EE264 Project : Simultaneous Structure and Texture Image Inpainting

Shane Brennan
EE 264 - Spring 2007
shanerb@soe.ucsc.edu

June 6, 2007

Abstract

Image inpainting is an old technique for filling in missing or damaged regions of an image. The general problem is: given an image and a user-defined region of the image to be filled, fill the region in such a manner that one cannot tell that the region has been altered. In this project I have implemented the method proposed by Bertalmio et al in their 2003 paper "Simultaneous Structure and Texture Image Inpainting" to accomplish this task automatically on digital images.

1 Traditional Image Inpainting

Image inpainting is a method of modifying images in such a manner that one cannot detect the modification to the image. A typical application of inpainting is the restoration of old and decaying paintings. In this scenario, cracks or other defects have appeared in the work and one wishes to restore the piece so that there are no visible defects. An example of such restoration can be seen in Figure 1. To perform this restoration one has to paint over the cracks in order to hide their existence. If a photograph or some other copy of the original, undamaged artwork exists that can be used as a reference it is a relatively straightforward, if not tedious, task to painstakingly recreate the original image by using the reference image. However, it is often the case that a reference image is unavailable. In this case it is up to the restorator to fill in the damaged regions on their own. A common approach for accomplishing this inpainting task is to fill in the damaged regions using information from the surrounding areas. This includes extending lines and curves entering the damaged regions, as well as filling in the regions with the color and texture information from the surrounding area.

A second example of when image inpainting is needed is when one wishes to remove an undesired object or individual from an image. A classic example of such a scenario are the images which the dictator Joseph Stalin had altered in order to remove political enemies from photographs,

Figure 1: On the left is the damaged image. On the right is the restored version. Note that cracks in the image have been believably filled in. Image courtesy of www.topcstudio.com.

Figure 2: On the left is the original photograph, on the right is the same photograph with Nikolai Yezhov removed. Image obtained from http://www.newseum.org/berlinwall/commissar_vanishes/index.htm.

as can be seen in Figure 2. A more benign example of this type of object removal would be removing blemishes from the face of an individual in order to enhance their apparent beauty, or removing people swimming in the ocean in a photograph of a couple walking along a beach at sunset, in order to make the image have a more romantic appeal.

1.1 Digital Image Inpainting

Digital image inpainting refers to the process of performing inpainting on digitized images. The applications of this procedure are similar to those of traditional image inpainting. It may be the case that a digital image needs to be restored due to damage from the loss of data in a storage medium, or due to lost information during compression or transmission of the image. The desire to remove unwanted objects or individuals from a digital photograph is also commonplace. Therefore, a technique for performing digital image inpainting is needed.

One method of performing digital image inpainting would be by using an image editing tool, manually filling in the regions to be inpainted in a method very similar to what would be done with a physical painting. However, what is really desired is an automatic algorithm which can fill in a region dictated by the user. The end goal of digital image inpainting would be to have the user provide the algorithm with the region to be inpainted, which shall be referred to as the mask, and then to have the computer automatically fill in this region in a manner such that the resulting image looks natural and unaltered. In this paper I shall introduce and discuss a method of achieving this goal, as proposed originally by Marcelo Bertalmio, Luminita Vese, Guillermo Sapiro, and Stanley Osher in their 2003 paper "Simultaneous Structure and Texture Image Inpainting." [7]

1.2 The Proposed Method

The method presented in Bertalmio et al's paper does not contribute any new algorithms to the field of image inpainting. Instead, the authors observe that current inpainting algorithms fail to adequately reconstruct regions that contain texture, and texture synthesis algorithms fail to properly synthesize regions which contain structure. The authors therefore propose to solve the inpainting problem by decomposing an image into two sub-images, one which contains all the texture contained in the image but no structure, and an image which contains the structure of the image but no texture. This can be achieved using a technique proposed by Vese and Osher in 2003 [18].

The image inpainting method proposed by Bertalmio et al is the following:

1. Decompose the input image I_{in} into two sub-images: U, the structure image, and V, the texture image
2. Fill in U using image an image inpainting algorithm
3. Fill in V using a texture synthesis algorithm
4. Recombine the reconstructed U and V image to form output image I_{out}

In the rest of this document I will discuss each of the steps listed above in further detail. For each, I will explain the algorithm used to accomplish the goal and provide some results. In addition, I will discuss conditions under which the techniques will fail.

1.3 Related Work

Automatic digital image inpainting first came into the field of image processing in Bertalmio and Sapiro's 2000 paper "Image Inpainting" [6]. Since that time there have been a number of algorithms proposed to solve the inpainting problem. Betalmio and Sapiro have been the primary researchers in this field, publishing a number of papers since their initial paper in 2000. Previous work in automatic digital image inpainting can be found in [4], [5], and [9]. While these algorithms all work well in images that are relatively smooth and do not contain too much noise or texture, they are unable to fill in regions that are highly textured or contain too much noise.

A topic of study related to image inpainting is texture synthesis. In this field the goal is: given a small image patch containing texture, create a larger region having a visually similar texture. This procedure can be seen as a type of inpainting since if a region of an image needs to be inpainted and one has a sample of the texture present in the region a coarse inpainting solution can be obtained by filling the region to be inpainted using a texture synthesis algorithm. However, this method of inpainting is unable to maintain the structure of an image such as region boundaries.

2 Image Decomposition

Knowing that texture synthesis algorithms exist to accurately fill in regions of missing texture, and image inpainting algorithms exist to fill in regions of missing image structure, a method is desired of decomposing a given image into two sub-images. One sub-image will be a structure image which will be a cartoon-like version of the input image where large-scale edges are preserved but interior regions are smoothed. The other sub-image will be a texture image which will contain all of the texture information of an image, including noise. These sub-images can then be reconstructed using image inpainting and texture synthesis techniques that would have failed on the original image.

In 2002 Stanley Osher and Luminate Vese of UCLA published a paper entitled "Modeling Textures with Total Variation Minimization and Oscillating Patterns in Image Processing" [18]. This paper sought to solve the problem of separating a given image into a structure image and a texture image. The basic model used in the paper is: $f = u + v$ where f is the input image, u is the structural image, and v is the texture image. In this model, having the structure and texture images allows one to exactly reconstruct the original image. In practice this is not the case and one can only approximately reconstruct the original image, though the algoirthm presented by the authors provides results with a very small reconstruction error. The end goal of the deconstruction method is to have a very smooth image u which preserves all the dominant edges in an image but is smooth on interior regions, and an image v which contains all the texture in an image as well

as the noise. These images will then be fed into an inpainting algorithm and a texture synthesis algorithm respectively. The output of those algorithms can be recombined to obtain a final result.

2.1 Derivation of the Sub-Images

The method used to construct the structure image u is based on considering u to be a 2D function and trying to minimize this function in the space of all functions of bounded variation (denoted BV in this document). Functions in BV space are functions whose total variation are bounded by some constant value less than infinity. Searching to minimize u in the BV space ensures that the resulting image is stable and does not blow up to infinity at any point. It should be noted however that this space allows for functions which have very large (though non-infinite) derivatives, thus ensuring that edges can be preserved.

Keeping in mind the intuition from above, the minimization problem must logically have two terms. One term will be a data fidelity term that will seek to keep the difference between f and u small. This fidelity term will ensure that the data from the input image is maintained in the result. The second term will enforce a smoothness over u, though not necessarily at every location within u. The authors initially used the minimization presented by Rudin, Osher, and Fatemi [16] which is computed as:

$$\underset{u \in BV}{\overset{infimum}{}} \quad F(u) = \int |\nabla u| + \lambda \int |f - u|^2 \, dxdy$$

In the above equation, the second term is the data term, the first term is a regularization term to ensure a relatively smooth image, and λ is a tuning parameter. As can be seen, this only seeks to find the optimal u, and ignores the v image. The reason for this is that in previous work the authors had considered the v image to be noise, and therefore to be discarded.

It has been shown in [1], [11], [17], and [2] that there exists a unique result to this optimization problem, and methods exist for finding the solution. Noting that $v = f - u$ it is possible to easily modify the above equation to incorporate v:

$$\underset{u \in BV}{\overset{infimum}{}} \quad F(u) = \int |\nabla u| + \lambda \int \|v\|^2 \, dxdy$$

which yields the Euler-Lagrange equation: $u = f + \frac{1}{2\lambda} div(\frac{\nabla u}{|\nabla u|})$

Solving for v we have: $v = f$ - $u = -\frac{1}{2\lambda} div(\frac{\nabla u}{|\nabla u|})$

At this point it is useful to break v into its x and y components respectively. We will denote these as g_1 and g_2, where:
$g_1 = -\frac{1}{2\lambda} div(\frac{\nabla u_x}{|\nabla u|})$
$g_2 = -\frac{1}{2\lambda} div(\frac{\nabla u_y}{|\nabla u|})$

Figure 3: From left to right: the original image, the structure image, and the texture image.

This allows us to write v as: $v = div\ \vec{g}$ where $\vec{g} = (g_1, g_2)$. It can be seen that $g_1^2 + g_2^2 = \frac{1}{2\lambda}$, so that $\left\| \sqrt{g_1^2 + g_2^2} \right\| = \frac{1}{2\lambda}$. This allows us to rewrite v as:
$v(x, y) = div\ \vec{g} = \partial_x g_1(x, y) + \partial_y g_2(x, y)$

This now leads us to the final minimization problem:

$$\underset{u \in BV}{infimum} \quad G(u, g_1, g_2) = \int |\nabla u| + \lambda \int |f - u - \partial_x g_1 - \partial_y g_2|^2 dxdy + \mu \left[\int \sqrt{g_1^2 + g_w^2}\, dxdy \right]$$

Solving the above minimization problem yields the Euler-Lagrange equations:
$u = f - \partial_x g_1 - \partial_y g_2 + \frac{1}{2\lambda} div(\nabla u / |\nabla u|)$
$\mu \frac{g_1}{\sqrt{g_1^2 + g_2^2}} = 2\lambda \left[\frac{\partial}{\partial x}(u - f) + \partial_{xx}^2 g_1 + \partial_{xy}^2 g_2 \right]$
$\mu \frac{g_2}{\sqrt{g_1^2 + g_2^2}} = 2\lambda \left[\frac{\partial}{\partial y}(u - f) + \partial_{xy}^2 g_1 + \partial_{yy}^2 g_2 \right]$

2.2 The Discrete Algorithm

Discretization of the image decomposition algorithm is accomplished using an iterative approach. Details on the discretization can be found in [3] and [16]. The values used to initialize u, g_1, and g_2 are: $u_0 = f \quad g_1 = -\frac{1}{2\lambda} \quad g_2 = -\frac{1}{2\lambda}$. In the interest of space I refer you to [18] for specifics on the update equations, though it should be mentioned that there exists another tuning parameter μ in the equations which can be tuned, in addition to λ, to obtain different levels of smoothness in u.

2.3 Image Decomposition Results

Figure 4 shows the results of the decomposition algorithm on the classic "lena" image. As can be seen, the structure image is a smoothed version of the original image, and the texture has been preserved in the texture image.

Figure 4: From left to right: the original image, the structure image, and the texture image.

Figure 4 shows the results of the decomposition algorithm on the "leg" image provided by Bertalmio on his website. As can be seen, the texture has been removed in the structure image, but preserved in the texture image. Note that the texture images have been rescaled to allow for easier display.

3 Image Inpainting

Once a structural image is obtained using the decomposition method described in the previous section the next step is to perform image inpainting on the structure image. Any inpainting algorithm can be used to accomplish this task, but Bertalmio et al propose to use an inpainting algorithm developed by themselves several years previously in a paper entitled "Image Inpainting" published in 2000 [6], however the inpainting method used could be any of the methods described in section 1.3.

Let Ω represent the region to be inpainted, and $\partial\Omega$ be the boundary of the region to be inpainted. The basic idea of the inpainting algorithm in [6] is to find the isophate lines arriving at Ω and continue those lines into $\partial\Omega$. An isophate line is a line between adjacent sections in the image which have nearly constant intensities within each region, but different intensities across the regions. For the sections of $\partial\Omega$ where no lines arrive at the nearest $\partial\Omega$, in other words a region of constant color arrives at $\partial\Omega$, it suffices to simply propogate the color of the constant region into Ω. Once this has been done for every pixel on $\partial\Omega$ we can then erode Ω by 1 pixel and repeat the entire approach. In this manner the algorithm "eats away" at Ω, eventually filling it entirely. This idea of propagating information from $\partial\Omega$ into Ω can be encapsulated by the iterative procedure:

$$I_{i,j}^{n+1} = I_{i,j}^{n} + \Delta t I_{t,i,j}^{n} \quad \forall (i,j) \in \Omega$$

At each "time" in the inpainting algorithm the image is updated and moved closer to the desired result, with Δt being a small value to ensure convergence. Looking at the problem in this way, it can be seen that I^0 is the input image and I^∞ is the desired inpainted image. However, since infinite time is not available the algorithm runs until I^{n+1} is within some small threshold of I^n. If desired we can set the ending condition to be $I^{n+1} = I^n$, though convergence is not guaranteed in this case.

3.1 Information Propogation

Let $L_{i,j}$ represent the information that we wish to propagate into Ω, and $\vec{N}_{i,j}$ represent the direction that we wish to propagate the information. Given these two pieces of information, a logical choice for the value of the current pixel being inpainted would be:

$$I^n_{t,i,j} = \delta\vec{L}_{i,j} \cdot \vec{N}_{i,j}$$

where $\delta\vec{L}_{i,j}$ is a measure of the change of L at position (i,j), which can be as simple as the gradient of L at (i,j). In other words, we find the information at the boundary of Ω and project that information onto the boundary. Given this description all that is left is to describe L and $\vec{N}$ using information available in the image.

To obtain a natural and believable inpainting result we need the information propagated into Ω to be smooth. This will ensure that the result of the inpainting isn't noisy or jagged. While this constraint holds true in most cases it would not perform well on regions that are naturally not smooth. This is the cause of the algorithms failure on regions that contain fine texture or noise and led the authors to take the approach of decomposing the image into separate structure and texture images. By ensuring the input to the inpainting algorithm is smooth and does not contain texture we can be reasonably assured that the result will be good since our assumption of image smoothness constraint will hold true. Knowing that the information to be propogated should be smooth we can define $L_{i,j}$ to be an estimation of the smoothness of I^n at location (i,j). The authors use the laplacian operator to compute L since the laplacian is a well-known estimator of image smoothness.

$\vec{N}$ is meant to represent the direction that the information L needs to be propogated into. Therefore $\vec{N}$ needs to be tangent to the isophate line arriving at $\partial\Omega$. Letting I_x and I_y be the x and y gradients of I^n respectively, the tangent to the isophate line (which is itself an image brightness edge) can be estimated as the vector $[-I_y, I_x]$. Note that I will use the coordinate convention of (x,y) in this paper.

Putting all the above ideas together yields the image inpainting updating scheme as:

$$I^{n+1}_{i,j} = I^n_{i,j} + \Delta t \left(\delta\vec{L}_{i,j} \cdot \vec{N}_{i,j}\right) \left|\nabla I^n_{i,j}\right|$$

where $\left|\nabla I^n_{i,j}\right|$ is a slope-limited version of the norm of the gradient which is added in order to provide stability. The slope-limited norm ensures that extremely large values will not be obtained for $I^n_{t,i,j}$ and will keep the resulting image from "blowing up." The authors compute the slope-limited norm as:

$$\left|\nabla I^n_{i,j}\right| = \begin{cases} \sqrt{\left(I^n_{xbm}\right)^2 + \left(I^n_{xfM}\right)^2 + \left(I^n_{ybm}\right)^2 + \left(I^n_{yfM}\right)^2} & \text{if } \left(\delta\vec{L}_{i,j} \cdot \vec{N}_{i,j}\right) > 0 \\ \sqrt{\left(I^n_{xbM}\right)^2 + \left(I^n_{xfm}\right)^2 + \left(I^n_{ybM}\right)^2 + \left(I^n_{yfm}\right)^2} & \text{if } \left(\delta\vec{L}_{i,j} \cdot \vec{N}_{i,j}\right) < 0 \end{cases}$$

where b and f represent backward and forward differences respectively, and m and M represent the minimum and maximum of the value with 0 respectively.

3.2 Anisotropic Diffusion

A failing of the image inpainting approach described above is that since only pixels adjacent to the pixel being inpainted are measured and used to compute the current pixels intensity it is possible that a small scale fluctuation in the gradient field caused by texture or image noise can be interpreted as a large-scale isophate line. Therefore a method is needed to smooth out these small scale fluctuations while maintaining large scale edges. The authors propose to interweave an anisotropic diffusion stage after every few iterations of inpainting in order to smooth out any intermediate artifacts created by the inpainting algorithm that would otherwise cause the algorithm to converge to an incorrect result.

The goal of anisotropic diffusion is to perform a smoothing operation that varies spatially, as opposed to isotropic diffusions such as gaussian, median, or average blurring. Theoretically, any anisotropic diffusion can be used so long as it removes small scale fluctuations caused by texture and image noise but maintains large scale edges such as isophate lines. However, as will be discussed in section 3.3 and seen in section 5.1, the image inapinting algorithms success is extremely dependent on the specifics of the anisotropic diffusion used. In the inpainting algorithm this diffusion allows regions that contain no edges to "bleed" into Ω. In the equations from the previous section the inpainting equation centered around the term $\delta\vec{L}_{i,j} \cdot \vec{N}_{i,j}$, where $\vec{N}_{i,j}$ is defined as $[-I_y, I_x]$. Note that in regions of constant intensity $\vec{N}_{i,j}$ is the zero vector. Despite this we still wish to propogate this constant color into Ω. Performing anisotropic diffusion accomplishes this goal by "smearing" the region of constant color into Ω. The authors propose an anisotropic diffusion of the form:

$$\frac{\partial I}{\partial t}(x,y) = g_x(x,y)\kappa(x,y)\left|\nabla I(x,y)\right| \quad \forall (i,j) \in \Omega^\epsilon$$

where Ω^ϵ represents all pixels in Ω in addition to the pixels within 3 pixels of Ω.

The specifics of the above anisotropic diffusion as used in my implementation are:

$$I_{i,j}^{n+1} = I_{i,j}^n + \Delta t \left(I_{xx,i,j}I_{y,i,j}^2 - 2I_{xy,i,j}I_{x,i,j}I_{y,i,j} + I_{yy,i,j}I_{x,i,j}^2\right) / \left(I_{x,i,j}^2 + I_{y,i,j}^2\right)$$

where I_x and I_{xx} represent the first and second derivatives of I in the x direction, and similarly for I_y and I_{yy}. I_{xy} is the "cross laplacian" defined as:

$$I_{xy,i,j} = I_{i+1,j+1} + I_{i-1,j-1} + I_{i+1,j-1} + I_{i-1,j+1} - 4I_{i,j}$$

Credit and thanks goes to Antonin Stefanutti for providing me with the the code and understanding to implement this anisotropic diffusion.

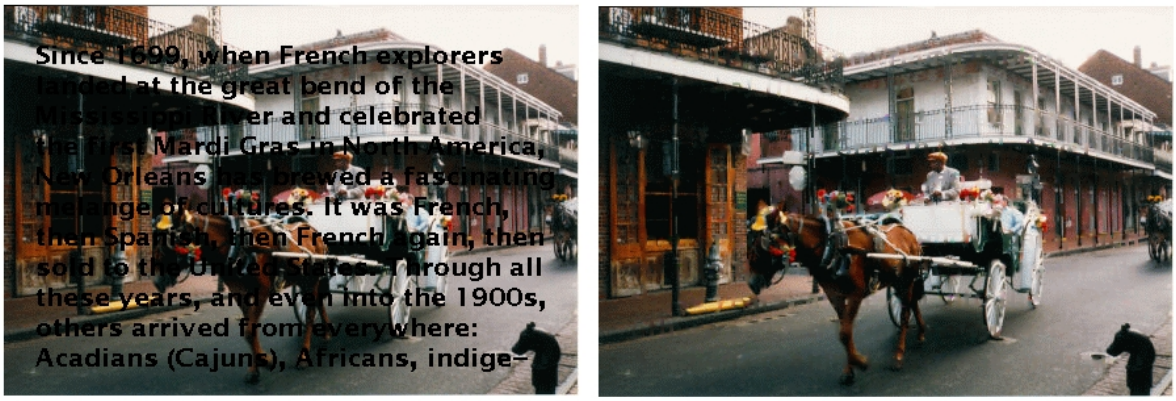

Figure 5: On the left is the original image, with the black text being the region to be inpainted. On the right is the result of the image inpainting algorithm.

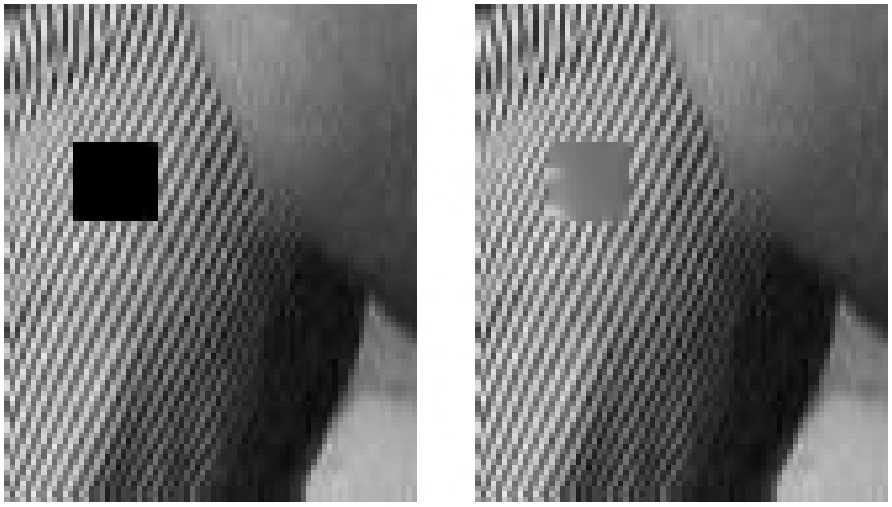

Figure 6: On the left is the original image, with the black box being the region to be inpainted. On the right is the result of the image inpainting algorithm. Notice that the algorithm failed to reconstruct this textured region.

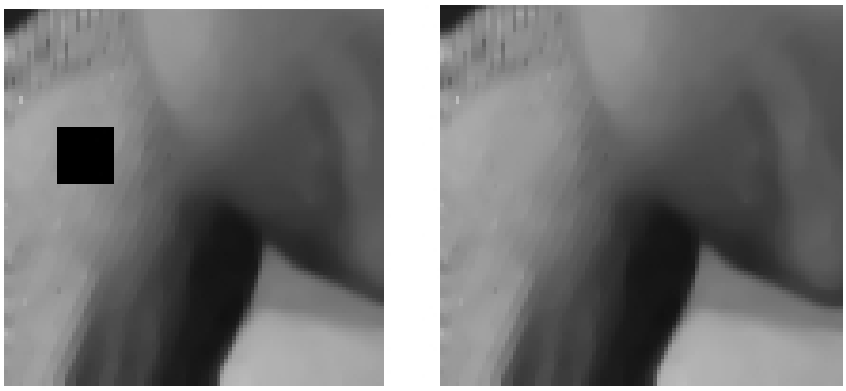

Figure 7: On the left is the structure image, with the black box being the region to be inpainted. On the right is the result of the image inpainting algorithm. Notice that now the region is properly inpainted.

3.3 Image Inpainting Results

Figure 5 shows a typical application of the inpainting algorithm. The results seem pretty good, but the failure of the algorithm on regions of texture can be seen in Figure 6. Of real interest are the results of the inpainting algorithm on the structure images produced by the decomposition algorithm in section 2. As can be seen in Figure 7 the algorithm is able to sucessfully inpaint regions of the structure image that it was not able to inpaint in the original image, demonstrating the need and utility of the image decomposition scheme.

In addition to failings in the algorithm itself there are failures in my implementation of this algorithm. My implementation is unable to obtain results comparable to those given by the authors on all images. This will be shown more in section 5.1. I spent many hours trying to remedy this problem, which I narrowed down to an issue in the specifics of the anisotropic diffusion used. I was able to find implementations of both anisotropic diffusions mentioned by the authors [13] [10]. However, using these versions of anisotropic diffusion did not lead to results comparable with those of the authors. I also tried another anisotropic diffusion as mentioned in section 3.2 in addition to a basic laplacian smoothing. Incidentally, the diffusion used in 3.2 and the laplacian smoothing worked best. In browsing the results of this algorithm as implemented by others I found that the algorithms sensitivity to the anisotropic diffusion used is a major issue for everyone who implements this algorithm. A result common among the implementations I have found, including my implementation, is that the algorithms performs well on text and thin cracks such as in Figure 5, but fails in when the region to be inpainted is too large.

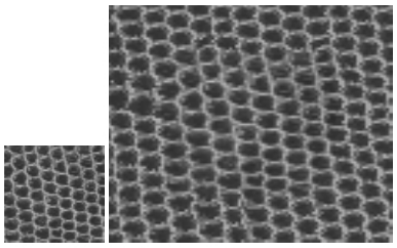

Figure 8: On the left is the example texture path. On the right is the larger image patch synthesized using the example texture patch.

4 Texture Synthesis

In texture synthesis the problem is that given a small patch containing texture we wish to fill a much larger region with a texture that is visually similar to the texture patch. From a probabilistic viewpoint the problem is that given a region that has some stationary distribution we wish to synthesize additional samples from this distribution. A straightforward example of this problem is given in Figure 8.

The texture synthesis algorithm used by Bertalmio et al [7] on the texture images is the algorithm presented by Efros and Leung in a 1999 paper titled "Texture Synthesis by Non-parametric Sampling" [12]. While this method is not state-of-the art it is considered a simple, quick, and effective algorithm for solving the texture synthesis problem. Additional texture synthesis algorithms can be found in [8], [14] and [15].

4.1 The Texture Synthesis Algorithm

The proposed approach synthesizes texture at the pixel level as opposed to a block-based or holistic approach. The algorithm works by taking a region which has already been partly synthesized, and a pixel which has yet to be synthesized, and synthesizing the appropriate intensity value to fill the pixel with. The method used to synthesize an appropriate intensity value is to find a pixel in the image whose neighborhood is similar to the neighborhood around the pixel to be synthesized. The algorithm then keeps either the K most similar regions or the regions whose distance (using any distance metric) to the region around the pixel to be synthesized are below some threshold T as candidate regions, where K and T are parameters that can be set by the user. These intensity value of the pixels at the center of these candidate regions can be considered a probability distribution that can be sampled from to obtain an intensity value for the pixel to by synthesized. Psuedo-code for the proposed algorithm is as follows:

1. For each pixel to be filled:
2. Define the WxW neighborhood around the current pixel as the target neighborhood, where W is a user-defined parameter

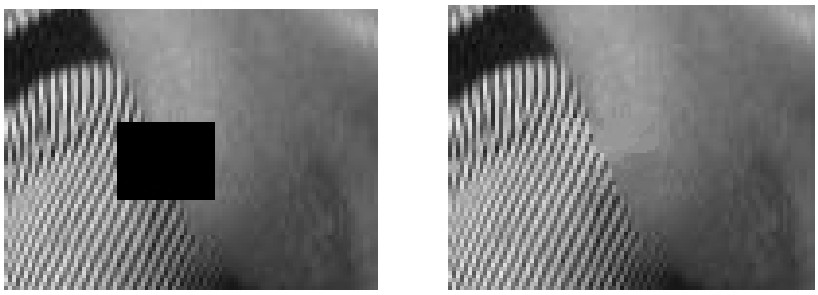

Figure 9: On the left is the damaged image, with the black box being the region to be synthesized. On the right is the image with the damaged region filled in. Note that the algorithm accurately reconstructs the textured region, but doesn't perform well on the relatively untextured region.

3. Define a WxW binary image which is true on pixels in the target neighborhood which have already been filled, and false elsewhere, call this the target mask
4. Collect many samples from the rest of the image, call these the candidate regions
5. For each candidate region compare it to the target neighborhood using a distance metric using only pixels that lie on the regions of the target mask which are true
6. Keep any regions whose distance is below the user-defined threshold T
7. Randomly select one of the remaining candidate regions
8. Fill in the current pixel with the center pixel of the selected candidate region

Larger values of the parameter W will ensure that the synthesized texture maintains larger-scale texture similarity to the sample image patch, while smaller values of W will not maintain large-scale textures and instead represent more fine-grained texture detail, and in the extreme case of $W = 3$ the resulting synthesized texture is close to random.

4.2 Texture Synthesis Results

A typical use of the texture synthesis algorithm can be seen in Figure 9. However, as can be observed in Figure 10 the texture synthesis technique fails in regions which contain structure, as the boundary lines fail to be preserved properly.

Of real interest are the results of the texture synthesis algorithm on texture images produced by the image decomposition technique described in section 2. Results of the algorithm on texture images produced by the decomposition algorithm are very encouraging and can be seen in Figure 11.

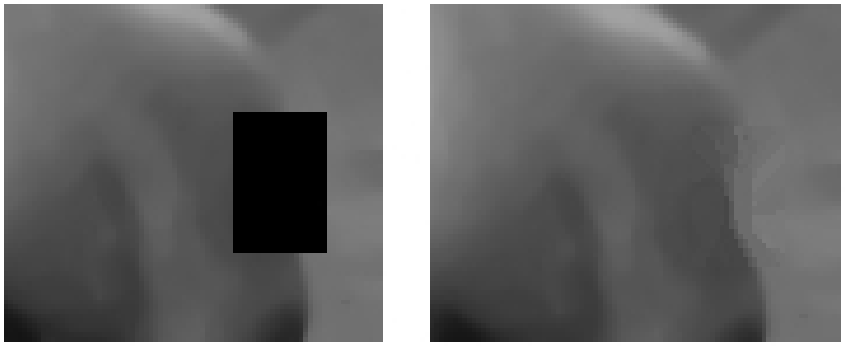

Figure 10: On the left is the damaged image, with the black box being the region to be synthesized. On the right is the image with the damaged region filled in. Note that the algorithm does not believably reconstruct the structure of the object, this shows that texture synthesis alone is inadaquate as an inpainting method.

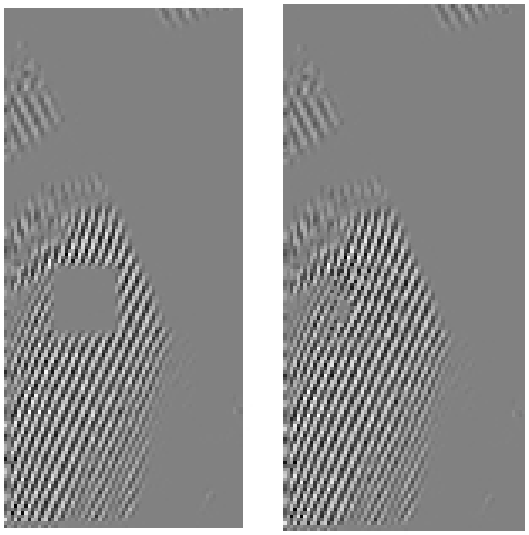

Figure 11: On the left is the example texture path. On the right is the larger image patch synthesized using the example texture patch.

Figure 12: The top row contains the original image and the sub-images. The bottom row contains the reconstructed versions of each image. Note that this image contains very little texture. This is likely the reason why this image was used as an example in [6].

5 Recombining the Texture and Structure Images

As mentioned in section 2 the structure and texture images can be added together to recreate the original image. The same holds true of the structure and texture images after image inpainting and texture synthesis have been performed. Therefore, once the structure image has been inpainted and the texture image has been filled using texture synthesis, the final output of the simultaneous structure and inpainting algorithm can be obtained by: $I_{out} = U + V$, where U is the structure image after inpainting has been performed and V is the texture image after texture synthesis has been used to fill the region to be inpainted.

5.1 Results and Conclusions

Results of the simultaneous structure and texture inpainting algorithm can be seen in Figures 12, 14, and 13. As can be seen the results are encouraging but suffer due to the problems in my implementation of the image inpainting algorithm.

In conclusion, while the algorithm performs well on a wide array of images it has its short-

Figure 13: The top row contains the original image and the sub-images. The bottom row contains the reconstructed versions of each image

comings. One shortcoming is the necessity to tune several parameters. Even the tuning of three parameters prevents the algorithm from being useful enough to the average user, inhibiting its ability to be used in a commercial software package such as Adobe Photoshop. In addition, the algorithm is difficult to implement due to the vagueness of the anisotropic diffusion that must be used. As my experience has shown the specifics of the anisotropic diffusion used in the image inpainting algorithm make an enormous difference on the performance of the algorithm. Without knowledge of the specific anisotropic diffusion used by the authors an implementation that has results comparable to theirs proves elusive. In the end, this project was a worthwhile experience and I am glad to have learned much about these fields of image processing of which I was previously ignorant.

References

[1] R. Acar and C. Vogel. Analysis of bounded variation penalty method for ill-posed problems, 1994. 5

[2] F. Andreu, C. Ballester, V. Caselles, and J. Mazón. Minimizing total variation flow, 1998. 5

[3] G. Aubert and L. Vese. A variational method in image recovery. *SIAM Journal on Numerical Analysis*, 34(5):1948–1979, 1997. 6

[4] C. Ballester, M. Bertalmio, V. Caselles, G. Sapiro, and J. Verdera. Filling-in by joint interpolation of vector fields and grey levels, 2000. 4

[5] M. Bertalmio, A. Bertozzi, and G. Sapiro. Navier-stokes, fluid dynamics, and image and video inpainting, 2001. 4

[6] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester. Image inpainting. 2000. 4, 7, 15

[7] M. Bertalmio, L. Vese, G. Sapiro, and S. Osher. Simultaneous structure and texture image inpainting, 2002. 3, 12

[8] J. D. Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. *Computer Graphics*, 31(Annual Conference Series):361–368, 1997. 12

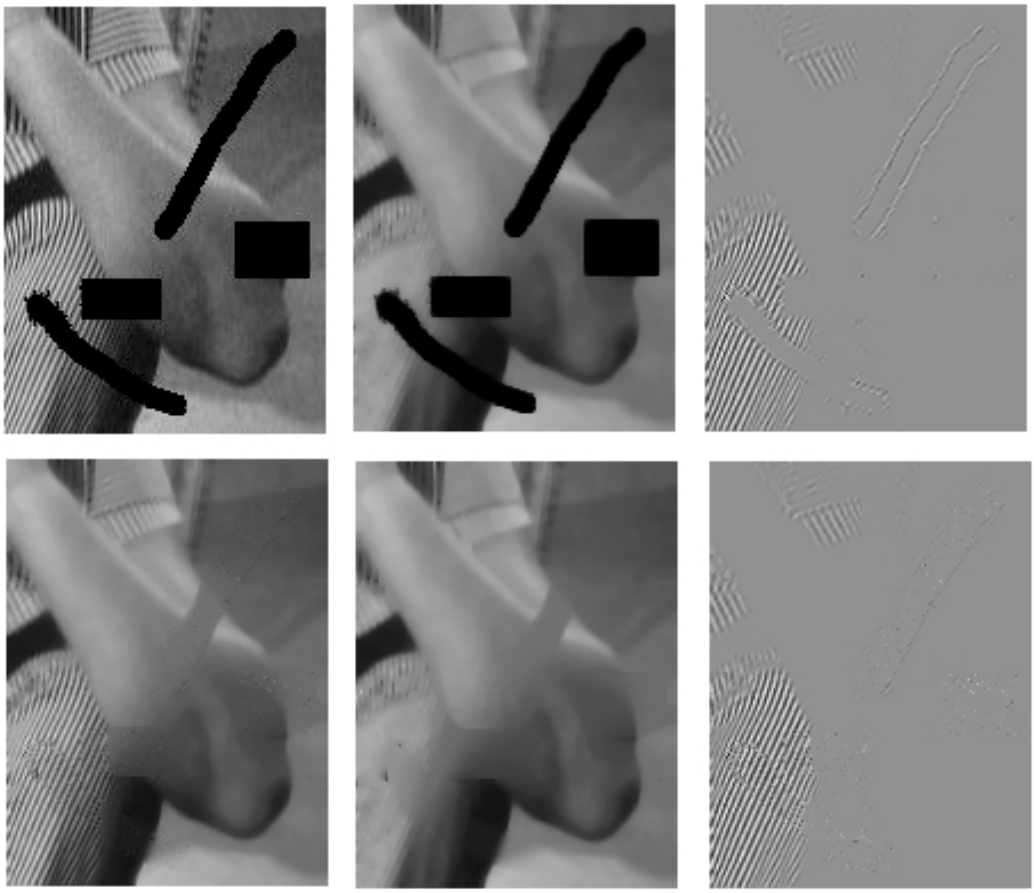

Figure 14: The top row contains the original image and the sub-images. The bottom row contains the reconstructed versions of each image

[9] R. Cant and C. Langensiepen. A multiscale method for automated inpainting, 2003. 4

[10] F. Catte, P. L. Lions, J. M. Morel, and T. Coll. Image selective smoothing and edge detection by nonlinear diffusion. *SIAM J. Numer. Anal.*, 29(1):182–193, 1992. 11

[11] A. Chambolle and P. Lions. Image recovery via total variation minimization and related problems, 1997. 5

[12] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. In *ICCV (2)*, pages 1033–1038, 1999. 12

[13] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. 11

[14] K. Popat and R. Picard. Novel cluster-based probability model for texture synthesis, classification, and compression, 1993. 12

[15] J. Portilla and E. P. Simoncelli. Texture modelling and synthesis using joint statistics of complex wavelet coefficients. In *IEEE Workshop on Statistical and Computational Theories of Vision*, 1999. 12

[16] L. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms, 1992. 5, 6

[17] L. Vese. A study in the bv space of a denoising-deblurring variational problem. 5

[18] L. Vese and S. Osher. Modeling textures with total variation minimization and oscillating patterns in image processing, 2002. 3, 4, 6