M2 - Optimization and Inference Techniques for CV

2. Poisson editing

Group 8

Alex Carrillo
Johnny Nuñez
Guillem Martínez

MSc in Computer Vision

Recap: Week 1

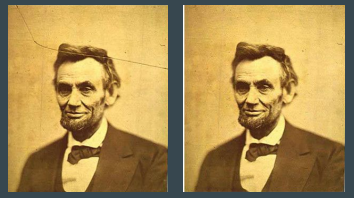
Inpainting:

- Technique in which damaged, deteriorated or missing regions of images are reconstructed by interpolation of surrounding areas

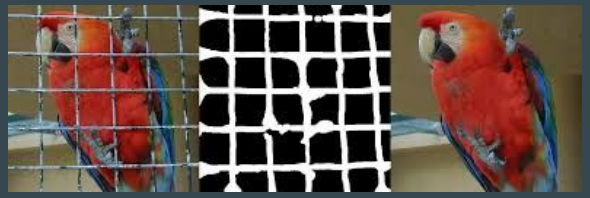
Conclusions and Results

- Notable results for inpainting small regions
- Bigger region, bigger blur in inpainted region, (may occur because):
  - The content available for inpainting is partial
  - The # of neighbours pixels to look at is not enough (4-connectivity)
- The smooth transition of the Laplacian operator is not enough to “hide” a visible region in an optimal way

before after



before after



original mask result

original mask result

Poisson editing: this week's goal

Image A (*destination*)

"copy & paste"

Image B (*source*)

Insert one image (B) into another (A)...

…without introducing any visually unappealing seams

OPTIMAL

Criteria for the blending (Method #1: Importing Gradients)

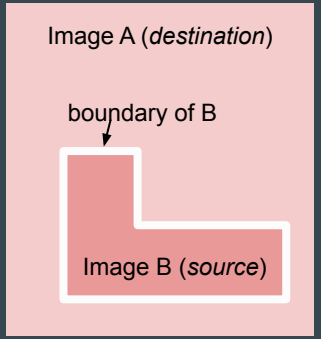
For a seamless “copy & paste” of the inserted image (*source*; B) into the target image (*destination*; A), we want to generate a **new image H** to be pasted, that (visually):

1. Ensures that the color of the inserted image is shifted as part of the environment of the target image
2. Keeps intact all of the details (edges, corners, transitions, etc.) of the inserted image

Preliminary intuition

This (mathematically thinking) translates into:

1. Pixels on H’s boundary should be the same as the pixels of A on that boundary
2. The gradient on the interior of H should be equal to the gradient on the interior of B



Mathematically (Method #1: Importing Gradients)

For a seamless “copy & paste” of the inserted image (*source*; B) into the target image (*destination*; A), we want to generate a **new image H** to be pasted, that (visually):

1. $H(x, y) = A(x, y)$ at each (x, y) of the boundary of B

2. $\nabla H(x, y) = \nabla B(x, y)$ at each (x, y) inside B

   $\Longrightarrow$

   $4H(x, y) - \Sigma H(x + dx, y + dy) = 4B(x, y) - \Sigma B(x + dx, y + dy)$ at each (x, y) inside B

Finite differences and Laplace operator (gradient computation)

Approximate gradients by means of finite differences equations, on each direction *(x, y)*.

Partial derivative

$$\frac{\partial f(x)}{\partial x} \approx \Delta f(x) = \frac{f(x+h) - f(x)}{h} = [h := 1] = f(x+1) - f(x)$$

Forward difference

Implemented in functions *DiFwd()* and *DjFwd()*

Partial derivative

$$\frac{\partial f(x)}{\partial x} \approx \Delta f(x) = \frac{f(x) - f(x-h)}{h} = [h := 1] = f(x) - f(x-1)$$

Backward difference

Implemented in functions *DiBwd()* and *DjBwd()*

2nd partial derivative

$$\frac{\partial^2 f(x)}{\partial x} \approx \frac{\Delta^2 f(x)}{h^2} = \frac{\frac{f(x+h)-f(x)}{h} - \frac{f(x)-f(x-h)}{h}}{h} = \underbrace{f(x+1) - f(x)} - \underbrace{f(x) - f(x-1)}$$

Forward difference

Backward difference

In the code, we can use:

drivingGrad_i = DiBwd(·) – DiFwd(·)

drivingGrad_j = DjBwd(·) – DjFwd(·)

Laplacian operator

$$\Delta f(x) = \sum_{i=1}^{n} \partial^2 f(x) = [\text{in 2D}] = \frac{\partial^2 f}{\partial x} + \frac{\partial^2 f}{\partial y}$$

```
drivingGrad_i = G1_DiBwd( src(:,:,nC), param.hi ) - G1_DiFwd( src(:,:,nC), param.hi );
drivingGrad_j = G1_DjBwd( src(:,:,nC), param.hj ) - G1_DjFwd( src(:,:,nC), param.hj );
driving_on_src = drivingGrad_i + drivingGrad_j;
```

In the code, we can use:

driving_on_src/dst = drivingGRad_i + drivingGrad_j

Code implementation (Method #1: Importing Gradients)

1. Calculate the partial derivatives on each direction for the source image and calculate the central difference

```
% Second order central diference (second parcial derivatives)
drivingGrad_i = sol_DiBwd( src(:,:,nC), param.hi ) - sol_DiFwd( src(:,:,nC), param.hi );
drivingGrad_j = sol_DjBwd( src(:,:,nC), param.hj ) - sol_DjFwd( src(:,:,nC), param.hj );
```

2. Assemble the Laplacian operator by combining the central differences in each axis

```
% Laplacian
driving_on_src = drivingGrad_i + drivingGrad_j;
```

3. Create the gradients mask and compute the Poisson equation

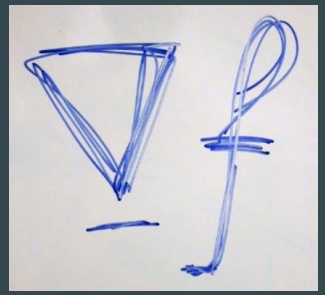
```
driving_on_dst = zeros(size(dst(:,:,1)));
driving_on_dst(mask_dst(:)) = driving_on_src(mask_src(:));

param.driving = driving_on_dst;

dst1(:,:,nC) = sol_Poisson_Equation_Axb(dst(:,:,nC), mask_dst,  param);
```

Goal (Method #2: Mixing Gradients)

With Importing Gradients, no trace of the destination image A is kept inside the source image B.

There are situations where it is desirable to **combine properties of A with those of B**

(i.e. add objects with holes, or partially transparent ones, on top of a textured or cluttered background).



Source image B

Destination image A

Importing Gradients

Mixing Gradients

Criteria for the blending (Method #2: Mixing Gradients)

For a seamless “copy & paste” of the inserted image (*source*; B) into the target image (*destination*; A), we want to generate a **new image H** to be pasted, that (visually):

1. Ensures that the color of the inserted image is shifted as part of the environment of the target image

~~Keeps intact all of the details (edges, corners, transitions, etc.) of the inserted image~~

2. Keeps the most relevant (striking) details of the each, the inserted image or the target image

Preliminary intuition

This (mathematically thinking) translates into:

1. Pixels on H’s boundary should be the same as the pixels of A on that boundary
2. The interior of H should be equal to the maximum gradient (magnitude) between A and B

Mathematically (Method #2: Mixing Gradients)

For a seamless “copy & paste” of the inserted image (*source*; B) into the target image (*destination*; A), we want to generate a **new image H** to be pasted, that (visually):

1. $H(x, y) = A(x, y)$ at each (x, y) of the boundary of B

2. $\nabla H(x, y) = \nabla A(x, y)$ if $|\nabla A(x, y)| > |\nabla B(x, y)|$, otherwise $\nabla H(x, y) = \nabla B(x, y)$ at each (x, y) inside B

Code implementation (Method #2: Mixing Gradients)

1. Calculate the partial derivatives on each direction for source and destination

```
% Calculate the finite partial derivatives of the src
DiBwd_src = sol_DiBwd( src(:,:,nC), param.hi );
DiFwd_src = sol_DiFwd( src(:,:,nC), param.hi );
DjBwd_src = sol_DjBwd( src(:,:,nC), param.hj );
DjFwd_src = sol_DjFwd( src(:,:,nC), param.hj );


% Calculate the finite partial derivatives of the dst
DiBwd_dst = sol_DiBwd( dst(:,:,nC), param.hi );
DiFwd_dst = sol_DiFwd( dst(:,:,nC), param.hi );
DjBwd_dst = sol_DjBwd( dst(:,:,nC), param.hj );
DjFwd_dst = sol_DjFwd( dst(:,:,nC), param.hj );
```

2. Get the maximum gradient for each direction

```
% Get the maxium gradient on each direction
DiBwd_dst = get_max_grad(DiBwd_dst, DiBwd_src, mask_dst, mask_src);
DiFwd_dst = get_max_grad(DiFwd_dst, DiFwd_src, mask_dst, mask_src);
DjBwd_dst = get_max_grad(DjBwd_dst, DjBwd_src, mask_dst, mask_src);
DjFwd_dst = get_max_grad(DjFwd_dst, DjFwd_src, mask_dst, mask_src);
```

```
function [ result ] = get_max_grad( grad_dst, grad_src, mask_dst, mask_src)
% Compute the Forward finite differences with respect to the
% i coordinate only for the 1:end-1 rows. The last row is not replace

    grad_dst_masked = grad_dst(mask_dst(:));
    grad_src_masked = grad_src(mask_src(:));
    grad_mask = abs(grad_dst_masked) <= abs(grad_src_masked);
    grad_dst_masked(grad_mask) = grad_src_masked(grad_mask);
    grad_dst(mask_dst(:)) = grad_dst_masked;
    result = grad_dst;
end
```

3. Calculate the central differences and assemble both directions to form the Laplacian operator

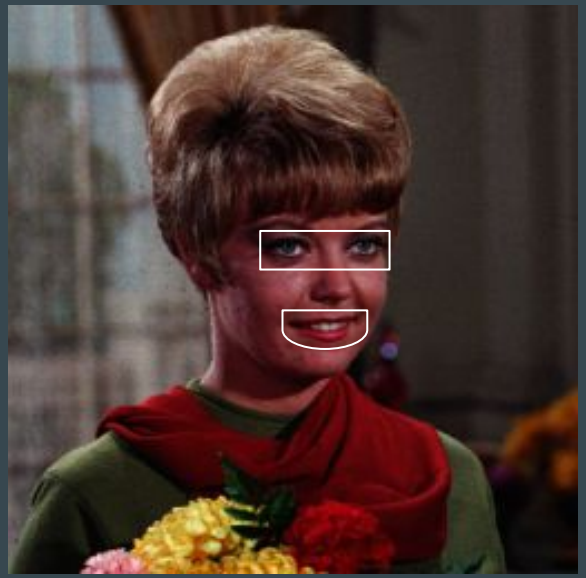
```
drivingGrad_i = DiBwd_dst-DiFwd_dst;
drivingGrad_j = DjBwd_dst-DjFwd_dst;
driving_on_dst = drivingGrad_i + drivingGrad_j;
```

4. Compute the Poisson equation

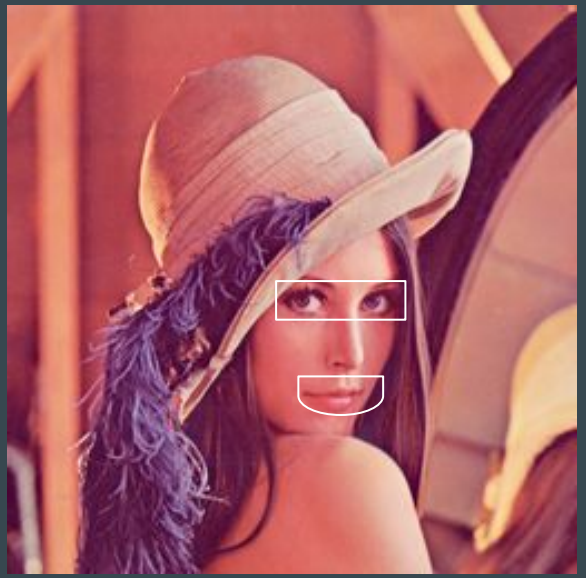
```
param.driving = driving_on_dst;

dst1(:,:,nC) = sol_Poisson_Equation_Axb(dst(:,:,nC), mask_dst,  param);
```

Results: Importing Gradients

Source



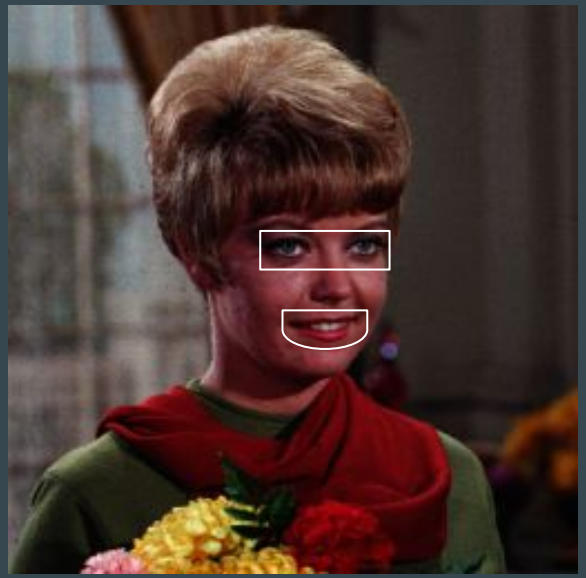
Target



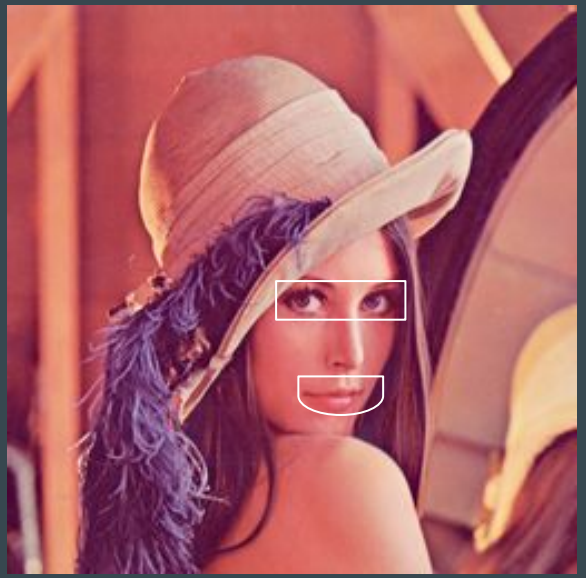
Result

Results: Mixing Gradients

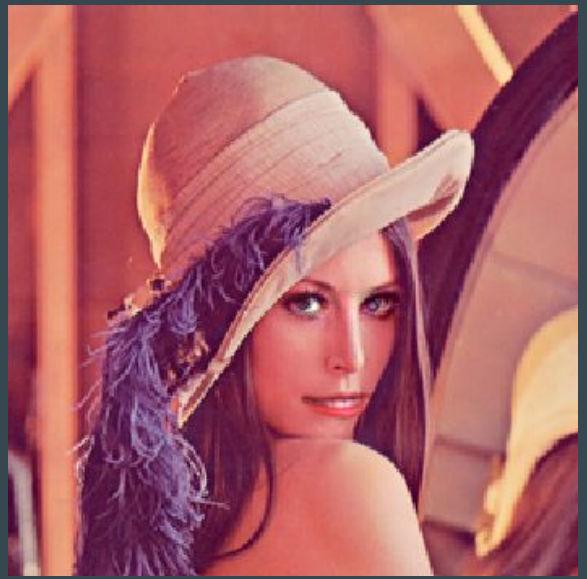
Source



Target



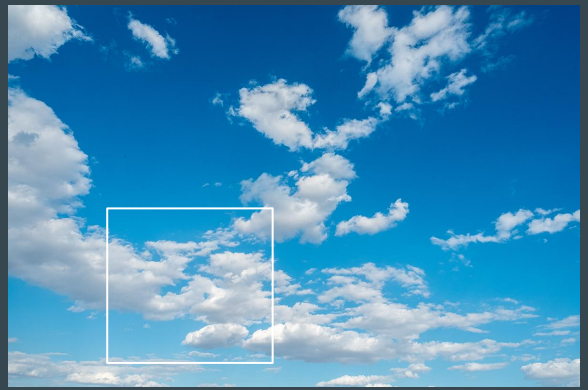
Result



Results: Importing Gradients

Source

Target

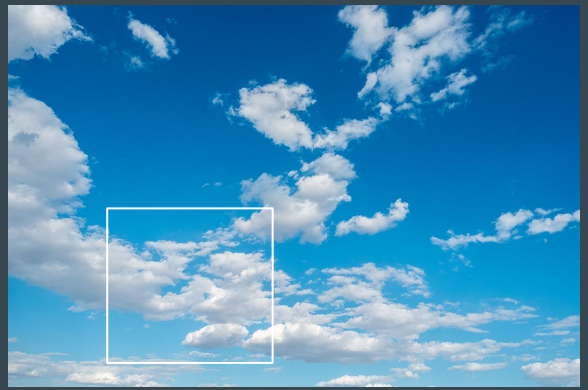


Result

Results: Mixing Gradients

Source

Target



Result

Custom Results: Importing Gradients

Source

Target

Result

Custom Results: Mixing Gradients

Source

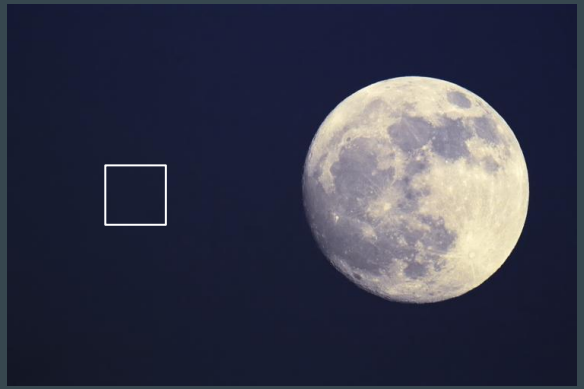
Target

Result

Custom Results: Importing Gradients

Source

Target

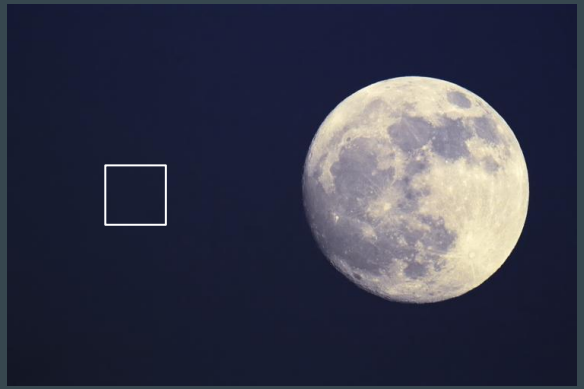


Result

Custom Results: Mixing Gradients

Source

Target



Result

Custom Results: Importing Gradients

Source

Target

Result

Custom Results: Mixing Gradients

Source

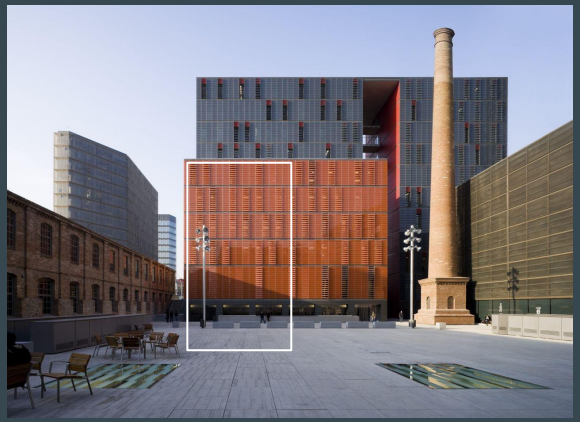
Target

Result

Custom Results: Importing Gradients

Source

Target

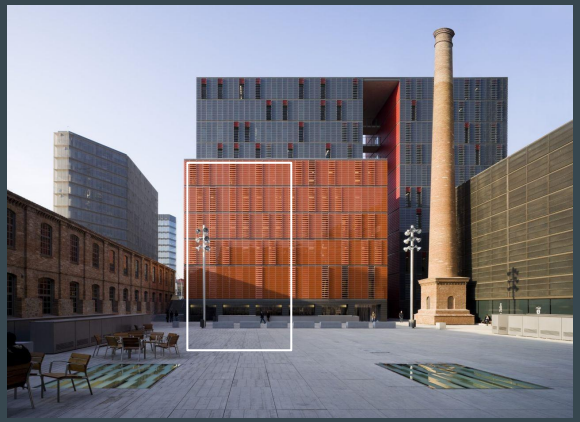


Result

Custom Results: Mixing Gradients

Source

Target



Result

Conclusions

- When pasting objects, importing gradients can be a better approach if the mask is defined properly delimiting the object.
- We obtain good results with Poisson, but there is a blurring in the background, on the cloned image.
- With Mixing gradients we can find unwanted textures when the pasted object is smooth and the destination image is not.
- The issue with the borders of the imported object is resolved by **mixed gradients**.
- When selecting the maximum, the system will not always work, because if the values you want to copy are smaller, you will lose information about the cloned image.