}}, I_{\text{source}}) = \sqrt{\sum_{\text{neighborhood}} (p_{\text{red source}} - p_{\text{red target}})^2 + (p_{\text{green source}} - p_{\text{green target}})^2 + (p_{\text{blue source}} - p_{\text{blue target}})^2}
\]
It takes $9m$ operations ($m$ is the size of the neighborhood or block) to find the distance over one neighborhood. So over a search strip the number of operations is ($9m$ times size).

A search strip is applied for every block inside the matte, so the computations required are for every block in the matte i.e. $M/m$.

Therefore the computations involved for all the blocks in the matte =

$$= 9m \text{ size } M/m.$$ 

$$= 9m (3n / (2m)) (M/m)$$

$$= 27nM / 2m^2$$

This operation is repeated for the other colored mattes. But since the area of the unknown region remains the same irrespective of the color of the matte, the computation time is included in the above step.

**Total Computations in our method**

The total computations in our method considering all the steps

$$= (n_b + 27nM / 2m^2)$$

$$= (n^2 / m) + (27nM / 2m^2)$$

For a typical image of size 256 X 256, a typical matte of 50 X 75 and a neighborhood of 3 X 3, the total number of operations turn out to be $= 167282$

$$= 1.6 \times 10^5$$
Timing Estimates:

Our method does not apply the computations over many levels; also the search operation is only over a small region in the image and not over different levels and orientations of the image as in [Drori03]. This considerably reduces the computation time of our algorithm. As can be seen from the above analysis, the time taken by the algorithm [Drori03] exceeds far too much than the time taken by our algorithm. The authors in their paper [Drori03] indicate that the computation times range between 120 and 419 seconds for 192 by 128 images and between 83 and 158 minutes for 384 by 256 images on a 2.4 GHz PC processor [Drori03]. In our case, the computation time for images of 332 by 223 ranges from 10 seconds to 25 seconds depending on the matte area, while there is little degradation in the quality of the filled image as compared to [Drori03].
LIST OF REFERENCES


[mountainlake] www.mountainlake.com
