C2: Optimization Methods for CV

Lab 2: Image Inpainting (& Blending)

Group 5

Cristian Gutiérrez
Marco Cordón
Iñaki Lacunza

Index of the Lab

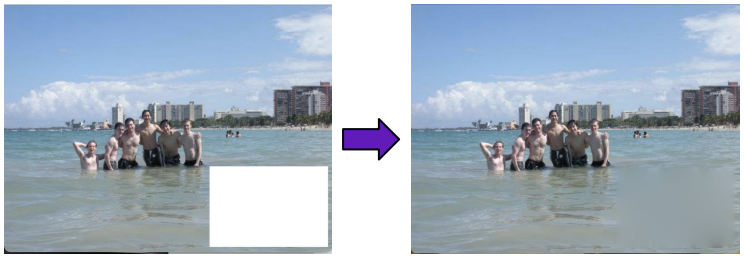
1) Mathematical Model
2) Constraints & Solutions
3) Methods Implemented:
    a) Seamless Cloning
    b) Seamless Cloning + Background Addition
    c) Seamless Cloning + Destination Average
    d) Mixing Gradients
4) Comparisons (I) (II) (III)
5) Discussion
6) Conclusions

Problem: Image Inpainting (& Blending)

Previously in Lab 1, we had a gap B to be filled consisting in a mask with no added value, meaning that the mask was empty.

Now we have another image that we would like to seamlessly blend together.

Lab 1 (Before)



Lab 2 (Now)

Seamless Cloning: Mathematical Model

As always, the first step is to modelize the problem. We will define the destination image, to be inpainted, as $A(x, y)$, and the source image, as $B(x, y)$.

In this context, the image we aim to transfer will be the smaller one.

Such that:

$$A > B, \text{ where } > \text{ denotes a higher resolution or size.}$$

This time, $V(A, B^*)$ will contain both of the images A and B^*. Where * denotes that the transferred image is blended with the original.

Seamless Cloning: Constraints

First Constraint: Smooth transition (Intersection between regions)

As in previous week, we do not want to alter the region that will not get inpainted (the region A).

$$V(x, y) = A(x, y), \quad \forall (x, y) \in A, \ \forall (x, y) \notin B.$$

And, in the boundaries of B, we have to assign the original pixels of A to ensure that the transition between A and B is smooth.

$$V(x, y) = A(x, y), \quad \forall (x, y) \notin A, \ \forall (x, y) \in \text{ Boundaries of } B.$$

```
# Boundaries of B:
if ((bin_mask_ext[i + 1, j] == 0) or
   (bin_mask_ext[i - 1, j] == 0) or
   (bin_mask_ext[i, j + 1] == 0) or
   (bin_mask_ext[i, j - 1] == 0)):

    idx_Ai.append(p)
    idx_Aj.append(p)
    a_ij.append(1)

    b[p] = f_ext[i, j]
```

Second Constraint: Gradients (Region B)

In order to get a smooth output image without losing the added value of the transferred image, we want to take the gradients of image B into account. So that we can blend them without losing high-frequency elements.

$$\nabla V(x, y) = \nabla B(x, y), \quad \forall (x, y) \in B.$$

Seamless Cloning: Matrix Representation

Considering the current constraints, the formulation of the sparse matrix is the following:

$$\begin{pmatrix} 2 & -1 & 0 & \dots & 0 & -1 & 0 & & & & \dots & & & & 0 \\ & & & & & & & \dots & & & & & & & \\ 0 & & & \dots & & & 0 & 1 & 0 & & & \dots & & & 0 \\ & & & & & & & \dots & & & & & & & \\ 0 & \dots & 0 & -1 & 0 & \dots & 0 & -1 & 4 & -1 & 0 & \dots & 0 & -1 & 0 & \dots & 0 \\ & & & & & & & & \dots & & & & & & & & \\ 0 & & & & & \dots & & & & & 0 & -1 & 0 & \dots & 0 & -1 & 2 \end{pmatrix} * \begin{pmatrix} v_1 \\ \dots \\ v_{i,j} \\ \dots \\ \\ \dots \\ \\ \dots \\ v_{(m+2)*(n+2),(m+2)*(n+2)} \end{pmatrix} = \begin{pmatrix} 0 \\ \dots \\ A_{i,j} \\ \dots \\ \nabla B_{i,j} \\ \dots \\ 0 \end{pmatrix} \begin{matrix} \leftarrow \text{North boundary} \\ \\ \leftarrow \text{Image A} \\ \\ \leftarrow \text{Image B} \\ \\ \leftarrow \text{South boundary} \end{matrix}$$

Seamless Cloning: Gradients

The gradient is a differentiation operation that outputs the rate of change of a given function f and a small increasing rate tending to 0, h. In other words, it give us the regions of an image with an abrupt color intensity difference, where the image goes from a stable set of values to a different one.

$$\nabla V(x,y) = \nabla B(x,y), \quad \forall (x,y) \in B.$$

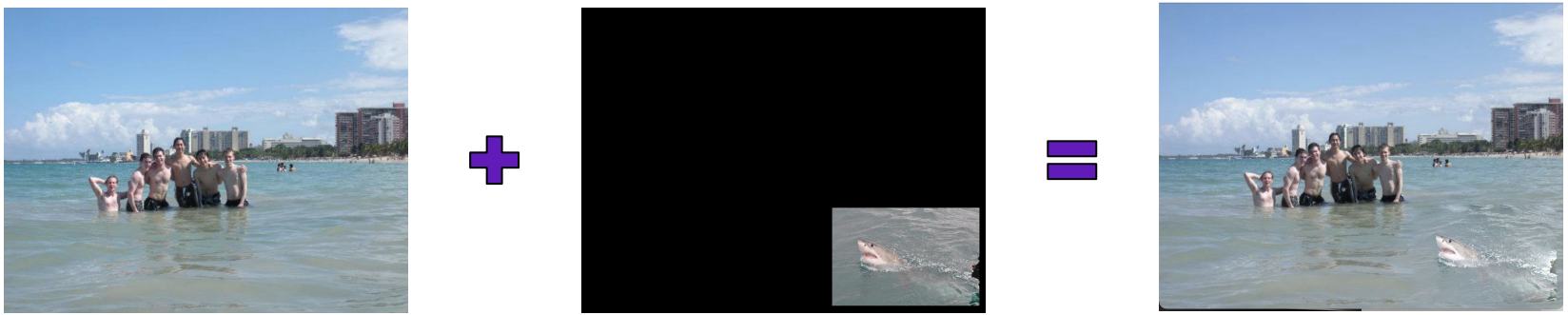
$$\downarrow$$

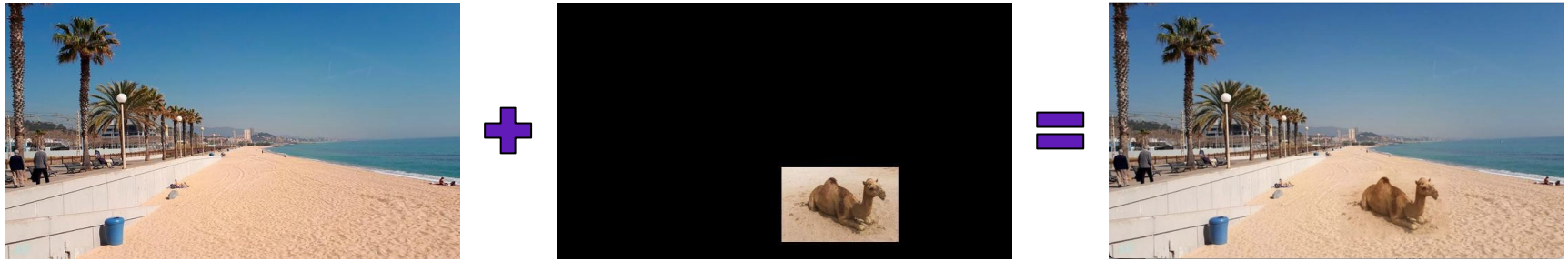
$$\begin{cases} \nabla V(x,y) = 4V(x,y) - V(x \pm 1, y \pm 1) \\ \nabla B(x,y) = 4B(x,y) - B(x \pm 1, y \pm 1) \end{cases}$$

```
idx_Ai.append(p)
idx_Aj.append(p)
a_ij.append(4)

idx_Ai.append(p)
idx_Aj.append(p+1)
a_ij.append(-1)

idx_Ai.append(p)
idx_Aj.append(p-1)
a_ij.append(-1)

idx_Ai.append(p)
idx_Aj.append(p+ni+2)
a_ij.append(-1)

idx_Ai.append(p)
idx_Aj.append(p-(ni+2))
a_ij.append(-1)

grad_B = ((4 * dom2Inp_ext[i, j]) -
          (dom2Inp_ext[i-1, j] +
           dom2Inp_ext[i, j-1] +
           dom2Inp_ext[i+1, j] +
           dom2Inp_ext[i, j+1]))
b[p] = grad_B
```

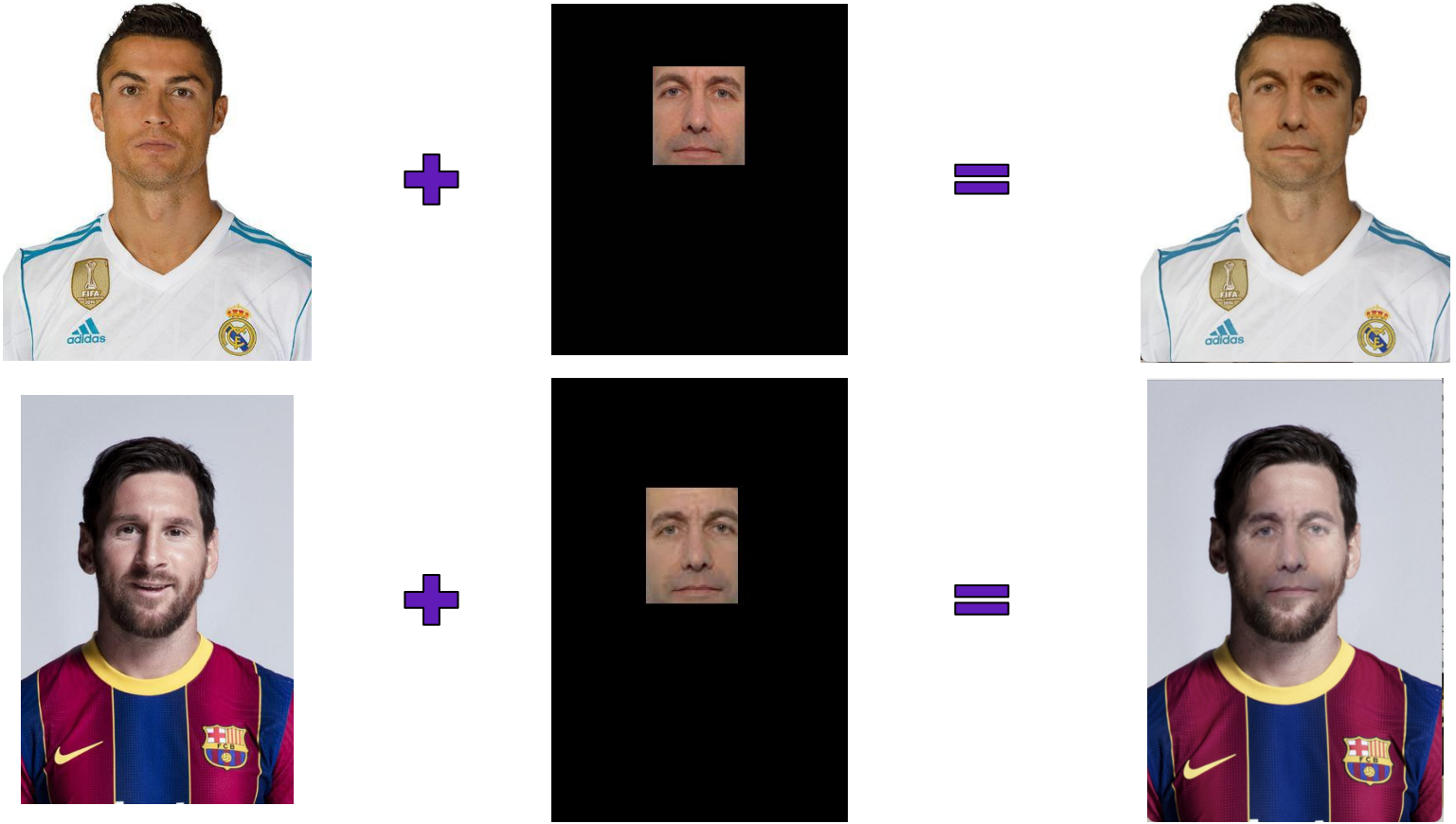
* Python index notation will be used through the entire Lab.

Seamless Cloning: Examples I





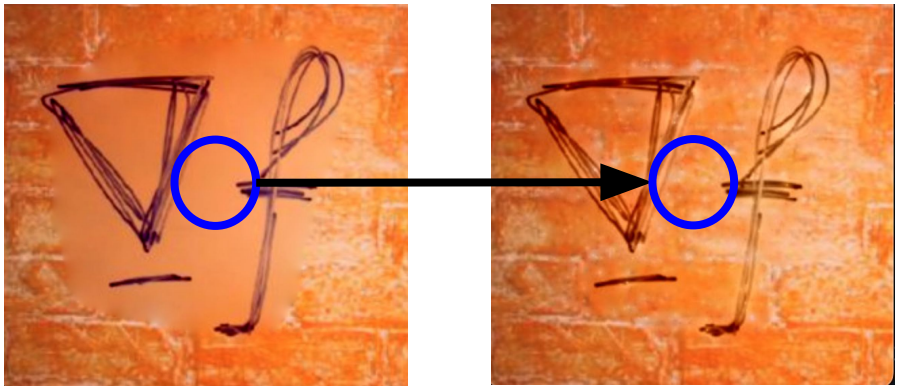
Seamless Cloning: Examples II



Method 2: Seamless Cloning with Background Addition

Even though Seamless Cloning works well with the source image B, the background corresponding to the destination image A is affected by our Smoothness constraint.

In order to avoid an averaged background without detail, we will impose an extra constraint on regions of B where its gradient is zero. In which case we will assign the original pixels.



Seamless Cloning

Seamless Cloning
w/ Background Addition

```
# Avoid regions where grad B is zero
if abs(grad_B) > 0:

    # Same procedure as Seamless Cloning...

else:
    # Assign pixel of destination image A
    idx_Ai.append(p)
    idx_Aj.append(p)
    a_ij.append(1)

    b[p] = f_ext[i, j]
```

* With the addition of a simple *if* statement we are able to keep the background.

Method 3: Seamless Cloning and Destination Averaged

Instead of just considering the non-zero gradients of the source image, there might be cases in which we want to get a combination of the source and destination images' gradients. One approach is to get the average of both gradients. This method is called Seamless cloning and destination averaged.

$$\nabla V(x,y) = \frac{\nabla A(x,y) + \nabla B(x,y)}{2}$$

```
gradb = 4 * dom2Inp_ext[i, j] - (dom2Inp_ext[i-1, j] + dom2Inp_ext[i, j-1] + dom2Inp_ext[i+1, j] + dom2Inp_ext[i, j+1])
grada = 4 * f_ext[i, j] - (f_ext[i-1, j] + f_ext[i, j-1] + f_ext[i+1, j] + f_ext[i, j+1])
b[p] = (grada + gradb)/2
```

Method 4: Mixing Gradients

Mixing Gradients is a strategy that tries to always account for the biggest gradient of both destination image A and source image B. Therefore maintaining always the image that disrupts the most up-front.

In our case, during the Lab, we have used a modified version of Mixing Gradients that worked better for our set of images defined by the following model:

$$\begin{cases} V(x,y) = A(x,y) & \forall (x,y) \in |\nabla A(x,y)| > |\nabla B(x,y)| \\ \nabla V(x,y) = \nabla B(x,y) & \forall (x,y) \in |\nabla A(x,y)| \leq |\nabla B(x,y)| \end{cases}$$

```
if abs(grada) > abs(gradb):
    idx_Ai.append(p)
    idx_Aj.append(p)
    a_ij.append(1)
    b[p] = f_ext[i,j]
else:
    idx_Ai.append(p)
    idx_Aj.append(p)
    a_ij.append(4)

    idx_Ai.append(p)
    idx_Aj.append(p+1)
    a_ij.append(-1)

    idx_Ai.append(p)
    idx_Aj.append(p-1)
    a_ij.append(-1)

    idx_Ai.append(p)
    idx_Aj.append(p+ni+2)
    a_ij.append(-1)

    idx_Ai.append(p)
    idx_Aj.append(p-(ni+2))
    a_ij.append(-1)
    b[p] = gradb
```

Comparisons (I)

In this case, the image B is an outline of a person, composed by the edges of the image.

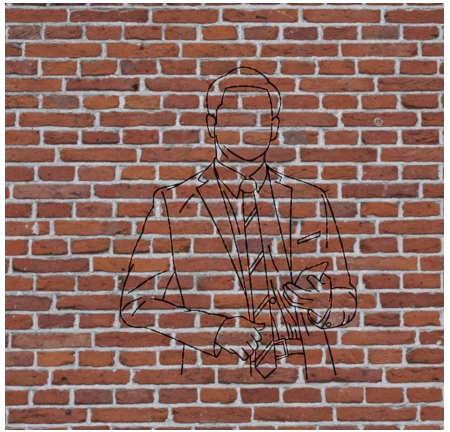
By the given nature of the image, it will have bigger gradients than the destination image and Mixing Gradients will make better use of the imposition of more abrupt changes, such as this one.

Seamless Cloning

Seamless Cloning
w/ Background Addition

Average

Mixing Gradients ✅



Comparisons (II)

In this case we take an angry man and we add him a smile.

The best option is seamless cloning since in the final image we want all the smile image not only the parts where the gradient is not 0.

Seamless Cloning ✅

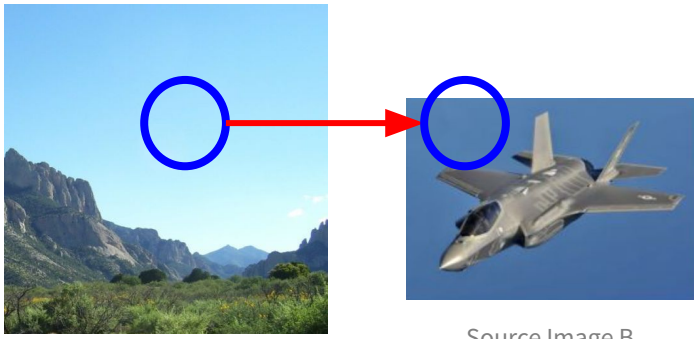
Seamless Cloning
w/ Background Addition

Average

Mixing Gradients

Comparisons (III)

In this case, both the destination and the source image have similar monotone blue backgrounds. The method tries to assemble the images the most, therefore making the jet more brighter to assemble the brighter blue of the background.



Destination Image A

Source Image B

Seamless Cloning ✅

Seamless Cloning
w/ Background Addition

Average

Mixing Gradients

Discussion

Seamless Cloning: This is the simplest method. Since it just takes into account the gradients of the source image, its pronounced edges are clearly inpainted in the destination image. But, for the regions of the source image where the gradients are null, the result is achieved by smoothing the surrounding pixels of the destination image (what we implemented last week). So, on big constant regions the result is not satisfactory.

Seamless Cloning + Background Addition: The biggest problem of seamless cloning is easily resolved by just copying the pixels of the source image which have a null gradient. This extra constraint improves the results a lot, especially for source images with big flat regions. But, we are not considering the gradients of the destination image, and there may be cases where we want to use a combination of both gradients.

Seamless Cloning and Destination Averaged: With this method we take into account both gradients. It is helpful when we want to inpaint partly transparent objects, or where the source image is very similar in color to the destination image we want to inpaint. But, it is not effective when we want to get sure that the most pronounced edge is selected.

Seamless Mixing Gradients: Because this method will always favour those images with very abrupt changes, hence a bigger gradient, it will display the contours of the source image. By the own given nature of the property we've just defined, this method will work best when we want to just get the contours of the destination and source images.

Conclusions

- We have analyzed four different methods for inpainting a source image into a destination image. Each method has its own advantages and disadvantages and the selection of the most suitable method depends on the source and destination images, as well as the desired output.

- As demonstrated in the Poisson Image Editing paper, using a guidance vector allows for seamless blending of the images. Depending on the vector chosen, we can select the desired elements from each image to be included in the output image.

- Last week we were able to smoothly fill selected regions of an image and we have now been able to complete this week's task with just a few additional code implementations.