Master in Computer Vision *Barcelona*

Module: M2

Project: Image Inpainting

Team:

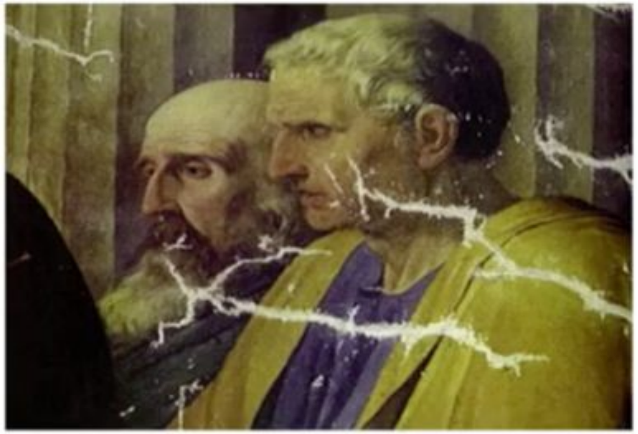
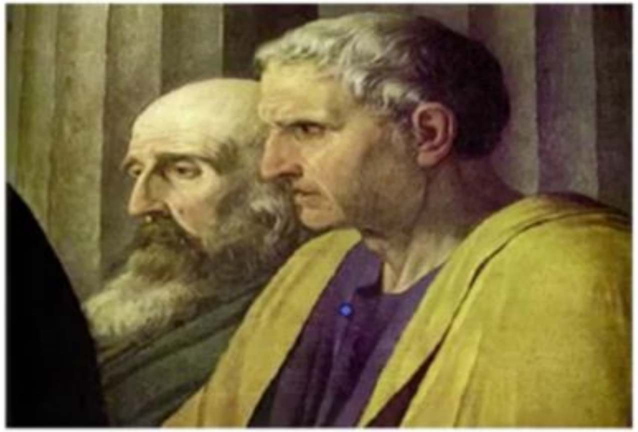
- Òscar Lorente Corominas
- Antoni Rodriguez Villegas
- Aditya Sangram Singh Rana

https://github.com/adityassrana/MCV-M2-OICV

Problem to Solve

This project aims to solve the problem of inpainting.

Inpainting: Input image has lost information in some parts, inpainting produces an output image with newly filled information for these regions.

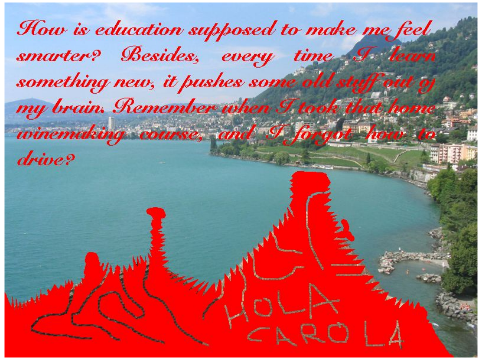



Method: Define Criterias

Given an image ***U*** divided in two regions ***A*** and ***B***, where ***B*** is the region to be inpainted and ***A*** the region that does not need to be inpainted. The criteria for producing the inpainted image ***V*** is as follows:

1. In A, the inpainted image should be the same as the original image
2. In B, the inpainted image should be smooth

Method: Form Mathematical Equations


How is education supposed to make me feel smarter? Besides, every time I learn something new, it pushes some old stuff out of my brain. Remember when I took that home winemaking course, and I forgot how to drive?
HOLA
CAROLA

Assuming an input image u(x, y) with two regions A and B

- *Region A: known*
- *Region B: unknown, to be inpainted (red region)*

The inpainted image v(x,y) can be obtained in the following way:

1. In region A: $u_{i,j} = v_{i,j}$

2. In region B:

$$4 * v_{i,j} - (v_{i+1,j} + v_{i-1,j} + v_{i,j+1} + v_{i,j-1}) = 0$$

Method: Boundary Conditions

To inpaint the pixels in the image, we add a ghost row/column on each side of the image. This is done so that the same mathematical constraints can be applied on all pixels of the image including the ones on the borders and corners of the original image.

Method: Boundary Conditions (von Neumann)

Given an image u of size(ni, nj) we add ghost boundaries and create an image v of size (ni+2, nj+2). The ghost boundaries must satisfy the following constraints:

- north boundary condition: $v_{1, j} - v_{2, j} = 0$
- south boundary condition: $v_{ni+2, j} - v_{ni+1, j} = 0$
- west boundary condition: $v_{i, 1} - v_{i, 2} = 0$
- east boundary condition: $v_{i, nj+2} - v_{i, nj+1} = 0$

Solution

The solution to the inpainting problem satisfying the above constraints is obtained by solving the following linear system of equations where u(size: ni, nj) is the original image and v(size: ni+2, nj+2) is the inpainted image with ghost boundaries

$$\begin{pmatrix}
2 & -1 & 0 & \dots & 0 & -1 & 0 & & & & & \dots & & & & & 0 \\
 & & & & & & & & \dots & & & & & & & & \\
0 & & & \dots & & & & 0 & 1 & 0 & & & & \dots & & & 0 \\
 & & & & & & & & \dots & & & & & & & & \\
0 & \dots & 0 & -1 & 0 & \dots & 0 & -1 & 4 & -1 & 0 & \dots & 0 & -1 & 0 & \dots & 0 \\
 & & & & & & & & \dots & & & & & & & & \\
0 & & & & & \dots & & & & & 0 & -1 & 0 & \dots & 0 & -1 & 2
\end{pmatrix} * \begin{pmatrix} v_1 \\ \dots \\ v_{i,j} \\ \dots \\ \dots \\ \dots \\ v_{(m+2)*(n+2),(m+2)*(n+2)} \end{pmatrix} = \begin{array}{ll} \left(\begin{matrix} 0 \\ \dots \\ u_{i,j} \\ \dots \\ 0 \\ \dots \\ 0 \end{matrix}\right) & \begin{matrix} \leftarrow \text{North boundary} \\ \\ \leftarrow \text{Region A} \\ \\ \leftarrow \text{Region B} \\ \\ \leftarrow \text{South boundary} \end{matrix} \end{array}$$

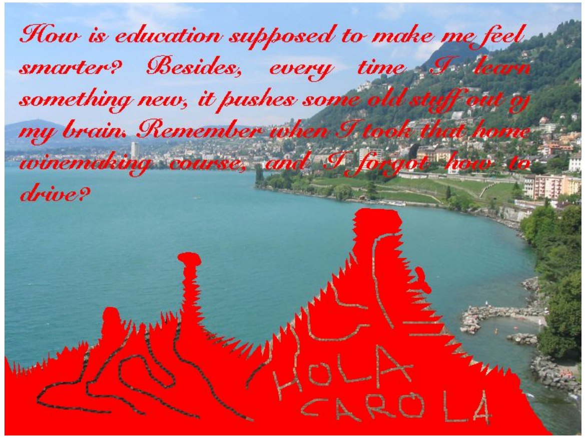
Size = ((m+2)*(n+2) , (m+2)*(n+2))

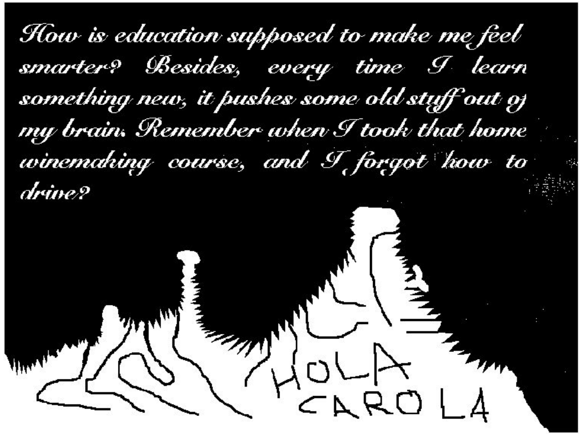
Size = ((m+2)*(n+2) , 1)

Size = ((m+2)*(n+2) , 1)

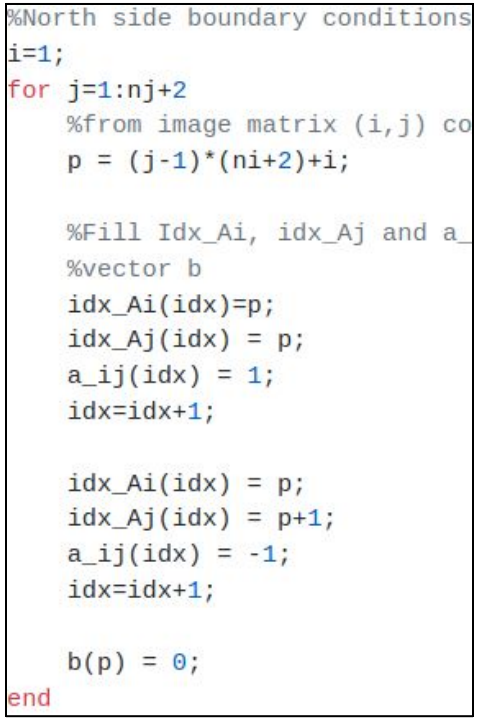
Code Implementation (Task 1: Separate region A and B)

```
%We want to inpaint those areas in which mask == 1 (red part of the image)
I_ch1 = I(:,:,1);
I_ch2 = I(:,:,2);
I_ch3 = I(:,:,3);

mask = I_ch1 == 1; %mask_img(i,j) == 1 means we have lost information in that pixel
                   %mask(i,j) == 0 means we have information in that pixel
```

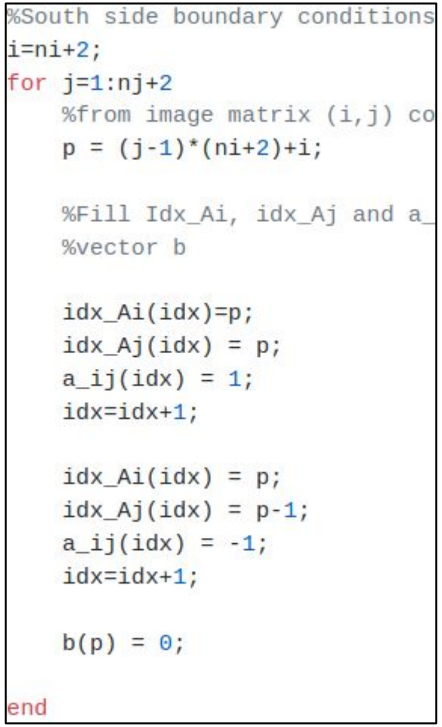

How is education supposed to make me feel smarter? Besides, every time I learn something new, it pushes some old stuff out of my brain. Remember when I took that home winemaking course, and I forgot how to drive?


How is education supposed to make me feel smarter? Besides, every time I learn something new, it pushes some old stuff out of my brain. Remember when I took that home winemaking course, and I forgot how to drive?

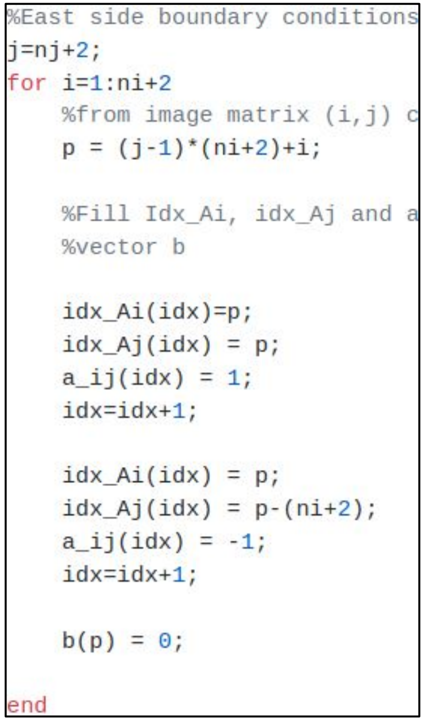
Code Implementation (Task 2, 3, 4: Boundary Conditions)



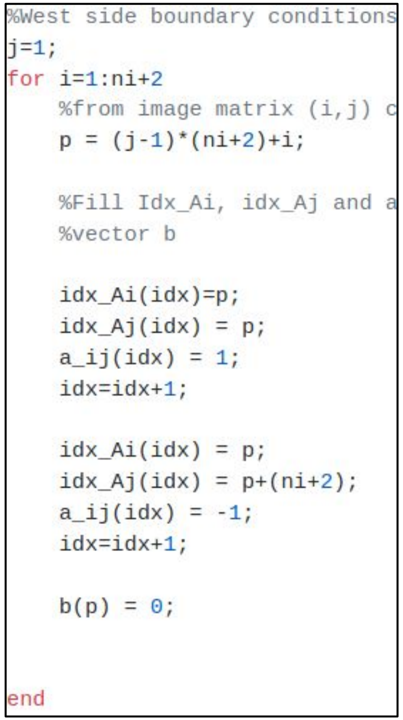
```
%North side boundary conditions
i=1;
for j=1:nj+2
    %from image matrix (i,j) co
    p = (j-1)*(ni+2)+i;

    %Fill Idx_Ai, idx_Aj and a_
    %vector b
    idx_Ai(idx)=p;
    idx_Aj(idx) = p;
    a_ij(idx) = 1;
    idx=idx+1;

    idx_Ai(idx) = p;
    idx_Aj(idx) = p+1;
    a_ij(idx) = -1;
    idx=idx+1;

    b(p) = 0;
end
```



```
%South side boundary conditions
i=ni+2;
for j=1:nj+2
    %from image matrix (i,j) co
    p = (j-1)*(ni+2)+i;

    %Fill Idx_Ai, idx_Aj and a_
    %vector b

    idx_Ai(idx)=p;
    idx_Aj(idx) = p;
    a_ij(idx) = 1;
    idx=idx+1;

    idx_Ai(idx) = p;
    idx_Aj(idx) = p-1;
    a_ij(idx) = -1;
    idx=idx+1;

    b(p) = 0;

end
```



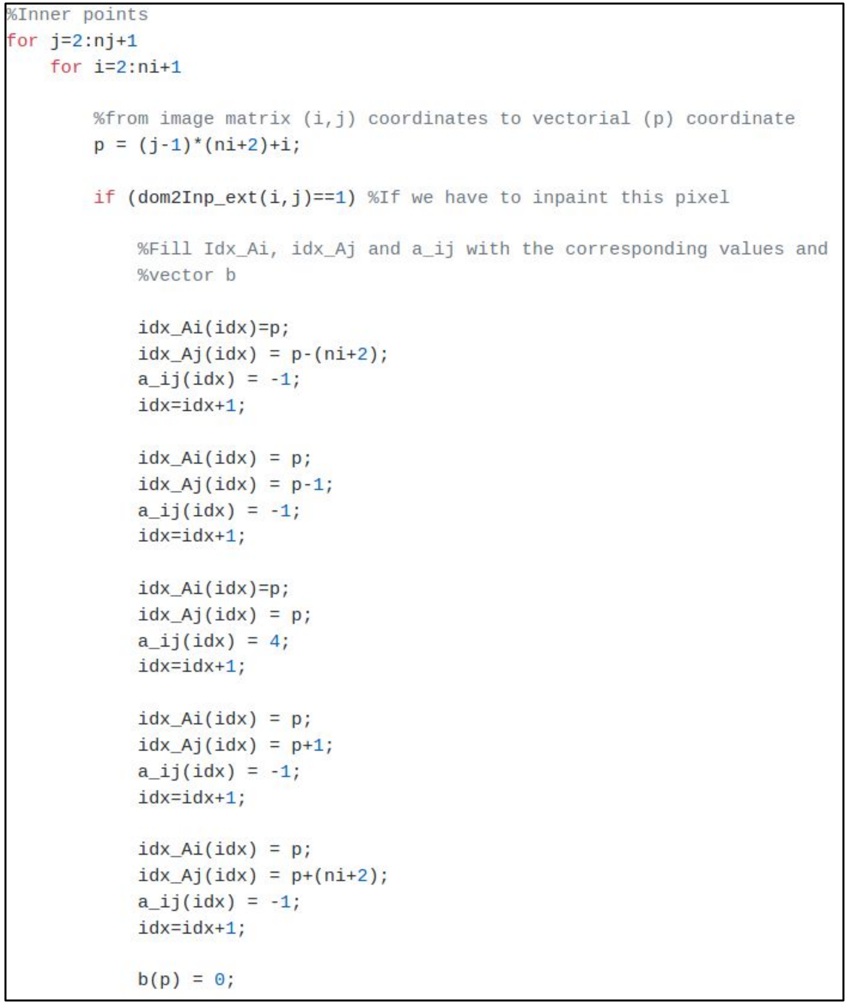
```
%East side boundary conditions
j=nj+2;
for i=1:ni+2
    %from image matrix (i,j) c
    p = (j-1)*(ni+2)+i;

    %Fill Idx_Ai, idx_Aj and a
    %vector b

    idx_Ai(idx)=p;
    idx_Aj(idx) = p;
    a_ij(idx) = 1;
    idx=idx+1;

    idx_Ai(idx) = p;
    idx_Aj(idx) = p-(ni+2);
    a_ij(idx) = -1;
    idx=idx+1;

    b(p) = 0;

end
```



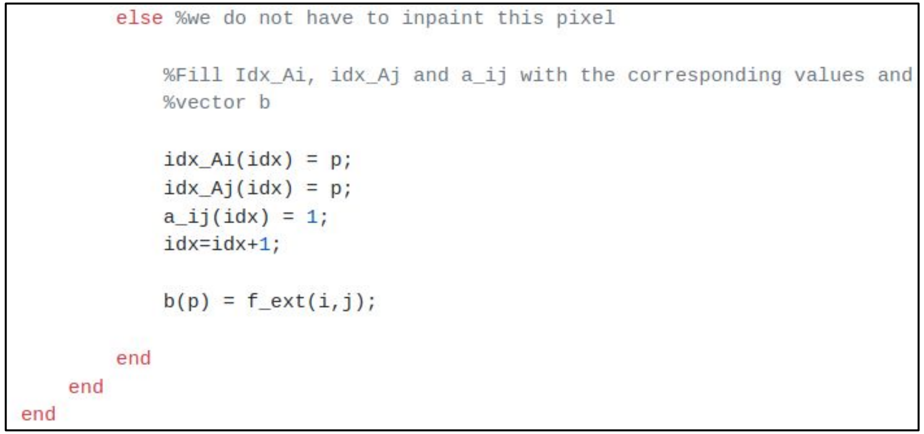
```
%West side boundary conditions
j=1;
for i=1:ni+2
    %from image matrix (i,j) c
    p = (j-1)*(ni+2)+i;

    %Fill Idx_Ai, idx_Aj and a
    %vector b

    idx_Ai(idx)=p;
    idx_Aj(idx) = p;
    a_ij(idx) = 1;
    idx=idx+1;

    idx_Ai(idx) = p;
    idx_Aj(idx) = p+(ni+2);
    a_ij(idx) = -1;
    idx=idx+1;

    b(p) = 0;


end
```

Code Implementation (Task 5, 6)

Inpainting region B and keeping region A same as the original image.

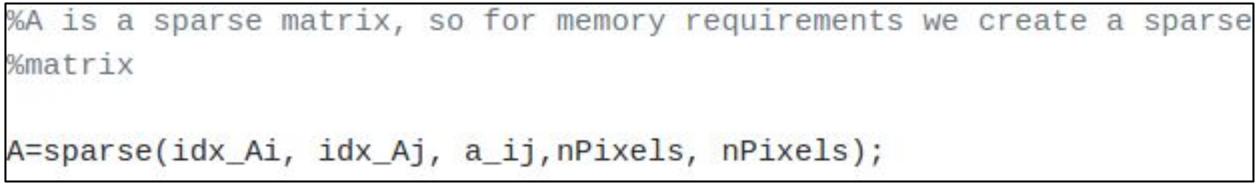


```
%Inner points
for j=2:nj+1
    for i=2:ni+1

        %from image matrix (i,j) coordinates to vectorial (p) coordinate
        p = (j-1)*(ni+2)+i;

        if (dom2Inp_ext(i,j)==1) %If we have to inpaint this pixel

            %Fill Idx_Ai, idx_Aj and a_ij with the corresponding values and
            %vector b

            idx_Ai(idx)=p;
            idx_Aj(idx) = p-(ni+2);
            a_ij(idx) = -1;
            idx=idx+1;

            idx_Ai(idx) = p;
            idx_Aj(idx) = p-1;
            a_ij(idx) = -1;
            idx=idx+1;

            idx_Ai(idx)=p;
            idx_Aj(idx) = p;
            a_ij(idx) = 4;
            idx=idx+1;

            idx_Ai(idx) = p;
            idx_Aj(idx) = p+1;
            a_ij(idx) = -1;
            idx=idx+1;

            idx_Ai(idx) = p;
            idx_Aj(idx) = p+(ni+2);
            a_ij(idx) = -1;
            idx=idx+1;

            b(p) = 0;
```



```
        else %we do not have to inpaint this pixel

            %Fill Idx_Ai, idx_Aj and a_ij with the corresponding values and
            %vector b

            idx_Ai(idx) = p;
            idx_Aj(idx) = p;
            a_ij(idx) = 1;
            idx=idx+1;

            b(p) = f_ext(i,j);

        end
    end
end
```

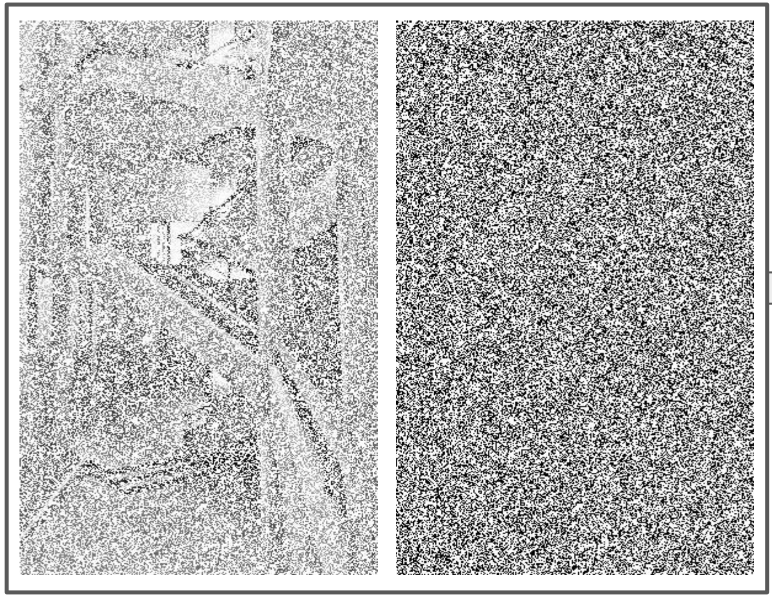
Code Implementation (Task 7: Creating Sparse Matrix)



```
%A is a sparse matrix, so for memory requirements we create a sparse
%matrix

A=sparse(idx_Ai, idx_Aj, a_ij,nPixels, nPixels);
```

→ *nPixels = (ni+2)*(nj+2)*

Results: Black and White images (I)



Some magic...

Results: Black and White images (II)

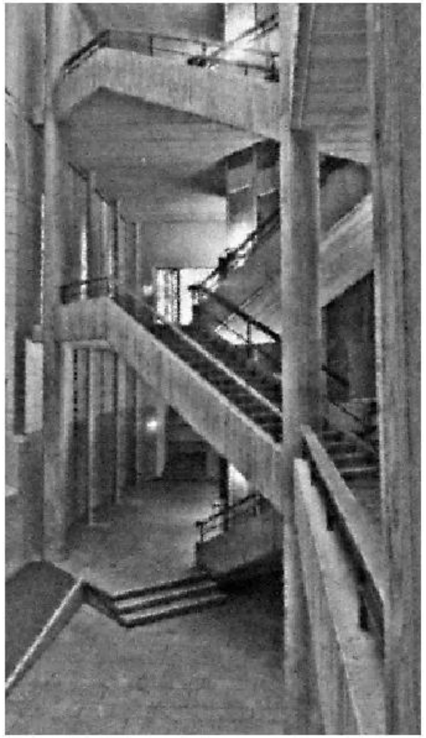


Image 1

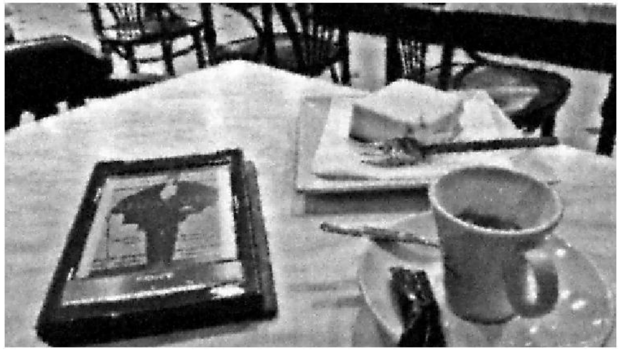


Image 2

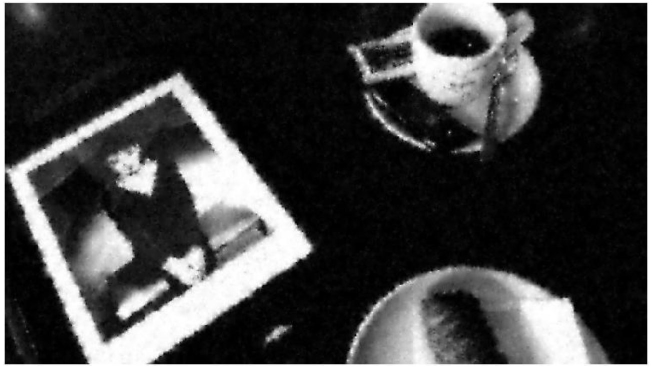


Image 3

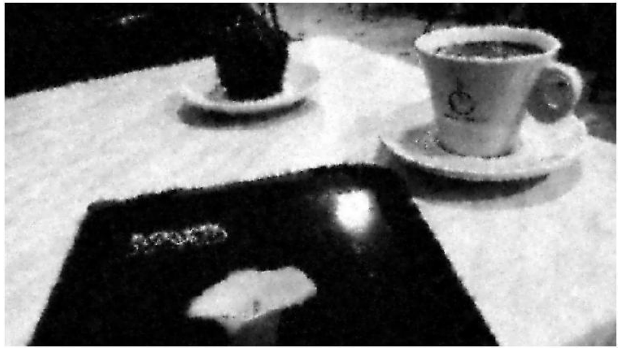


Image 4

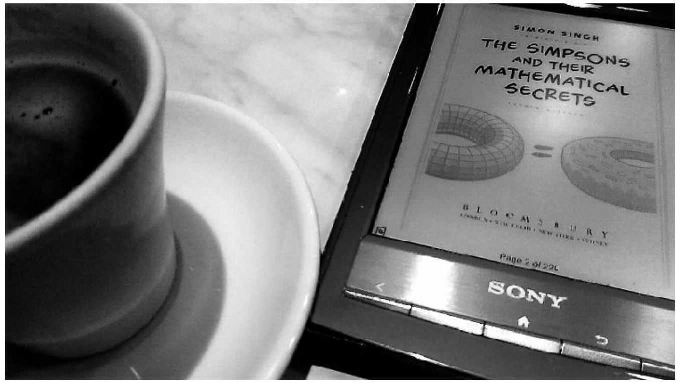
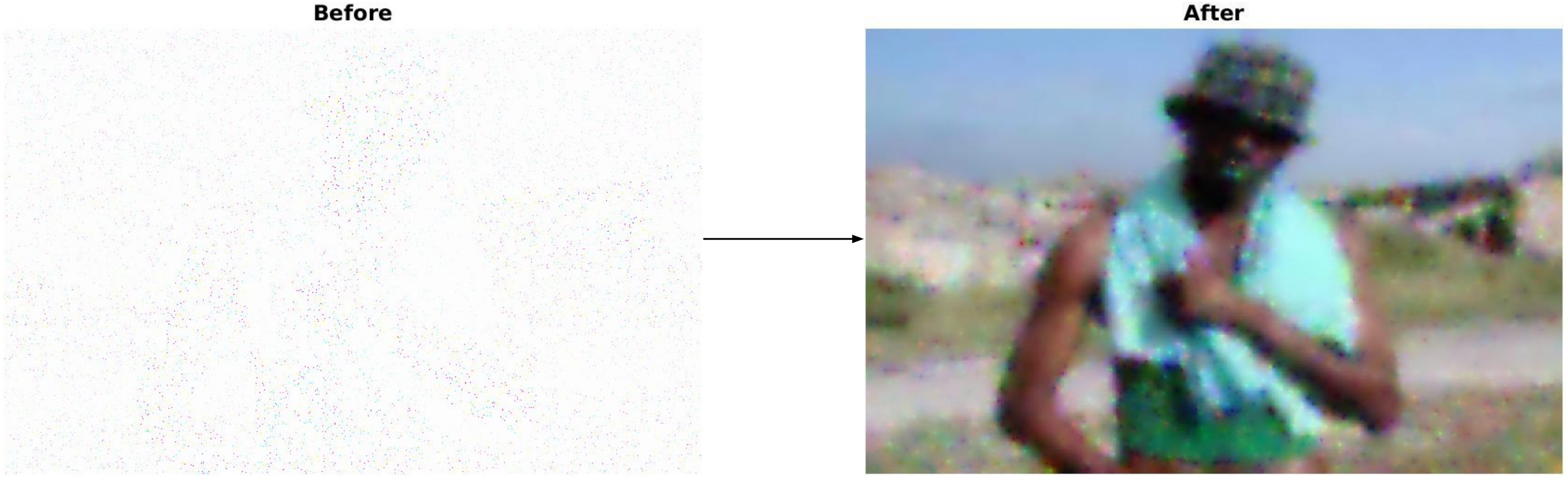


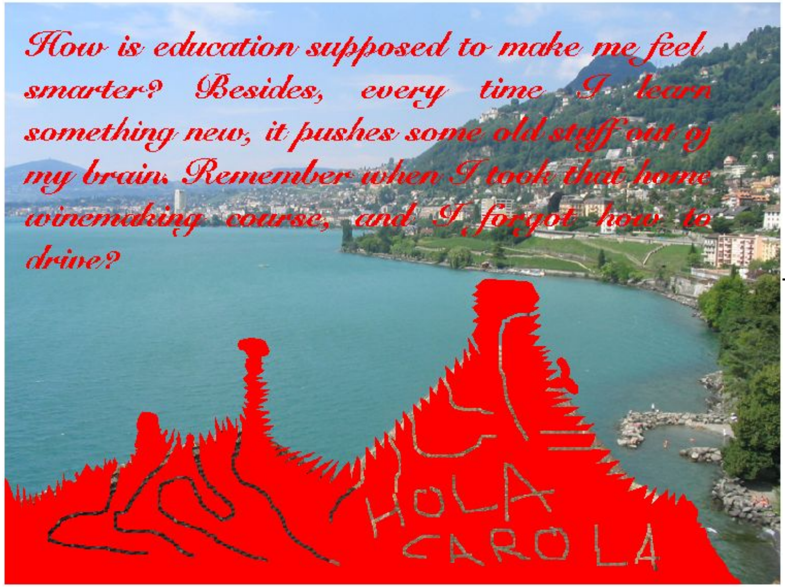
Image 5

Results: Challenge image



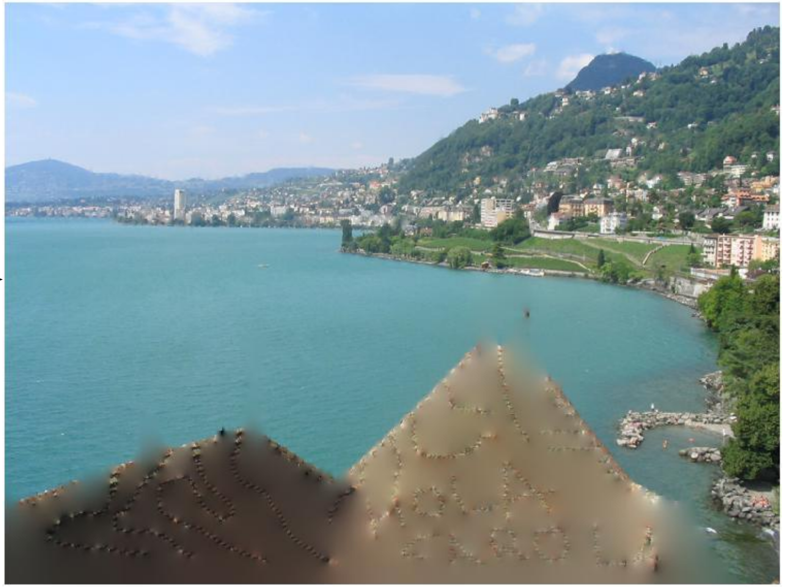
Results: Goal image

Before



How is education supposed to make me feel smarter? Besides, every time I learn something new, it pushes some old stuff out of my brain. Remember when I took that home winemaking course, and I forgot how to drive?

After



Proof that the solution of the Laplace's equation is a solution of the problem

$$\begin{cases} \underset{u \in W^{1,2}(\Omega)}{\arg\min} \int_D |\nabla u(x)|^2 dx, \\ u|_{\partial D} = f \end{cases}$$

· ***f***: ***known***, *damaged image to restore*
· ***u***: ***unknown***, *restored image*

$$J(u) = \int_\Omega |\nabla u|^2 dx$$

$$\boxed{\nabla u(x) = (\frac{\partial u}{\partial x_1}, \frac{\partial u}{\partial x_2}) = (u_{x_1}, u_{x_2})}$$

$$\boxed{|\nabla u(x)|^2 = \left(\sqrt{\left(\frac{\partial u}{\partial x_1}\right)^2 + \left(\frac{\partial u}{\partial x_2}\right)^2}\right)^2 = \left(\frac{\partial u}{\partial x_1}\right)^2 + \left(\frac{\partial u}{\partial x_2}\right)^2 = u_{x_1}^2 + u_{x_2}^2}$$

$$\frac{\mathrm{d}J}{\mathrm{d}u} = -\frac{\partial}{\partial x_1}\frac{\partial |\nabla u(x)|^2}{\partial u_{x_1}} - \frac{\partial}{\partial x_2}\frac{\partial |\nabla u(x)|^2}{\partial u_{x_2}} + \frac{\partial |\nabla u(x)|^2}{\partial u} = 0$$

$$\frac{\mathrm{d}J}{\mathrm{d}u} = -\frac{\partial}{\partial x_1}\frac{\partial (u_{x_1}^2 + u_{x_2}^2)}{\partial u_{x_1}} - \frac{\partial}{\partial x_2}\frac{\partial (u_{x_1}^2 + u_{x_2}^2)}{\partial u_{x_2}} + \frac{\partial (u_{x_1}^2 + u_{x_2}^2)}{\partial u} = 0$$

$$\frac{\mathrm{d}J}{\mathrm{d}u} = -\frac{\partial}{\partial x_1} 2 * u_{x_1} - \frac{\partial}{\partial x_2} 2 * u_{x_2} + 0 = 0$$

$$\boxed{\frac{\mathrm{d}J}{\mathrm{d}u} = \frac{\partial}{\partial x_1}\frac{\partial u}{\partial x_1} + \frac{\partial}{\partial x_2}\frac{\partial u}{\partial x_2} = \frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2} = \Delta u = 0}$$

$$\begin{cases} \Delta u = 0 & \text{in } D, \\ u = f & \text{in } \partial D \end{cases}$$

Discussion & Conclusions

This implementation works well with small regions of lost information (i.e. lost information surrounded by preserved regions of the image).

Big regions of lost information (e.g. lower part of the goal image) are noticeable when inpainted since they are very smooth, leading to unwanted results.

Some sharp edges are smoothened and thus lost. This is due to the use of the L^2 norm, which penalizes too much the gradients corresponding to edges. L^1 would preserve the edges better, but the overall inpainting result will be worse.