INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Poisson Editing Algorithm for Image Blending

Ayoub El Arroud El Hadari, Ayoub Samih, Nacer Lahlou, Yasmine Kenfaoui

Supervisor

Charles Dossal

January 10, 2023

1 Introduction

Poisson image editing is an image processing technique used to modify or synthesize images in a seamless and visually plausible way. It involves transfering texture and gradient information from a source image to a target image, such that the resulting image looks natural and coherent.

The technique is based on the Poisson equation, which describes the distribution of a certain physical quantity (e.g., electric potential or pressure) in a given region.
The Poisson equation becomes the corresponds to the Laplace equation with boundary conditions:

By solving the Poisson equation with appropriate boundary conditions, it is possible to obtain a smooth and continuous function that describes the quantity of interest.

In the context of image processing, the Poisson equation can be used to synthesize or modify images by transferring texture and gradient information from one image to another. This can be useful for tasks such as image retouching, object insertion, and seamless cloning.

There are several algorithms and techniques available for solving the Poisson equation in the context of image processing.

In this report, we will be presenting three ways of applying the Poisson Editing algorithm using gradient-based methods.

2 Theory

The goal of this assignment is to apply the Poisson editing algorithm for image blending. In the following, T^* is a target image, S a source image, and a binary mask representing an area Ω of S to copy in T^*. All images are defined on the same domain D of size $M \times N$. We aim to solve the Poisson equation over Ω:

$$\Delta f = div(v)$$

Such that: f is the image to interpolate and v is the guidance field This is equivalent to solving the minimization problem:

$$\min_u \int \|\nabla u - v\|^2, u_{\Omega - D} = T^*_{\Omega - D} \tag{1}$$

We choose to write the constraint as :

$$u \in K = \{z : z_{\Omega^c} = target_{\Omega^c}\}$$

The initial problem (1) can therefore be re-written as:

$$\min_u J(u) = \int_\Omega \|\nabla u - v\|^2 + \iota_K(u) \tag{2}$$

The indicator function being non-differentiable, we need to use a Forward-Backward method in order to obtain a solution. An iteration of the Forward-Backward algorithm is given by:

$$x_{n+1} = prox_g(x_n - \gamma \nabla f(x_n))$$

Where:

$$f = \|\nabla\|^2$$

and

$$g = \iota_K$$

In our case the proximal operator is nothing but the orthogonal projection on K. We will also be using the Fast Iterative Shrinkage-Thresholding Algorithm (FISTA) which is simply an accelaration of Forward-Backward.

3 Seamless cloning

Seamless cloning is a technique used to blend an image or a part of an image called the source onto another image called the target, while keeping the result natural and seamless. It is often used to remove or add objects in images, or to repair damaged or missing parts of images.

3.1 By Importing Gradients

One way to achieve seamless cloning using Poisson image editing is to import gradients from the surrounding image pixels into the area where the cloning changes are made. This helps to blend the changes smoothly with the rest of the image, resulting in a more natural and seamless appearance. In practice, this is done by choosing for the guidance field v we talked about in the previous section as a gradient field taken directly from a source image.
Denoting by g this source image, the interpolation is performed under the guidance of $v = \nabla g$ and the problem now becomes :

$$\min_f \int_\Omega ||\nabla f - \nabla g||^2 + \iota_K(f)$$

We defined a source region as the area of the image where we want to clone, and a target region corresponding to the area of the image where we want to place the cloned content. We can then use Poisson image editing to blend the gradients from the surrounding pixels in the target region with the pixels in the source region, resulting in a seamless blend of the two regions.
Here we display the results of the poisson editing algorithm using importing gradients to add a giant octupus on a boat in the sea:

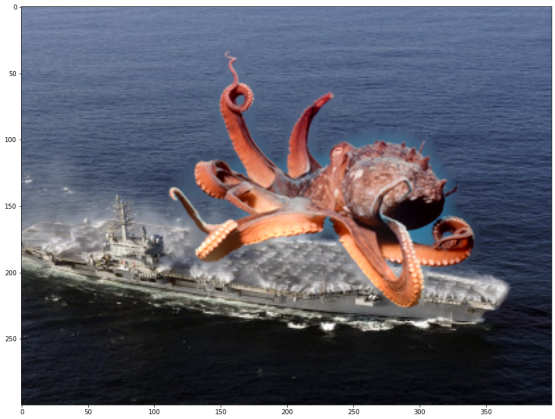

Figure 1: Image used for this study

This method does its job but we can see that it's still imperfect on the edges.

3.2 By Mixing Gradients

With the tool described in the previous section, no trace of the destination image f^* is kept inside Ω. However, there are situations where it is desirable to combine properties of f^* with those of g. For example, one might want to add objects with holes, or partially transparent ones, on top of a textured or cluttered background.
Another approach would be then to define the guidance field v as a linear combination of source and destination gradient fields since this has the effect of washing out the textures. This is done in practice by comparing the gradient of the two images and only keeping the pixels with the highest variation. We can formulate this as follows:

$$v = \begin{cases} \nabla t & \text{if } |\nabla t| > |\nabla s| \\ \nabla s & \text{otherwise} \end{cases}$$

Here we display the result of the algorithm using mixing gradients used to add a rainbow to an image of the sky (to always bring joy and colors in people's life).

(a) Happy sky but no rainbow (b) Sad sky with a rainbow (c) Happy sky WITH a rainbow

Figure 2: Poisson editing algorithm using mixing gradients used to add a rainbow to an image of the sky

4 Selection editing

In the previous sections, the guidance field v was retrieved from an other source image. In the following, we will rely on the original image itself to perform texture flattening tasks.

4.1 Texture Flattening

The goal of this part is to remove or "flatten" the texture from an image while preserving the overall structure and shape of the objects in the image. This can be useful for a variety of applications, such as object recognition, image segmentation, and image manipulation.

To achieve this goal, we use edge detection to identify the boundaries of the objects in the image, and then use image inpainting to fill in the texture within these boundaries with a smooth, uniform color or texture.

The first step in our approach was to apply an edge detection algorithm to the input image. We chose to use the Canny edge detector, which is a well-known and widely-used algorithm that is effective at identifying the boundaries of objects in an image.

The result of this edge detector is not bad as we can see in this image where edges are represented by white pixels and the rest by black pixels :

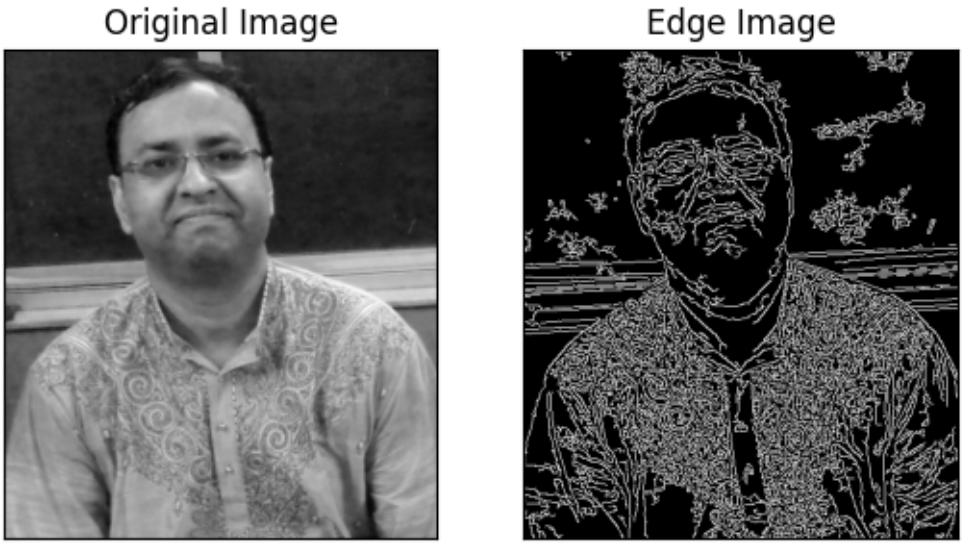

Figure 3: Result of canny edge detector

Next, we transformed the image data using a gradient transformation and the edge map produced by the Canny detector. This transformation was designed to highlight the edges of the objects in the image and suppress the texture within the objects, as shown above in the "edge image".

After transforming the image data, we used a binary mask to select the region of the image that we wanted to apply the inpainting algorithm to. The mask is an array of zeros with dimensions 287x373, with a rectangular region of ones centered in the middle of the array. This mask is used to select the region of the image that delimits to the skin of the figure in the image.

Finally, we applied an image inpainting algorithm to the transformed image data, using the edge map and the mask as a guide for filling in the missing data. We chose to use the Fast Iterative Shrinkage-Thresholding Algorithm (FISTA) for this task, as it is a fast and effective algorithm for image inpainting

Here is the result obtained:

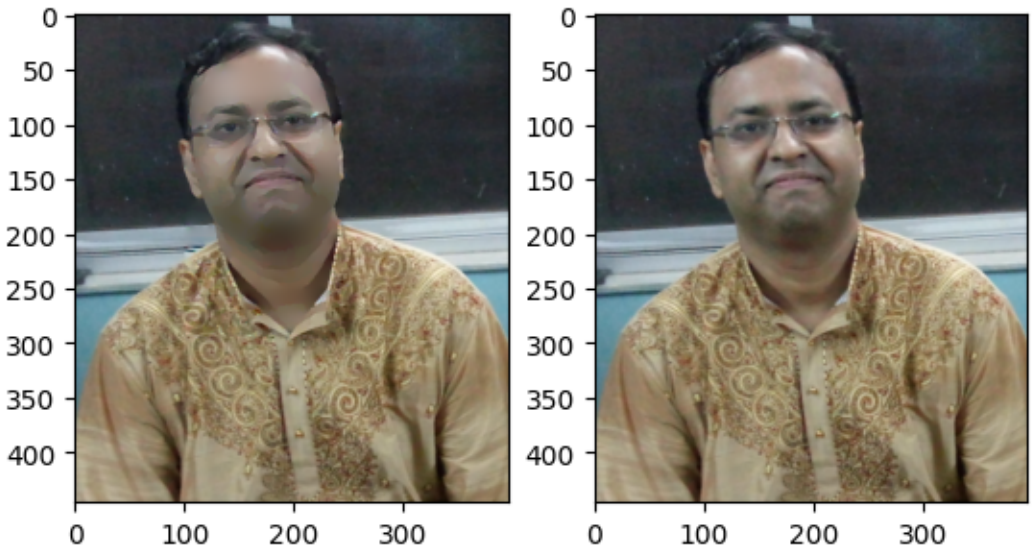

Figure 4: Result of texture flattening

The resulting image showed the texture removed or "flattened" in the regions marked by the mask, while the overall structure and shape of the objects in the image was preserved.

Finally, we successfully implemented a method for removing texture from an image while preserving the overall structure and shape of the objects in the image. The combination of edge detection, image transformation, and image inpainting proved to be an effective approach for achieving this goal, and the use of the Canny edge detector and the FISTA allowed us to implement this method efficiently and accurately.

4.2 Contrast enhancement

Contrast enhancement is a technique used in image processing to improve the visual quality of an image by adjusting the relative intensity of its pixels. The goal of contrast enhancement is to stretch the intensity range of the image so that the darkest pixels are as dark as possible and the lightest pixels are as light as possible, while maintaining the relationships between the different intensities. This can make the image appear more vivid and easier to see details in. There are many different techniques for contrast enhancement, in the next part we will explain our methodology/coding process.
You can refer to the Notebook and the code that implements an image processing pipeline that involves applying a contrast enhancement function to an image.
We try here to weight the values of the gradient by comparing its norm with the average value of the

norm of the gradients. The parameter $\alpha \geq 0$ allows to designate a threshold from which the gradients will be amplified or decreased. The parameter β represents the strength of this amplification. We then apply FISTA to the enhanced image.

We then created a dashboard enabling to set alpha and beta at different values and see their impact on the resulted image.
Here is the result obtained:

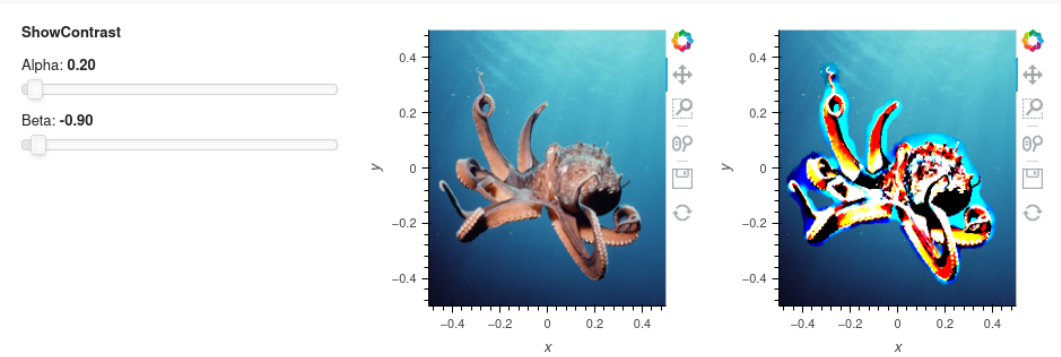

Figure 5: Result of Contrast enhancing

4.3 Local illumination changes

In this part, we aim at modifying the apparent illumination of an image. To do so, we apply a non-linear transformation to the gradient field inside the selection and then integrating back with a Poisson solver. This is useful to highlight under-exposed foreground objects or to reduce reflections. The guidance filed to be used is:

$$v = \alpha^{\beta} |\nabla f|^{-\beta} |\nabla f|$$

In the following example, we fix $\beta = 0.5$ and try different values of α. Here are the results obtained respectively for $\alpha = 100$, and $\alpha = 500$

Figure 6: Low Light Image

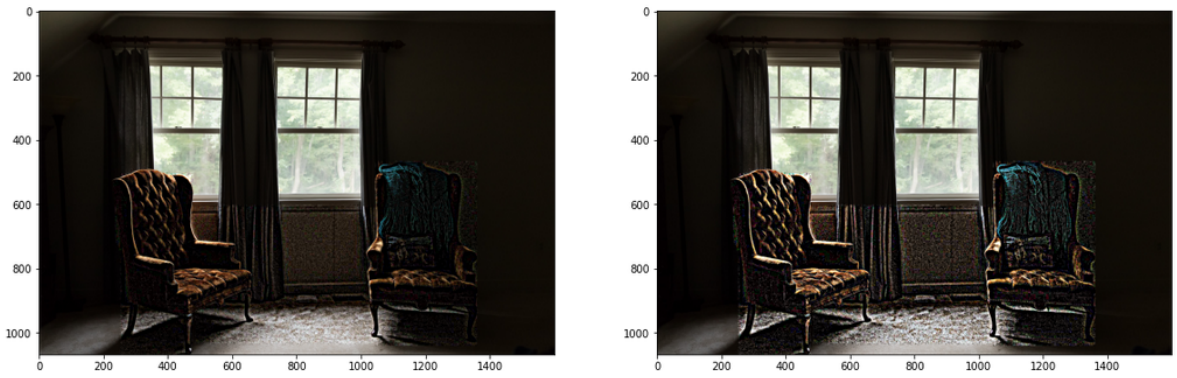

Figure 7: Luminosity Increase

We can see that there exists a trade-off according to α: high values of α provide more enlightened images, but they also have a strong "insertion" effect, and vice-versa.