

CGDI assignment: patch-based inpainting

Gabriel Dehame

29 avril 2023

1 Context

For this CGDI assignment, I chose to implement an image inpainting method.

Image inpainting consists on given a source image and a mask (black and white), remove the foreground of the pixels in the source image that are white in the mask. There are different kinds of techniques to achieve such goal but they generally use the background of the source image that is in the black part of the mask to replace the white part. This way, these second types of techniques hope to stay close to the real background that is hidden behind the foreground to remove by mimicking the part of the background that isn't hidden.

Among the major kinds of method, I chose to implement a patch-based inpainting method, these types of methods analyze patches in the source region (black in the mask) and copy them onto patches in the target region (white in the mask). This way, they propagate little by little the background on the part to remove and hope to make little errors as they try to fit patches that are close to the surrounding image and thus lead to a credible image. However, an important issue to tackle when thinking about such methods is the order in which we fill the patches in the target region.

The patch-based inpainting I chose to implement uses a notion of priority of the patches to fill that uses an equilibrium between the confidence we have in the pixels surrounding the patch and the strengths of isophotes flowing into the patch. The idea behind this confidence notion is that we completely trust the source region as it's the source, and we do not trust the target region as we want to remove it. Then, as we use the surrounding of a target patch to determine how to replace it, we don't have a complete trust in the patch once it is replaced because it might not actually be the background that the target region was hiding, it simply is a plausible one inspired by the logic of the source background. With this notion of priority, the method I implemented hopes to find an equilibrium between linearity in the structures and breadth first replacement, because replacement in depth pushes toward making mistakes as you don't have much source around the patch once you're too deep in the target.

Then, once the priority is defined, you have to have a way of choosing which patch from the source region you will inpaint in the target patch of top priority, for this the method I chose computes the distance between the target patch and

every patch of the source region taken as the sum of the euclidean distances between the pixels, as 3d vectors, of the patches that are both in the source region and have the same coordinates relatively to the center of their patch. Then, the patch is copied and the algorithm loops to find a new patch of top priority then the source to copy until the target region is filled.

2 Implementation choices

First of all, as mentionned by the title, I chose to implement the patch-based method for image inpainting.

Then, in the article, some parts are unclear or not explained and so I'll detail here how I interpreted it (according to have the best results, but still not as good as those presented in the article).

First of all, the article defines a formula for a so-called "data term" using *isophotes* and *normals* of the frontier between the part to remove and the one to keep but I didn't know how to calculate those. Therefore, I chose for the *normals* to calculate the gradients of the mask image, then normalize it. For the *isophotes*, I guessed by the way it's written (∇I^\perp) that it's the orthognal vector of the gradient, thus I calculated the gradients in both coordinates, g_x, g_y and defined the *isophotes* as $(-g_y, g_x)$ because $-g_y g g_x + g_x g_y = 0$. To calculate those gradients, as there are multiple methods, I chose to use forward differences and if the next pixel doesn't exist (border of the image) I use backward difference.

Now, for the algorithm itself, the article mentions a computation of the priorities of the frontier patches but doesn't detail whether we should at that point recompute the data terms and confidence terms or not, priority being the product of both. I found that more intuitive to recompute them as the updates at the end of the main loop do not affect the data terms but this data term can change with the filling of a patch as new pixels enter the frontier and the gradients can change as we modified the colors of some pixels. I also chose to basically update everything as the frontier obviously changes as well, and is computed based on the *normals*. The algorithm also uses a distance function between patches defining it as the sum of squared differences of pixels in CIE Lab color format. I thus implemented a translation from RGB color format to CIE Lab, without really knowing what this format is neither is the translation I found was true but the results seem ok. However I had some issues with a use case (number 7 in my tests) because many patches had the same distance but some were better than others as in this example, we want to reconstruct a half black, half gray image deleting the green circle but the bottom side of the circle was replaced in some patches by partly black patches therefore the result was unsatisfying. To solve this issue, I added the euclidian distance between the centers to break equality cases, intuitively I thought that the hidden background that we want to reconstruct has more chances to be similar to close pixels than to distant ones. Without this addition, test number 7 that is presented later resulted in the image shown on figure 1



FIGURE 1 – Result obtained on test number 7 with only the SSD distance

3 Experiments

The results of my experiments are showcased in the table 1

Overall, the results are satisfying for the tests 0, 2, 5, 6 and 7, but for the others it's really obvious that the image is digitally altered. Additionnally, the article presents the tests 1, 2, 3, 4 and 5 and all of them look fine (except number 4 which looks a bit strange when we zoom in) but in my results only two of them are good, which is quite disappointing. I feel like I misimplemented something but I really don't see what and I've tested different interpretations of the points I mentionned in the first section and those final ones are the ones giving the best results.

From what I tested, my issue seems to come from the priority of the patches, at least for tests 1 and 3, I'm not sure for test 4. However I'm confident I correctly implemented the algorithm, especially on that part, what I'm a bit less confident on is the distance because of the CIE Lab encoding.

4 Review

Here, as mentioned in the image inpainting subject I will do a small review of the strengths and weaknesses of the method, this will be mainly based on the presented results in the article as I failed to implement it correctly it seems. The subject of the assignment also points out a parameter evaluation but there are no parameters in this method, except maybe the mask but I consider it as an input and the patch size but again the article gives a default value and a method to define the appropriate value, so it's not really a parameter.

4.1 Strengths

Thanks to an exemplar-based method, copying patches from a source region of the image to the target region to fill, this inpainting method avoids any blurring effect that other methods could have on images such as test number 4.

The priority based filling order also allows better results as in test number 2, where previous methods had worst results because the region was filled in a suboptimal order, but this patch-based inpainting and its data term allows to focus primarily on the texture boundaries and create a more realistic texture.



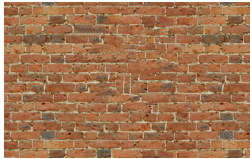












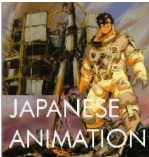





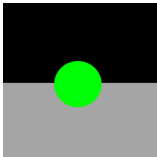
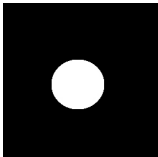

Test number	Input image	Input mask	Output
0			
1			
2			
3			
4			
5			
6			
7			

TABLE 1 – Results of the experiments ran

The tests number 5 and 7 show that this article’s method allow for a good image reconstruction, eliminating the text added on test 5 and the circle on test 7 (done with a torus in the paper). This was already successful with previous inpainting methods but this new one beats the previous when it comes to real photographs, test number 7 is in the paper almost satisfying, see figure 2, even though we see some strange trees on top of the middle of the building as well as a straight line separating them from the actual trees, unlike previous methods it doesn’t add any blurring in the center and thus provides a better result.



FIGURE 2 – Result given in the paper for test number 7

4.2 Weaknesses

As mentioned at the end of the strengths sections, despite handling better than previous methods the test number 7, it remains not great as we can depict by zooming in a bit that the image is a fake. This method still isn’t perfect on real photographs.

A second weakness is that this method is very slow, I cannot really compare with other methods and so I don’t know if relatively speaking it is slow, but in absolute it takes a lot of time as for every patch to fill we have to iterate over every patches of the entire image to find the best patch. For my experiments it generally took approximately 5 minutes for images of at most 200000 pixels, but usual images are at least 1920*1080 which is 10 times more. The execution time obviously depends on the mask too but the mask itself is bigger for bigger images so it’s even worse.

This patch-based inpainting method also tends to introduce irregularities when replacing big complex structures, as it lacks examples in the source region to properly fill it or because it doesn’t manage to reproduce the complexity of a background complex structure when removing the foreground element to fill if this element to fill hides some complex parts. This is what might be happening for test number 7. We can also see some irregularities in the result presented in the paper for test number 0 at the bottom of the picture, see figure 3.

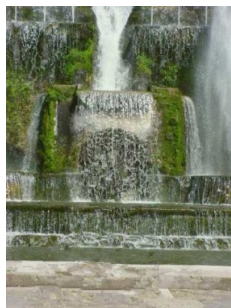


FIGURE 3 – Result given in the paper for test number 0

Another issue is that as the distance between patches is computed by comparing only the already filled pixels, because we know the others are wrong in the sense that they are not what we want to have, we completely ignore the overall structure of the object we’re rebuilding by being too local. In particular, one can see on the result 2 that we extend the grass onto the water at the bottom center of the picture even though the shape of the river probably isn’t actually this, still it remains credible. It can end up worse, for example if we have a blue object which has a bit of green on it and there is grass in the source region then we might put grass on the object which makes no sense.

5 Conclusion

To conclude, I tried to implement a patch-based inpainting method with relatively poor results, I obtain satisfactory results for basic tests but not for the more complex ones that this method pretends to solve in its presentation paper. However, this method generally seems quite satisfactory when we take the results of the paper and not mine, despite still creating images with defaults that make obvious the fact that it’s not an actual picture sometimes. Also, this method is extremely slow.