An Improved Algorithm for Traditional Image Inpainting

Ziji Cui
The Chinese University of Hong Kong, Shenzhen
120030056@link.cuhk.edu.cn

Zhao Songlin
The Chinese University of Hong Kong, Shenzhen
120090346@link.cuhk.edu.cn

Zhen Tong
The Chinese University of Hong Kong, Shenzhen
120090694@link.cuhk.edu.cn

Abstract

What can you do to tackle the unwanted object in your photos? This paper presents an improved algorithm based on the previous strategies. Our algorithm creates a mask for unwanted objects and fills the mask region seamlessly. Our chief insight is that we try to preserve the original image as much as possible by using graph cutting to create the mask. Then the algorithm will use the original image and other similar images to fill the mask region. Unlike the traditional strategy, our algorithm can efficiently provide sound and plausible results. We demonstrate the superiority of our algorithm over existing image completion approaches.

1 Introduction

Image inpainting (also called completion or hole-filling) is the task of filling in or replacing an image region with new image data such that the modification can not be detected. There are two steps to conduct image inpainting: generate the mask for unwanted objects and fill the mask region. For the mask, it is required to cover all unwanted regions while preserving as many wanted regions as possible. Almost all previous work pays less attention to the mask. They use pairs of images and masks in some existing databases[7],[9]. The drawback is that the mask also covers some important information in the image. Besides, it is also not user-friendly. Because, when users want to apply their algorithms to their own images, there are no corresponding mask images to use. For the filling part, the previous algorithms based on template matching can provide plausible results by propagating the texture and structure information in the original input image to fill the blank region. But they suffer from low speed because of pixel-by-pixel iteration[6]. There are also some diffusion-based algorithms that are fast but do not make the generated texture fit with the background.

Therefore, this paper proposes an improved algorithm to tackle the drawbacks. For an input image, this algorithm can provide a more accurate mask to cover unwanted objects based on graph cut. Then, to fill the mask region, the algorithm will first use an exemplar-based method to fill the boundary. After that, users are allowed to input additional similar images. And the algorithm can use these similar images to fill the rest of the mask region by using template matching and Poisson blending.

Paper Report of Final Project for ECE4513 - Digital Image Processing and Computer Vision.

link for our codes: https://github.com/ECE-Img-Inpainting/Examplar-based

1.1 Related Work

Segmentation There are many segmentation algorithms like thresholding, region growing [10] and morphological watersheds [1]. They usually achieve segmentation by clustering contents that have similar intensities. However, they may fail when the object to be segmented has complex color patterns. In our project, we focus on segmenting by dissimilarity between contents in the image.

Image Inpainting A number of algorithms have done on the task of image inpainting, where small range of area like speckles and scratches are removed [3]. These techniques are inspired by the partial differential equations of physical heat flow. They fill holes in images by propagating linear structures (called isophotes in the inpainting literature) into the target region via diffusion. Their drawback is that the diffusion process introduces some blur, which is noticeable when the algorithm is applied to fill larger regions. That is because the isophote propagation method loses texture information[4]. And exemplar-based method used in this paper can fix them.

2 The Proposed Algorithm

2.1 Pipeline

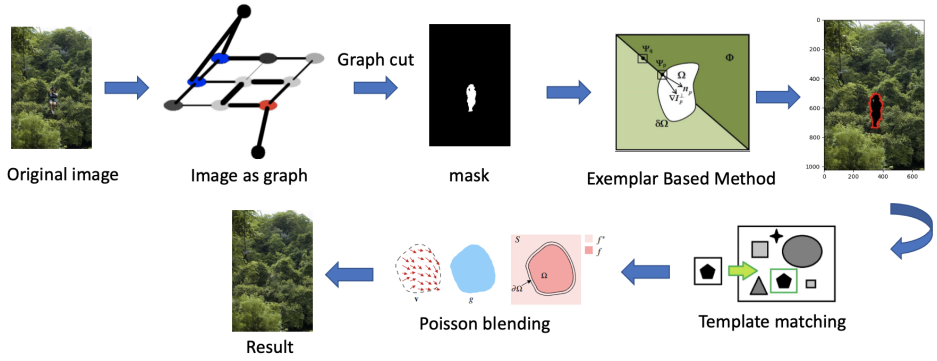

Figure 1: The overall logic of our project

2.2 Segmenting foreground from background

In this section, we implemented the graph cut algorithm. Some prior information is needed: user needs to specify the foreground and background by adding some scribbles to the image. We then represented the image as a graph and tried to find a cut that precisely divides the image into foreground and background segments.

2.2.1 Image as graph

We first convert an image to a network graph, where every pixel is represented by a node. We use F_i and B_i to represent the foreground and background nodes specified by the user. Two additional nodes S and T, called foreground terminal and background terminal, are connected to F_i and B_i respectively.

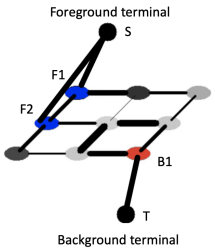

Figure 2: graph defined by the image

There are two kinds of edges in the graph. The first are edges between pixel nodes. They are defined by the similarity of neighboring ($4-$neighbor) pixels $a_{i,j}$ and $a_{u,v}$. Concretely, we want the value of an edge to be large when the two pixels are similar, and small when they are different. Another kind of edge are edges between terminal nodes and foreground or background pixel nodes, which are chosen to be larger than any edges in the graph. In conclusion, the weights of edges are defined as follows:

$$C\left(a_{i,j}, a_{u,v}\right) = k \cdot \exp\left(\frac{-\left\|I_{a_{i,j}} - I_{a_{u,v}}\right\|_2^2}{2\sigma^2}\right) \tag{1}$$

$$C\left(S, F_i\right) = k \tag{2}$$

$$C\left(T, B_i\right) = k \tag{3}$$

where $k = 100$ by default is a scaling factor and σ^2 is determined by trials. Since one pixel has only four neighbors, the constructed matrix is sparse, we use linked list to store the weights of the graph to facilitate the algorithm.

2.2.2 Segmentation Using Graph Cut

Our aim is to cut the edges whose weight is small, and forming two parts $A1$ and $A2$. So our objective is to minimize the following cost function:

$$\text{Cost} = \sum_{p \in A_1, q \in A_2} C(p, q) \tag{4}$$

By the duality property of maximum flow and minimum cut problem, we can minimize the cost function by finding maximum flow of the graph. We use Ford-Fulkerson algorithm[2] to find the maximum flow.

Algorithm 1 Graph Cut: the Ford–Fulkerson Algorithm

Input: G: A directed connected graph, s: source node, t: sink node.
Output: A graph G after cutting with cut edges equals 0.

while $\exists$ an augmenting path $p \in G$ **do**
  $c(p) \leftarrow \min\{c_f(u, v) : (u, v) \in p\}$
  for each edge $(u, v) \in p$ **do**
    $w(u, v) \leftarrow w(u, v) - c(p)$
  end for
end while

The time complexity of this algorithm is $O(\max|f| \cdot E)$, which works fast when the image size is below 100×100. Downsampling is needed when segmenting a larger image.

2.3 Image Inpainting

In the inpainting part, we first use exemplar-based inpainting at the front layer to propagate the texture information to some percentage (0.2-0.3) that works well enough. Then use the Scene Completion to fill the area left.

2.3.1 Exemplar-based

Filling order is crucial to non-parametric texture synthesis. Thus Exemplar-Based Inpainting[5] designing a fill order which encourages the propagation of linear structure, together with texture. The algorithm will fill the best priority first. The priority computation is biased toward those patches which are on the continuation of strong edges and which are surrounded by high-confidence pixels.

The priority $P(p)$ of a patch Ψ_p centred at the point p for some $p \in \delta\Omega$ defined as the product of two terms: $P(p) = C(p)D(p)$ We call C(p) the confidence term and D(p) the data term, and they are defined as follows:

$$C(p) = \frac{\sum_{q \in \Psi_p \cap \bar{\Omega}} C(q)}{|\Psi_p|} \tag{5}$$

$$D(p) = \frac{|\nabla I_p^{\perp}|}{\alpha} \tag{6}$$

The confidence term C(p) may be thought of as a measure of the amount of reliable information surrounding the pixel p. The intention is to give preference to pixels that were filled early on (or that were never part of the target region). The data term D(p) is a function of the strength of isophotes on the front $\delta\Omega$. It will propagate the linear structure to the target region, and broken lines tend to connect. In the patch matching stage, different from [5], this algorithm define the distance not only by SSD of two patches but also add the locality similarity as a heuristic. The assumption is that the neighbor patches may contain similar textures, and can avoid matching different objects far from the pixel p^*.

$$d(\Psi_p, \Psi_q) = \beta SSD(\Psi_p, \Psi_q) + \gamma\sqrt{(p_x - q_x)^2 + (p_y - q_y)^2} \tag{7}$$

Algorithm 2 Exemplar Inpainting

Input: I: A masked picture, Ω: fill range, ψ: the front edge
Output: picture I^* inpainted with convincing texture patch

1: **while** $\exists$ pixel of fill range $p \in \Omega$ **do**
2: Identify the fill front $\delta\Omega^t$P(p)
3: Update the contour Ψ in the region to be filled
4: Update the priority information each pixel
5: Find the patch Ψ_{p^*} where pixel $p^* = argmax_{p \in \delta\Omega^t} P(p)$ with max priority
6: Find the exemplar $\Psi_{q^*} \in \Phi$ that minimize $d(\Psi_q, \Psi_{p^*})$, Φ is the known range in original picture.
7: Copy image data from Ψ_{q^*} to Ψ_{p^*}
8: Update confidence $C(p)$ $\forall p | p \in \Psi_{p^*} \cup \Omega$
9: **end while**

2.3.2 Scene Completion

This section introduces the scene completion algorithm to fill the rest region. Here users are allowed to input additional images similar to the original image. We then do template matching. We will use the patch around the mask as a template, denoted by T. Using additional input matching image as I. Notice that the mask region here should not affect the SSD score. So, we have a mask function as M, which is 0 at the black region and 1 at the white region.

$$R(x, y) = \sum_{x', y'} \left[(T(x', y') - I(x + x', y + y')) * M(x', y') \right]^2 \tag{8}$$

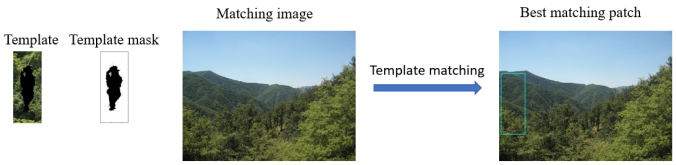

Figure 3: Template matching

In figure 3, the blue rectangle is the best-matching patch. Finally, as shown in figure 4, we will use Poisson blending to blend the best-matching patch to the template according to the mask.

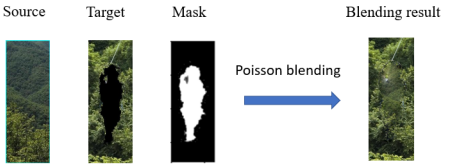

Figure 4: Poisson blending example

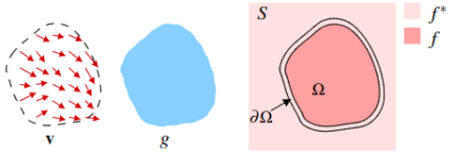

Figure 5: Blending process

The process of blending is shown in 5 where g is ROI(Source), v is the gradient of g, f^* is background image(Target) and f in Ω is the function we want to solve according to g and f^*. To calculate f, we need to solve 9. And 10 is the KKT condition for 9. These two constrains produce us enough number of equations to construct Linear system $Ax = b$. Here x is the value of f we want in Ω.

$$\min \iint_{\Omega} |\nabla f - v|^2 \text{ with } f|_{\partial\Omega} = f^* \mid \partial\Omega \tag{9}$$

$$\Delta f = \operatorname{div} v \text{ over } \Omega \text{ with } f|_{\partial\Omega} = f^* \mid \partial\Omega \tag{10}$$

3 Experiments

3.1 Dataset

The dataset contains 2,000 random pictures from all over the world, with indoor and outdoor scenarios, which can show algorithm flexibility[8].

3.2 Implementation Details

This project implements all the algorithms in Python. We first implement image segmentation. Two parameters σ^2 and $scale$ should be input by the user. σ^2 determines the sensitivity to intensity differences. When σ^2 is high, segmenting algorithm is less sensitive to small intensity differences. We set $\sigma^2 = 2700$ by default. $scale$ is the factor that the image is rescaled. In order to facilitate the algorithm, the size of the input image is resized to $(scale \times H, scale \times W)$.

In the exemplar patch searching part, instead of searching from all the patches, we approximately take a uniform distribution sample to reduce the searching range and speed up (sample 1000 patches each time and find the best as default). We set $\gamma = 1$ and $\beta = 1$ in the patch distance measurement part. As for the combine strategy, take ratio = 0.2 is good enough.

3.3 Performance evaluation of our algorithm

3.3.1 Evaluation Metrics

It is difficult to measure the result of our algorithm because it highly depends on human perception. Thus we performed a study on 17 participants to evaluate our results. Participants were given 10 random images and were asked to classify each image into manipulated or real images. An average number of correct classifications is recorded. Among the mix test, 55% of pictures generated from our algorithm were identified as real pictures with 1 second for the tester to recognize.

3.3.2 With and Without fine mask

We compared the performance of using segmentation to obtain mask and a roughly designed mask:

Figure 6: comparison of with and without fine mask

The average score is 8.2 for without fine mask and is 6.8 for fine mask ($p < 0.01$), which shows a significance improvement of our algorithm after using segmentation strategy.

3.3.3 Performance of Different Inpainting Strategies

We compared different inpainting strategies: using only Exemplar-based algorithm and a combination of scene matching and Exemplar-based image inpainting.

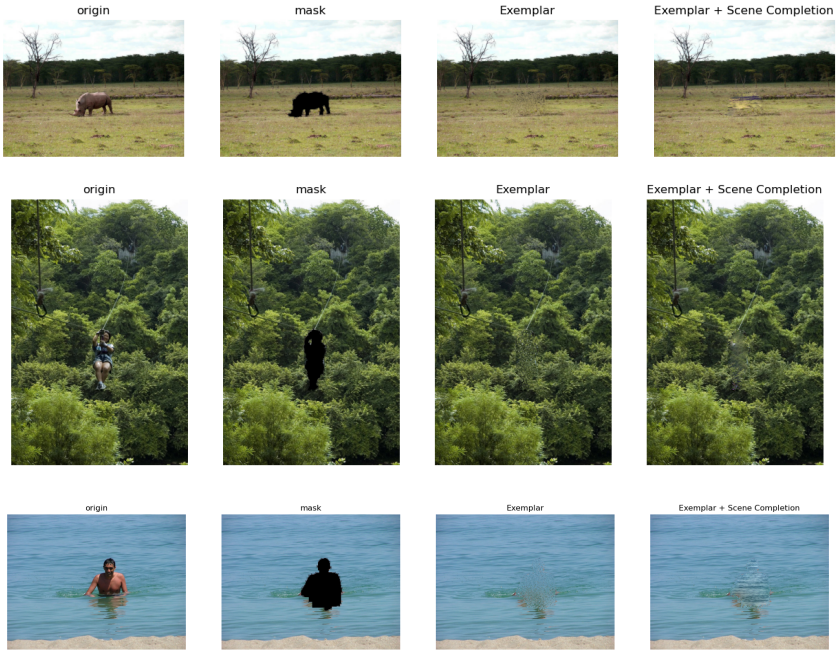

Figure 7: Performance of Different Inpainting Strategies

The average score is 7.9 for single inpainting strategy and is 7.1 for hybrid strategy, which also shows a significance ($p < 0.05$) improvement of our algorithm. The Exemplar part, on average uses 17 seconds to fill 1000 pixels, Scene Completion takes 0.134 seconds to do SSD for one matching picture. Poisson Blending takes 2.87 seconds to finish an image of size (358, 142).

4 Discussions

* The graph cut algorithm is slow for large images, further improvements include finding an approximation of maximum flow in graph cut, which can greatly speed up the algorithm.
* Although the isophote can control the linear structure propagate first, some unwanted textures will still pollute the unknown. In future work, the algorithm can use boundary information and guess the unknown structure. It can act like a skeleton and guide the texture filling.
* Users are required to manually add matching images. If we can construct a large image database. We can automatically search for matching images in the database and do template matching.

References

[1] . M. Bieniek, A. Watershed segmentation. *Mathematical Morphology and its Applications to Image and Signal Processing*, 12(1):215, 1998.

[2] . K. V. Boykov, Y. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. In *IEEE transactions on pattern analysis and machine intelligence*, pages 1124–1137. IEEE, 2004.

[3] J. V. M. B. C. Ballester, V. Caselles and G. Sapiro. A variational model for filling-in gray level and color images. *ICCV*, 1:10–16, 2001.

[4] T. F. Chan and J. Shen. Non-texture inpainting by curvature-driven diffusions (cdd). *Journal of visual communication and image representation*, 12(4):436–449, 2001.

[5] P. P. Criminisi, Antonio and K. Toyama. Object removal by exemplar-based inpainting. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings*, volume 2. IEEE, 2003.

[6] P. P. Criminisi, Antonio and K. Toyama. Region filling and object removal by exemplar-based image inpainting. *IEEE Transactions on image processing* , 13(9):1200–1212, 2004.

[7] J. Hays and A. A. Efros. Scene completion using millions of photographs. *ACM Transactions on Graphics*, 26(3):4–es, 2007.

[8] J. Hays and A. A. Efros. Im2gps: estimating geographic information from a single image. In *2008 ieee conference on computer vision and pattern recognition*, pages 1–8. IEEE, 2008.

[9] D. M. R. A. Y. F. T. R. Lugmayr, A. Repaint: Inpainting using denoising diffusion probabilistic models. arXiv, 2022.

[10] R. Pohle and K. D. Toennies. Segmentation of medical images using adaptive region growing. *Medical Imaging*, 4322(1):1337–1346, 2001.