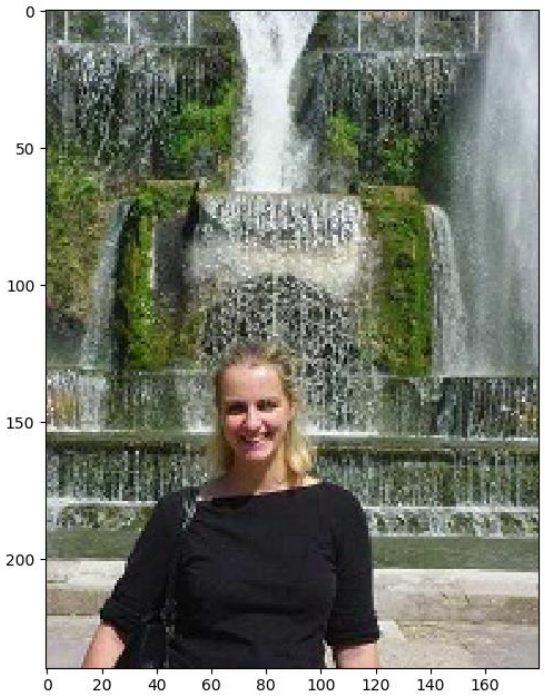


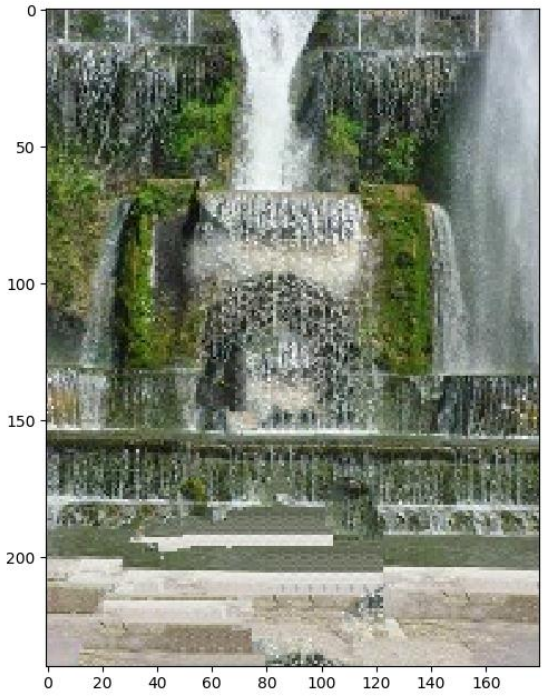


IMAGE INPAINTING

Final Course Project

WHAT'S INSIDE

Implementation report of "Region Filling and Object Removal" by A. Criminisi et al. [1]

Course

EE 645 3D Computer Vision

Name: Jaideep Singh Bankoti

Roll No: 19210044

Contents

Image Inpainting | Final Project

Abstract- Region filling using image inpainting techniques are widely used nowadays. Applications such as photoshop, illustrator or any editing software, whether it is for commercial purpose or application based (like completing an image with artifacts for forensics) use region filling algorithms to get the desired result. In this project I have studied and implemented a novel region filling algorithm in python.

Introduction

The algorithm used in the project is based on the seminal paper by *A. Criminisi et. al.* [1]. Region filling techniques fills holes in images by texture synthesis or by propagating the linear structures known as *isophotes.* But the major drawbacks as mentioned in [1] are that they have difficulty filling holes in real-life images (consisting of linear structures) or they might introduce blur which is noticeable when larger patches are filled.

So, the authors of [1] came up with a technique that amalgamates two approaches to get better results. One of them is exemplar-based texture synthesis and the other one is by following the peripheral constraints imposed by *isophotes*.

The Process

The notations indicated henceforth are in accordance to the paper [1]. Figure 1 shows the notations used, and Figure 2 indicates the importance of setting priorities to the target pixel. The green part indicates high priority whereas red indicates lower priority. Part a. of figure 2. corresponds to the confidence term and part b. corresponds to the data term.

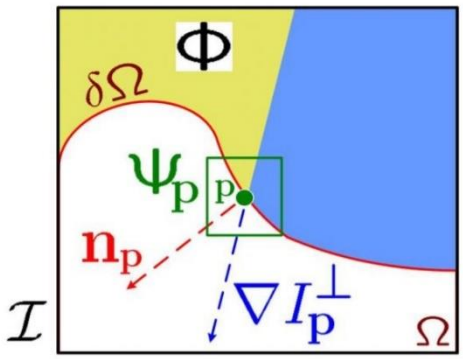


Fig. 1 **Notation diagram.** Given the patch $\Psi_{\mathbf{p}}$, $\mathbf{n}_{\mathbf{p}}$ is the normal to the contour $\delta\Omega$ of the target region Ω and $\nabla I_{\mathbf{p}}^{\perp}$ is the isophote (direction and intensity) at point $\mathbf{p}$. The entire image is denoted with $\mathcal{I}$.

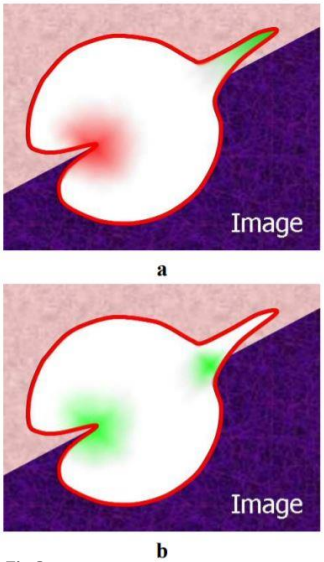


Fig. 2

** Both the figures above are taken from the paper [1]*

The confidence term smoothens the contour (the front) and remove sharp points so that the front becomes circular and we could progress concentrically inwards. The data term gives priority to the pixels on *isophotes*.

Besides the confidence and data terms needed to calculate the priority of the pixels, a data object is also made to store the color value. At each iteration the contour that indicates the mask boundary is calculated and it shrinks each time, until all the holes are filled.

The Procedure

The algorithm is mainly comprised of a three-step process that iterates inside a while procedure until all the target pixels inside the mask are filled. We also decrease the mask as we further progress through the algorithm. The step wise procedure is displayed in the table below.

- Extract the manually selected initial front $\delta\Omega^0$.
- Repeat until done:

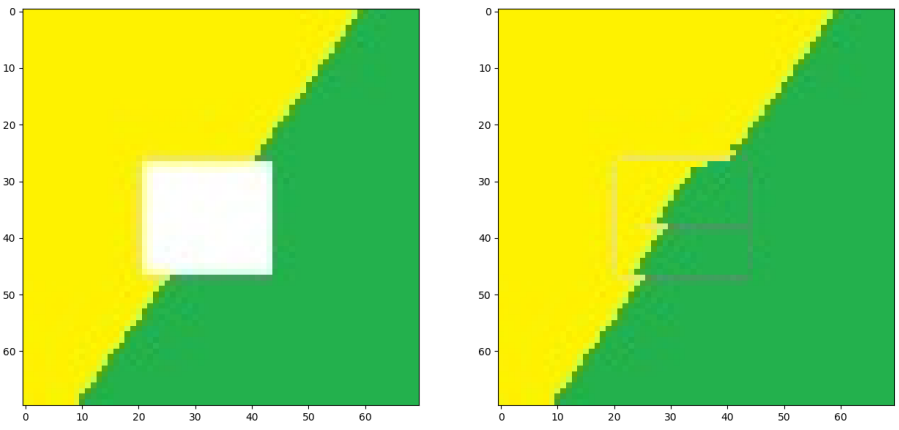
1a. Identify the fill front $\delta\Omega^t$. If $\Omega^t = \emptyset$, exit.
1b. Compute priorities $P(\mathbf{p}) \quad \forall \mathbf{p} \in \delta\Omega^t$.
2a. Find the patch $\Psi_{\hat{\mathbf{p}}}$ with the maximum priority, *i.e.*, $\hat{\mathbf{p}} = \arg\max_{\mathbf{p}\in\delta\Omega^t} P(\mathbf{p})$.
2b. Find the exemplar $\Psi_{\hat{\mathbf{q}}} \in \Phi$ that minimizes $d(\Psi_{\hat{\mathbf{p}}}, \Psi_{\hat{\mathbf{q}}})$.
2c. Copy image data from $\Psi_{\hat{\mathbf{q}}}$ to $\Psi_{\hat{\mathbf{p}}}$ $\forall \mathbf{p} \in \Psi_{\hat{\mathbf{p}}} \cap \Omega$.
3. Update $C(\mathbf{p}) \quad \forall \mathbf{p} \in \Psi_{\hat{\mathbf{p}}} \cap \Omega$

Table 1.

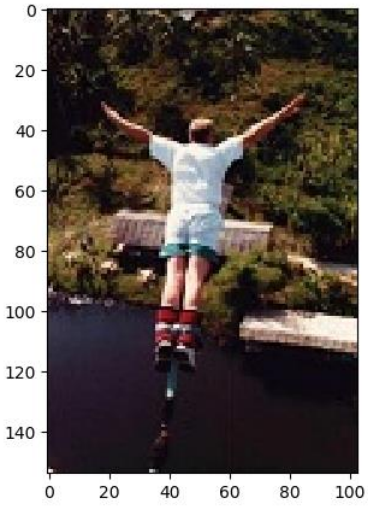
Results

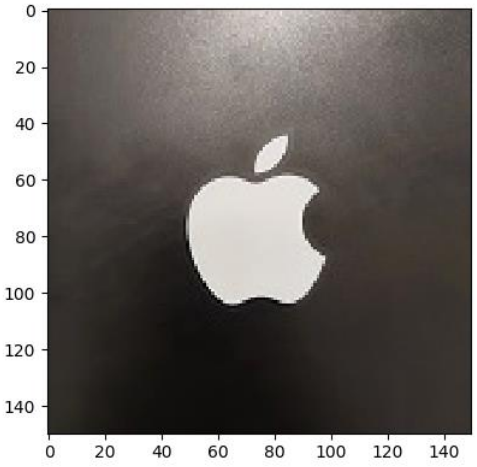
The python script ImageInpaint.py generated the following results:

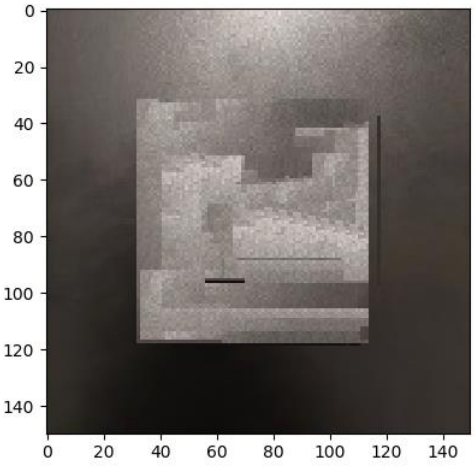
The left indicates the original image with holes or object, the right one is the generated one after region filling.



** The table used above is taken from the paper [1]*







All the results generated uses a patch length of 9.

Also, the original image and the mask are supplemented in the same directory as the python file in the project folder.

Remarks

- The algorithm works fine for almost all the query images but not on the one with apple logo. The reason could be, the surrounding textures are not as crisp and their color values are close for most of the pixels.
- The image used in cover page is also generated from the program.

References:

I have included all the references used in the project both in this report and the main python script.

- [1] *Paper Referenced: A. Criminisi, P. Perez and K. Toyama, "Region filling and object removal by exemplar-based image inpainting," in IEEE Transactions on Image Processing, vol. 13, no. 9, pp. 1200-1212, Sept. 2004, doi: 10.1109/TIP.2004.833105.*
- *Link to Paper: https://ieeexplore.ieee.org/abstract/document/1323101*
- *Referenced the following repository: https://github.com/igorcmoura/inpaint-object-remover*
- *https://numpy.org/doc/stable/reference/generated/numpy.argwhere.html*
- *https://stackoverflow.com/questions/21210479/converting-from-rgb-to-lab-colorspace-any-insight-into-the-range-of-lab-val*
- *https://scikit-image.org/docs/dev/api/skimage.filters.html#skimage.filters.laplace*
- *image1.jpg and mask1.jpg: self-made*
- *image2.jpg and mask2.jpg: from the resources folder of the above GitHub repository*
- *image3.jpg and mask3.jpg: self-clicked*
- *image4.jpg and mask4.jpg: from the resources folder of the above github repository*