Fast Algorithm for Completion of Images with Natural Scenes

Article · January 2004

CITATIONS
14

READS
302

3 authors, including:

Kanad Biswas
Indian Institute of Technology Delhi
101 PUBLICATIONS **1,140** CITATIONS

SEE PROFILE

Sumanta Pattanaik
University of Central Florida
124 PUBLICATIONS **5,326** CITATIONS

SEE PROFILE

Fast Algorithm for Completion of Images with Natural Scenes

Siddharth Borikar K.K.Biswas Sumanta Pattanaik
School of Computer Science
University of Central Florida, Orlando Florida
sborikar@cs.ucf.edu kanad@cs.ucf.edu sumant@cs.ucf.edu

Abstract

We present a fast method for completion of images of natural scenery, where the removal of a foreground object creates a hole in the image. Because of the strong horizontal orientation in natural scenes, it is intuitive to synthesize the missing part by image fragments drawn from horizontally located areas, without the need to search whole of the image area to find the appropriate texture. We are thus able to select the texture that naturally blends into the rest of the scene horizontally. We also propose a modification to deal with sloping horizons, such as mountain areas, and show how the texture filling areas can be suitably chosen. We demonstrate the method by completion of a variety of scenes.

1. Introduction

The removal of portions of an image is an important tool in photo-editing and film post-production, such as image restoration (e.g. scratch removal) and special effects (e.g. removal of objects).

Image completion is an area related to texture synthesis. Inpainting techniques were naturally extended from paintings to images and films [Bertalmio00]. Image inpainting and image completion 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. The region to be removed is specified by the user. It is, generally, some foreground element that needs to be taken off the scene. After removing the foreground the area is to be filled so that the image looks naturally complete [Drori03].

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.

Our approach focuses on image based completion, with no knowledge of the underlying scene [Drori03]. The image is completed by searching for appropriate textures all over the image, such that on completion it preserves the natural appearance of the image. However, it is observed that in images with natural scenery, 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. We follow the principle of figural familiarity and using the image as our training set, complete the image limiting the search to horizontal neighborhood.

The assumption of horizontal orientation in natural images is also supported by the Fourier transform of such images. Fourier transforms of two such images are shown in Figure 1. 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.

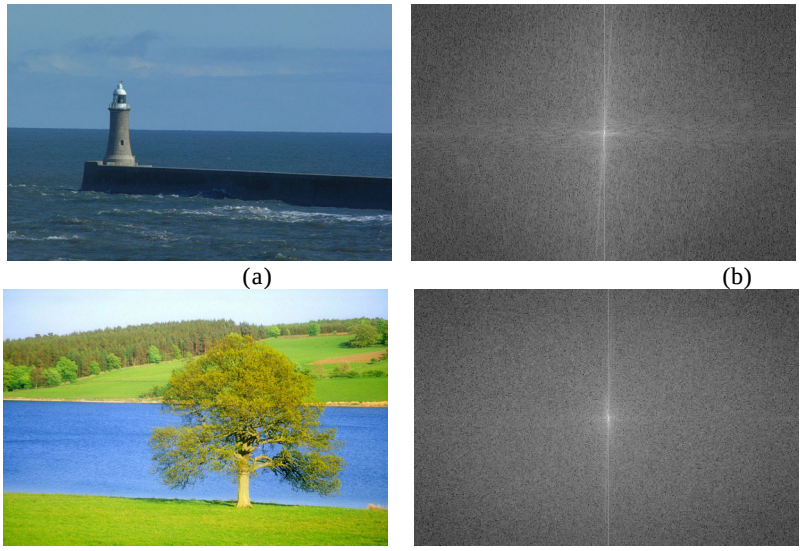
The organization of the paper is as follows. In section 2, we briefly discuss some of the related work. In section 3 we present our algorithm on image completion. In section 4, we make a comparison of computational requirements of our algorithm with that of [Drori03].

2. Related Work

Image inpainting deals with repairing damaged pictures or removing unnecessary elements from pictures, and has applications in restoration of damaged paintings and photographs. An algorithm based on the techniques used by professional restorators in image restoration is presented in [Bertalmio00]. 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.

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.

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]. The noisy image is converted to another image using the original image and a synthetic texture which is derived from the target.



(a) (b)

(c) (d)

Figure 1. Natural scenery images with their Fourier transform

The algorithm is derived from [Heeger95], where at every iteration; the original image is composited back into the noise image according to the mask using a multi-resolution compositing technique that avoids blurring and aliasing.

Efros et al. [Efros99] have modeled the texture as a Markov Random Field (MRF) which indicates 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. Based on this principle the neighborhood of the pixel is considered for generating the new texture.

Wei et al. [Wei 00] also use the concept of Markov Random Fields. Each pixel of the texture is characterized by a small set of spatially neighboring pixels. They generate the texture pixel by pixel by applying a search over source texture such that local similarity is preserved between the source texture and the resultant image.

An approach, similar to the one by Wei et al. is followed in [Ashikhmin01], where the search extends to finding appropriate pixels in source texture for a number of pixels in a L shaped neighborhood of the pixel to be filled in the scan line.

Bertalmio et al. [Bertalmio03] have tackled the problem of image filling. Since most image areas are neither pure texture nor pure structure, this approach is a first attempt in the direction of simultaneous texture and structure filling-in. The basic idea is to decompose the original image into two images, one capturing the basic image structure and the other capturing the texture (and random noise). The first image is inpainted following [Bertalmio00], while the second one is filled-in with a texture synthesis algorithm [Efros99]. The two reconstructed images are then added back together to reconstruct the original data.

The work done by Drori et al. [Drori03] comes closest to the work reported in our paper. In [Drori03] the missing region is defined through an inverse matte and is iteratively filled using the remaining image as the training set using the principle of self-similarity. The algorithm fills the matte by filling small regions in it called fragments. The authors have taken a multi resolution approach in which the algorithm progresses from the coarse level to the fine level. For every fragment the computation includes selection of the candidate fragment for filling, searching for the appropriate replacement i.e. the source fragment at all levels, and in eight different orientation, and finally compositing the candidate and the source fragment. Fragments at every scale are completed with increasing confidence, and thus it takes a lot of time to complete even a small portion of the image.

3. The Proposed Image Completion Method

Our approach 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]. However, as observed in most images involving natural scenes e.g. Figure 1, there is a strong horizontal orientation of texture/color distribution. This prompts us to limit the search just to

horizontal neighborhoods and thereby reduce the search complexity extensively.

To capture properly the texture content, we propose to complete the unknown region with blocks of pixels instead of individual pixels. We place a grid over the complete image with cell size $m \times m$. Now we search for the most appropriate block from the source image to fill up the $m \times m$ block of the matte. The algorithm is detailed below.

3.1. Grid Algorithm

For the given matte (the hole created by removing the foreground object), Drori et al. conduct the search for the appropriate pixel by looking for a match between the fragment surrounding the current pixel in the matte and fragments in rest of the remaining 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.

In our method, we consider $m \times m$ sized fragments (blocks) instead of individual pixels. We propose to carry out a search for $m \times m$ blocks for replacement. To maintain figural similarity, we start filling up the left portion of the matte with image fragments from the left side of the matte, and the right portion of the matte with fragments from portion of the image lying to right of the matte (Figure 2).

3.1.1 Completion from Left side. On each horizontal row we select 'B_b', the block lying closest to the left of the matte boundary. We now search the source image for 'B_s', a block whose characteristics match closely with that of 'B_b', the measure of closeness being the L_2 norm. Then we simply replace 'B_{br}', the block on the immediate right of 'B_b', by B_{sr}, the block to the immediate right of 'B_s'. The underlying idea is that it will capture the figural familiarity between 'B_s' and 'B_{sr}' and ensure that it is reflected across the blocks 'B_b' and 'B_{br}' as well. 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 either be the one right of this block, or the block located immediately below 'B_{sr}'. It is observed that in most cases, it turns out to be the same block. If it is not so, we make a selection based on how closely it fits in with other replaced blocks in the matte and the blocks near the boundary of the matte.

3.1.2 Completion from Right side. A similar procedure is used to start filling up the blocks along the right boundary of the matte. For the block 'B_b', located next to the right boundary of the matte, we locate 'B_s', the block with most similar characteristics by searching along the horizontal row in the right part of the source image. Now we replace 'B_{bl}', the immediately left block of 'Bb', with 'Bsl', which is located 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.

The method will result in filling up the matte row under consideration uniformly from both 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 might be a mismatch at the point where the blocks from the left and the right collide at the middle 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 to carry out 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. The outlined algorithm is illustrated in Figure 3, where one of the bushes has been removed from the foreground.

3.2. Correction for Self-Replacement

We fill-up the unknown region of the matte with grid blocks from the source image. In most of the images the removed foreground portion is one single area 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 could be concave. This may cause a small part of the source image to be available between two limbs of the concave region. Due to lack of sufficient number of blocks along the horizontal row, the algorithm would attempt to select some blocks from the matte area itself (self-replacement), causing figural discontinuities in the missing area along one or more rows.

When we come across a row where there are such discontinuities, the result of search for the source blocks would recover some blocks that are themselves unknown (low confidence). This means that the block in the matte is being filled in by another block from the matte itself. As we proceed further, the error gets propagated along the row.

We use the following method to handle such images. During the first pass, we keep track of the matte blocks that are incorrectly filled by blocks with matte blocks. We do this by assigning an error flag to the blocks that are incorrectly filled. The flagged blocks are assigned a new matte color (say bright red color) as shown in Figure 4. In the next pass, we create

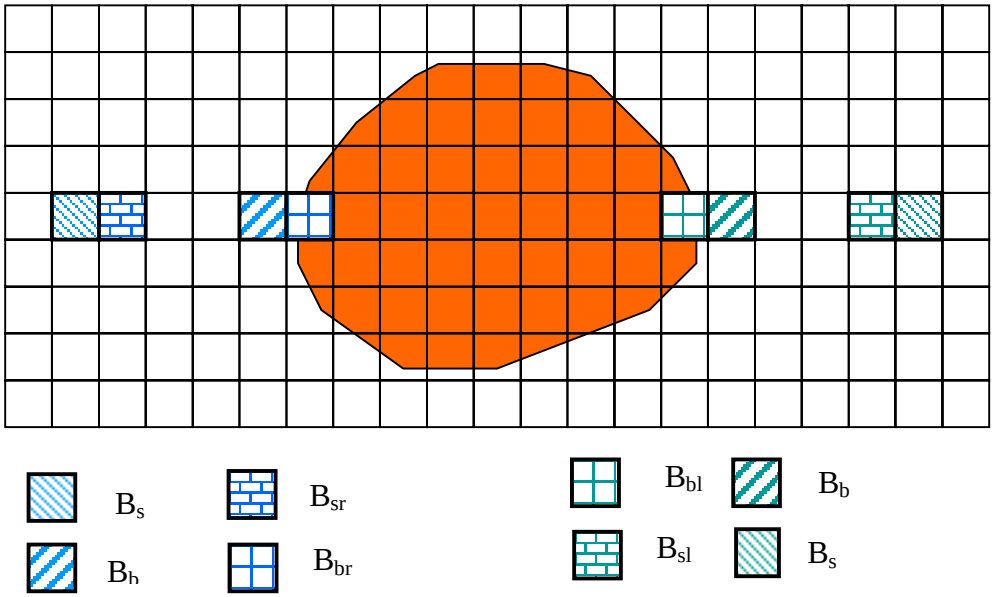
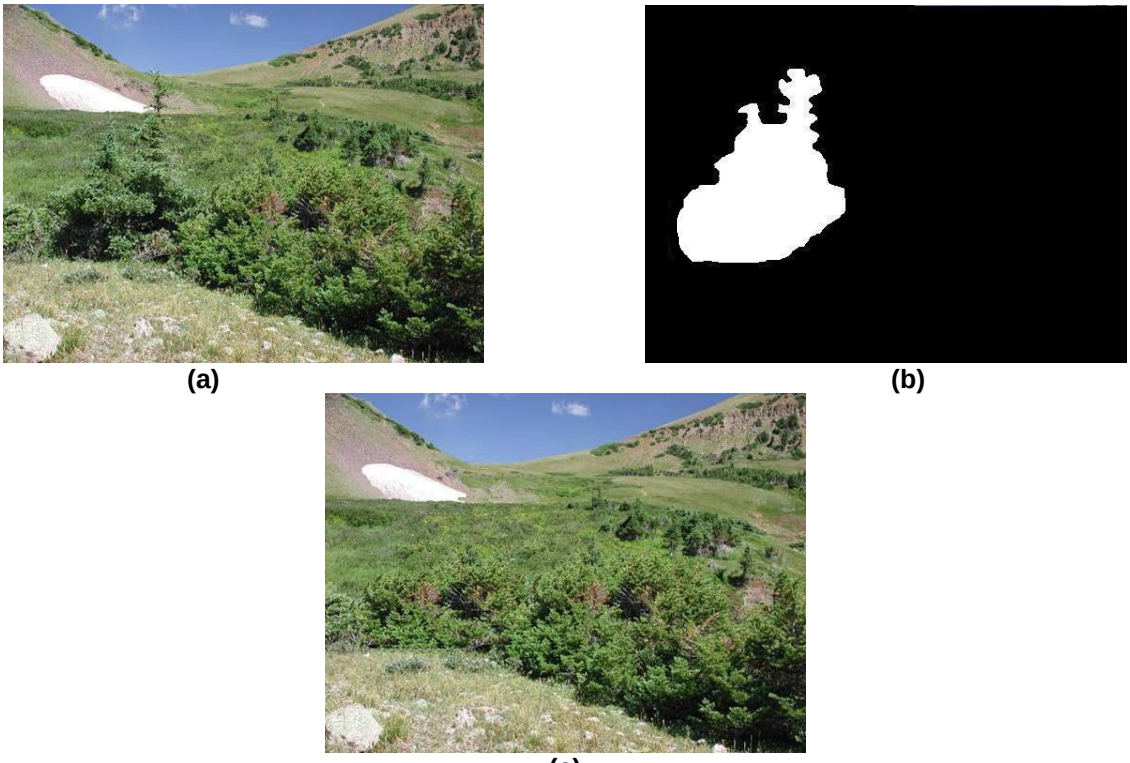


Figure 2. The m × m sized grid and the Block replacement policy



(a) **(b)**

(c)

Figure 3. Completion result

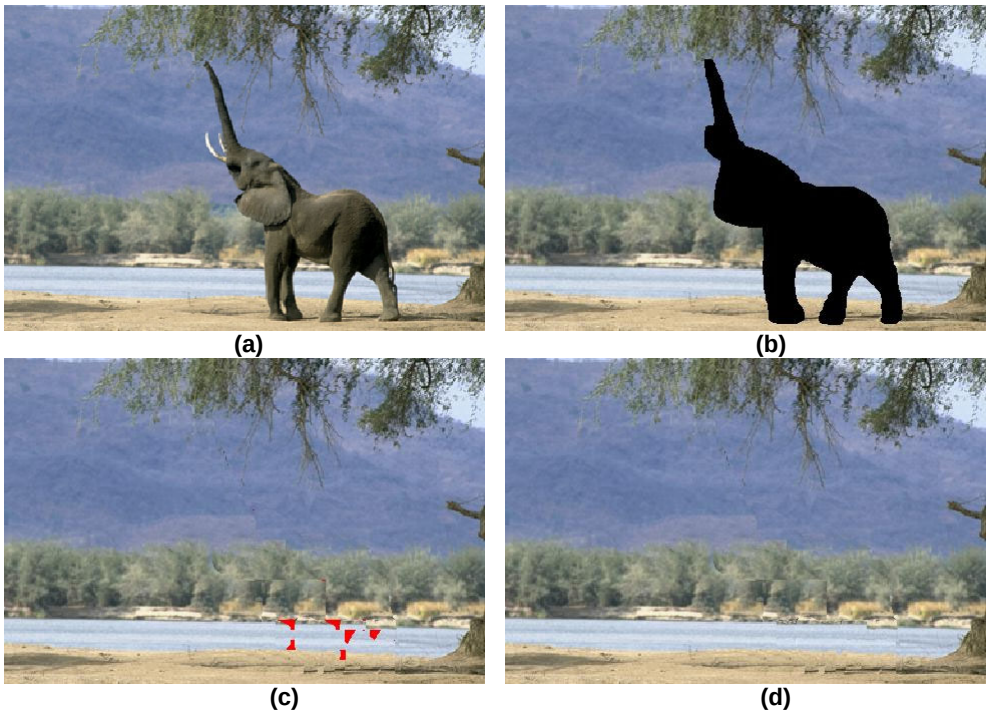


Figure 4. Self Replacement correction: (a) The original image (b) The matte created by removing the elephant from the image. (c) Areas marked red are the regions where the matte was filled with blocks from the matte itself. (d) The completed image after two passes of the grid algorithm.

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.

3.3. Tilted Orientations

Images of natural scenery containing mountains and similar slopes do not have perfect horizontal orientation. The slope could be quite appreciable. If the matte or a portion of the matte encompasses part of such terrain, the above algorithm would produce incorrect results, as it will try to fill in blocks from adjacent horizontal rows. The scene would not have the naturally pleasing look. In the middle of the missing region a line appears separating two parts at different heights. For example, in case of a mountain slope, blocks from the mountain area would fill one half of the matte, while the other half would be filled by blocks from the sky area. The slanting part of the mountain is lost in the replaced part and an odd shaped region shows up. Figure 5(a) shows scenery where we are trying to remove boats on the right side of the image. The result of applying the above algorithm is shown in Figure 5(b). A small forest area shows up at a greater height when the boat with the red, orange and green stripes is replaced. The matte used is shown in Figure 5(c).

The normal matte, filled up from the left and the right side will exhibit the problem shown in Figure 6(a). To take care of this problem we propose a two colored matte solution. The matte is divided in two parts along the slope in such a way that the two halves fall in differently textured region (Figure 6(b)). This could be done either by the user, or by carrying out an examination of the blocks around the central line and keep on rotating the line till the matte is neatly divided in appropriate regions. Once this is done, it becomes very easy to fill up the two matte areas. Each color specifies the area from where it is to be filled, say, the green colored matte be filled from blocks from just the right side (Figure 6(b)), and blue colored matte be filled with blocks from left side (Figure 6(c)). This would result in Figure 6(d) where figural continuity is perfectly maintained. A suitable matte for Figure 5(a) is shown in Figure 8 while the final results are shown in Figure 7(b).

In some cases, the matte happens to 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, another colored matte can be used (Figure 7).

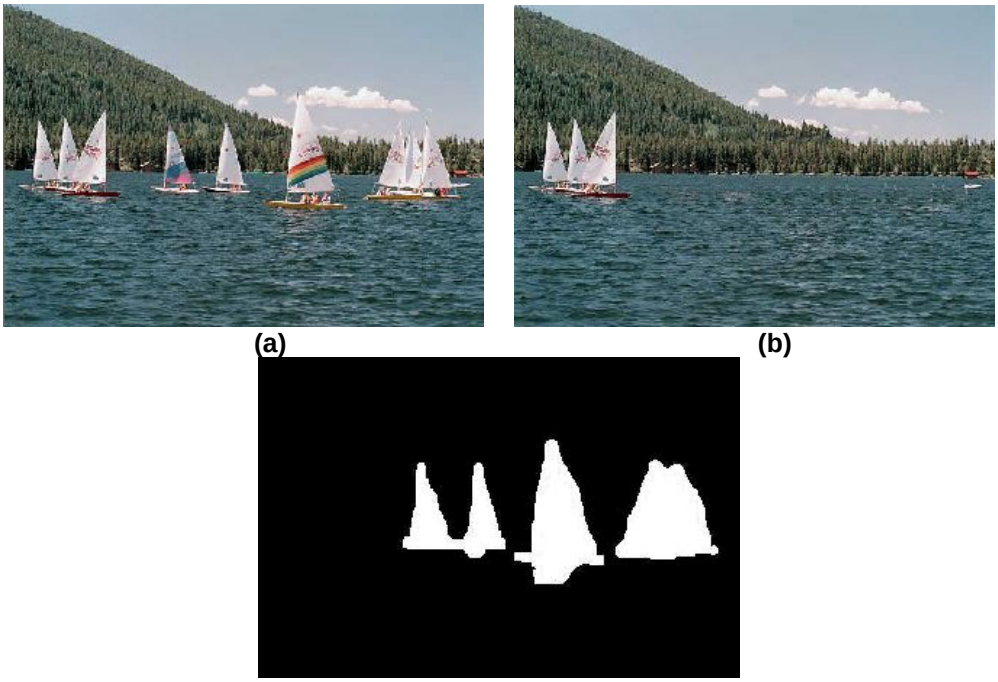


Figure 5. (a) The original scene (b) Result of applying the algorithm to remove some of the boats (c) Matte

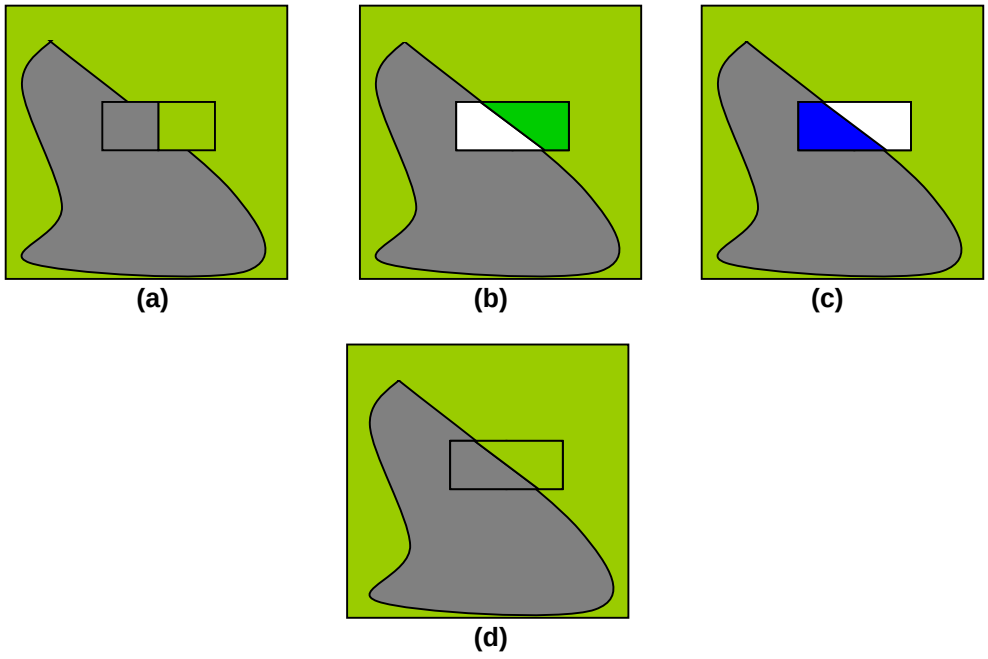
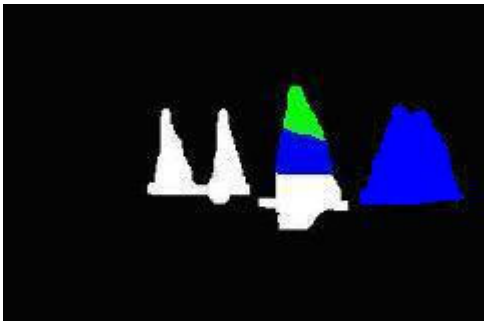


Figure 6. Two Colored Matte. (a) Original fill algorithm (b) Colored Mattes (c) Final result by using Color Mattes

Figure 6. Tilted Orientations

(a) **(b)**

Figure 7. Results using colored Matte



(c)

Figure 8. Colored Matte

4. Computational requirements

In this work, we have shown that even by using a restricted area of the source image, the matte area can be filled in to blend with the background texture. In this section we make a comparison of computational requirement of our method vis-à-vis with the approach outlined in [Drori03] . The authors [Drori03] fill the image matte by applying extensive search over all positions, and along eight orientations in the image. Also, it considers the texture frequency of the image to decide the size of the fragment. Our algorithm is orders of magnitude faster than [Drori03]. Our search is limited to a very small and restricted area in the image. Our algorithm takes advantage of the fact that all the natural images have horizontal distribution of texture and color. So the search is made over a smaller region, e.g. along a horizontal strip next to the missing region.

4.1 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

$$Y^{l}_{t+1} = (Y^{l}_{t}\alpha + C)(*K_{\varepsilon} \downarrow)^{l}(*K_{\varepsilon} \uparrow)^{l}$$

where,

l – Current level t – Iteration

Y – Approximation output

C – Image K_e – Kernel

α – inverse matte

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.

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

where, g is a Gaussian kernel and N is the neighborhood of the pixel. 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, $ß_i$ is 1.

• Calculation of the level set.

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

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 –

$$r = \arg\min \sum_{s=Sr(i),t=T(i),i \in N} \left(d(s,t)\beta_s\beta_t + (\beta_t - \beta_s)\beta_t\right)$$

where, s and t are source and target fragments respectively. 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. 8. n^2 (1 + ¼ + 1/16 + 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.

4.1.1 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 + ($11648mn^2 / 64$) + $3mn^2$) × 4}

For a typical image of size 256 × 256, a typical matte of 50 × 75 and a neighborhood of 3 × 3, the total number of operations (at the highest level) turn out to be 440158704, i.e around 4.4×10^8.

4.2. Computations in our method

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 nb in the grid. (m is the size of the block in the grid)

$n_b = n^2 / m$

• 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 *size* = (nRows*nCols). nRows is the number of rows in the search strip and nCols is the number of blocks in each row in the search strip. nCols does not exceed half the number of blocks in a row in the grid.

Typically, nRows = 3 and nCols = n / (2m). Therefore size of a search strip, *size* = 3n / (2m).

• Distance calculation using the $\mathbf{L}_2$ norm

$$d_k(I_{target}, I_{source}) = \sqrt{\sum_{neighborhood} \begin{pmatrix} (p_s^{red} - p_t^{red})^2 + \\ (p_s^{green} - p_t^{green})^2 + \\ (p_s^{blue} - p_t^{blue})^2 \end{pmatrix}}$$

where subscripts 's' and 't' refer to source and target areas respectively and p_s and p_t are pixels in the source and target fragments.

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.

• Thus the computations involved for all the blocks in the matte
= $9m \times size \times M/m$.
= $9m\ (3n / (2m))\ (M/m)$
= $27nM / 2m^2$

• 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.

4.2.1 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 256×256, a typical matte of 50×75 and a neighborhood of 3×3, the total number of operations turn out to be = 167282 = 1.6×10^5.

4.3 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 reduces considerably the computation time of our algorithm. As can be seen from the above analysis, the time taken by the other algorithm [Drori03] is much more than the time taken by our algorithm. The authors [Drori03] indicate that the typical computation time ranges between 120 and 419 seconds for 192×128 sized images and between 83 and 158 minutes for 384×256 sized images on a 2.4 GHz PC processor. In our case, the computation time for images of 332×223 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].

5. Conclusions

A fast method has been proposed to complete images of natural scenery from which a foreground object has been removed and a matte created. The matte could also be artificially created depending on the needs of the user. We have shown that the size of the search areas of the known regions can be drastically reduced by making use of strong horizontal orientation of the image. Results have been presented to illustrate the idea. Computational comparison with another existing algorithm has been proposed to show that image completion can be carried out quite fast.

References

[Ashikhmin01] Ashikhmin, M. 2001. Synthesizing natural textures. In ACM Symposium on Interactive 3D Graphics, 217–226.

[Bertalmio00] Bertalmio, M., Sapiro, G., Caselles, V., and Ballester, C. 2000. Image inpainting. In Proceedings of ACM SIGGRAPH 2000, ACM Press, 417–424.

[Bertalmio03] Bertalmio, M., Vese, L., Sapiro, G., and Osher, S. 2003. Simultaneous structure and texture image inpainting. In IEEE Conference on Computer Vision and Pattern Recognition, to appear.

[Brooks02] Brooks, S., and Dodgson, N. 2002. Self-similarity based texture editing. ACM Transactions on Graphics, 21, 3, 653–656.

[Drori03] Drori, I., Cohen-Or, D., Yeshurun, H. 2003. Fragment Based Image Completion. In Proceedings of ACM SIGGRAPH 2003, ACM Press.

[Efros99] Efros, A., and Leung, T. 1999. Texture synthesis by non-parametric sampling. In IEEE

International Conference on Computer Vision, 1033–1038.

[freefoto] www.freefoto.com – FreeFoto.com is a collection of free photographs for private non-commercial use on the Internet.

[Freeman02] Freeman, W. T., Jones, T. R., and Pasztor, E. 2002. Example-based super-resolution. IEEE Computer Graphics and Applications, 56–65.

[Heeger95] Heeger, D. J., and Bergen, J. R. 1995. Pyramid-based texture analysis/synthesis. In Proceedings of ACM SIGGRAPH 95, ACM Press, 229–238.

[Igehy97] Igehy, H., and Pereira, L. 1997. Image replacement through texture synthesis. In IEEE International conference on Image Processing, vol. 3, 186–189.

[mountainlake] www.mountainlake.com

[Wei00] Wei, L. Y., and Levoy, M. 2000. Fast texture synthesis using tree structured vector quantization. In Proceedings of ACM SIGGRAPH 2000, ACM Press, 479–488.

[Welsh02] Welsh, T., Ashikhmin, M., and Mueller, K. 2002. Transferring color to greyscale images. ACM Transactions on Graphics, 21, 3, 277–280.