Master in Computer Vision *Barcelona*

Module: M2

Project: Poisson Editing

Team:

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

Task

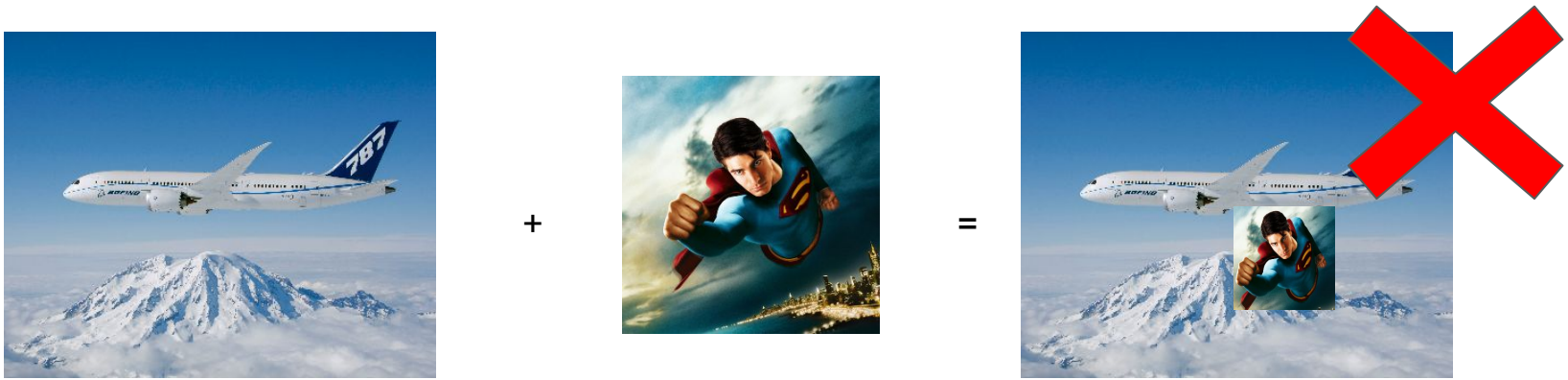
Add a region of a source image to a target image while keeping it photorealistic.

We want to seamlessly import both opaque and transparent source image regions into a destination region.

Problem

But we can’t just crop the source region and paste it into the destination image. The edit is easily noticeable because

1. Precise segmentation masks are required for source image
2. Pixel colors from both images are different



Idea

Human perception system is more sensitive to gradients than intensity values

So we want a new image H that keeps:

- The colors of image A
- The details of image B, i.e. the gradients at each point

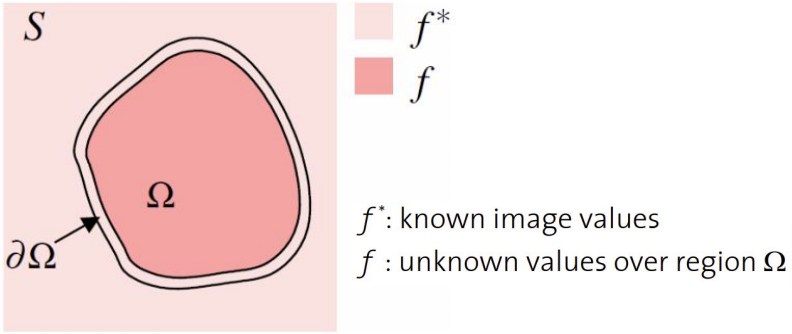
Image A

Image B

∇B (gradient)

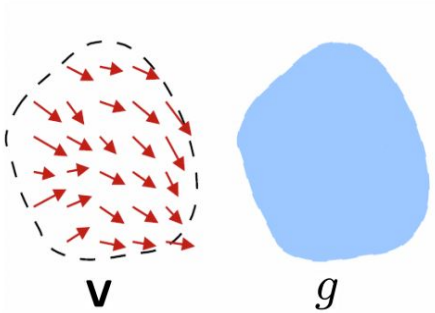
Image H

Method: Guided interpolation



f^*: known image values

f : unknown values over region Ω



$\mathbf{v}$: guided field

$\mathbf{v}$ may be gradient of a function g

Goal: Minimize difference of gradient fields

$$\min_f \int_\Omega \|\nabla f - \mathbf{v}\|^2$$

Solution: Poisson Equation with Dirichlet boundary conditions

$$\nabla^2 f = \operatorname{div} \mathbf{v}, \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

Method: Define Criterias

Given a source image ***B*** and and a target image ***A,*** the impainting of ***B*** in ***A*** follows two criteria:

1. The transition between **A** and **B** must be smooth, that is conserving the colours of image A.
2. The details of B must be kept, that is the gradients must be conserved.

Method: Form Mathematical Equations

Assuming a source image B and a target image A, the resulting image H(x,y) can be obtained in the following way:

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

$$4 * H_{i,j} - (H_{i+1,j} + H_{i-1,j} + H_{i,j+1} + H_{i,j-1}) = 4 * B_{i,j} - (B_{i+1,j} + B_{i-1,j} + B_{i,j+1} + B_{i,j-1})$$

Method: Boundary Conditions (same as week 1)

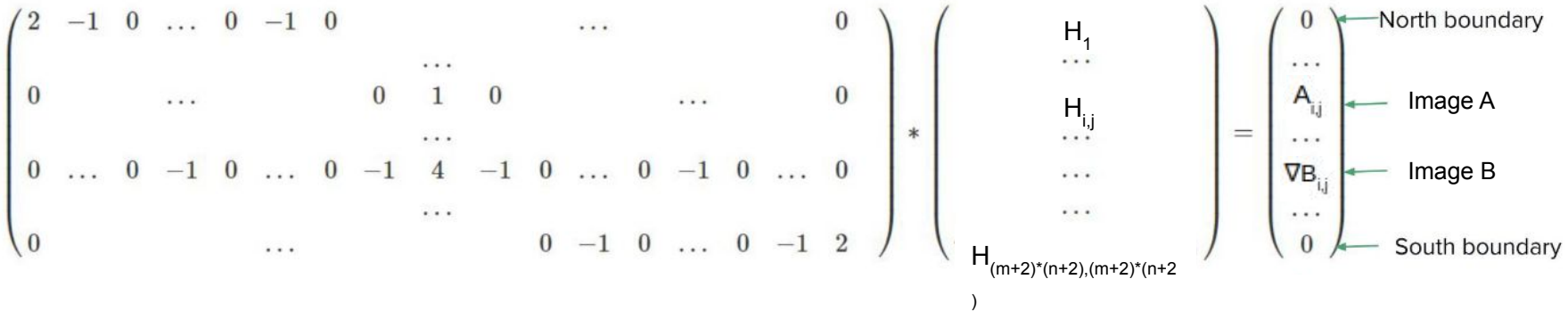
To inpaint the pixels in the image, we add a ghost row/column on each side of the image. This is done so that the same mathematical constraints can be applied on all pixels of the image including the ones on the borders and corners of the original image.

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 Poisson editing problem satisfying the above constraints is obtained by solving the following linear system of equations where A is the target image, B the source image and H the resulting image



Code Reminder: 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 Reminder: Inner points

```
        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
```

We want to keep region A same as the original image, but now we want ∇B at region B… so we need to change this line!!!!

```
%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;
```

New Code: Inner points

```
        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
```

We have ∇B at region B and we keep region A same as the original image.

```
if (isfield(param, 'driving'))
    b(p) = param.driving(i-1,j-1);
else
    b(p) = 0;
end
```

New Code!!

```
%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;
```

Code: Lena/Girl Eyes

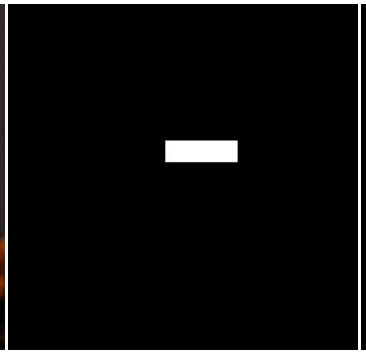
```
%masks to exchange: Eyes
mask_src=logical(imread('mask_src_eyes.png'));
mask_dst=logical(imread('mask_dst_eyes.png'));

output_filename = 'lena_fusion.png';

for nC = 1: nChannels

    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;

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

    param.driving = driving_on_dst;

    dst1(:,:,nC) = G1_Poisson_Equation_Axb(dst(:,:,nC), mask_dst, param);
end
```

Code: Lena/Girl Mouth

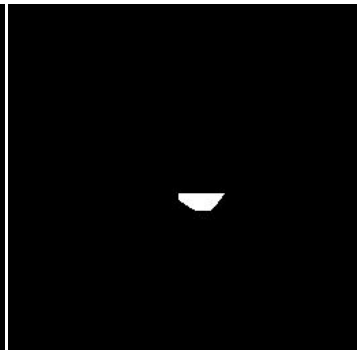
```
%Mouth
%masks to exchange: Mouth
mask_src=logical(imread('mask_src_mouth.png'));
mask_dst=logical(imread('mask_dst_mouth.png'));
for nC = 1: nChannels

    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;

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

    param.driving = driving_on_dst;

    dst1(:,:,nC) = G1_Poisson_Equation_Axb(dst1(:,:,nC), mask_dst,  param);
end
```

Results: Lena/Girl Eyes and Mouth

Girl (src)

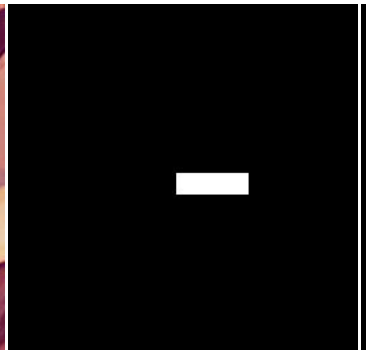


Eyes mask (src)

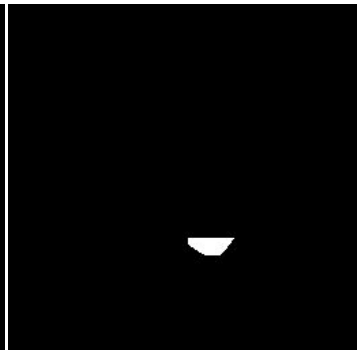


Mouth mask (src)

Lena (dst)



Eyes mask (dst)



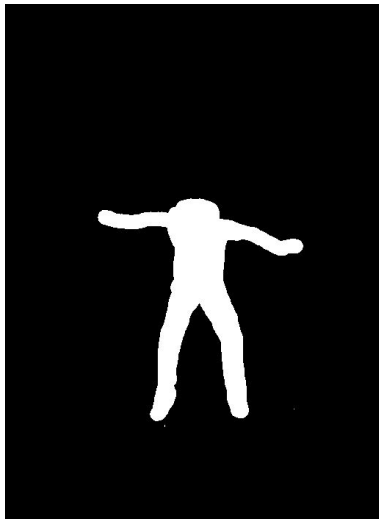
Mouth mask (dst)

Result

Optional: Our own images (I)

Image A

Image B

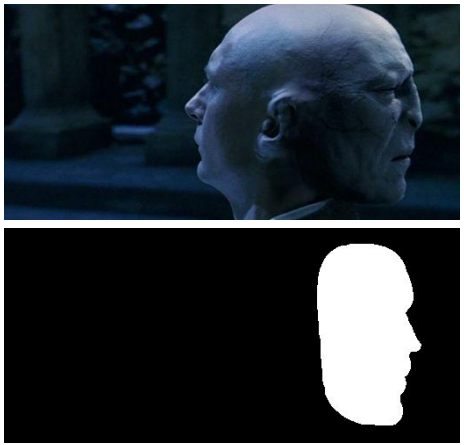


Binary mask

Image H

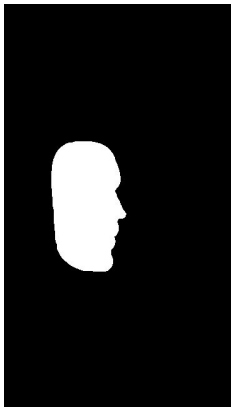
Optional: Our own images (II)

Image A



Mask A

Image B



Mask B

Image H

Optional: Our own images (III)

Image A

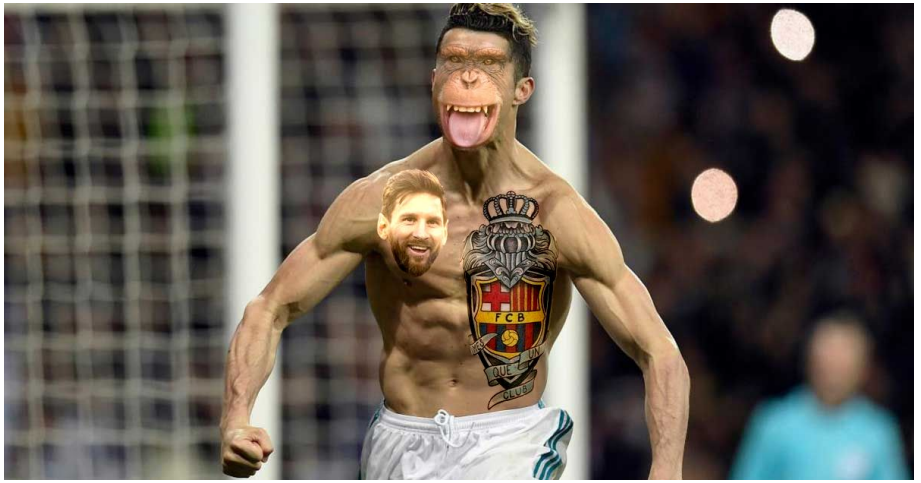


Image H

Image B (I)

Image B (II)

Image B (III)

Optional: Mixing gradients (I)

Until now, no trace of the target image remained inside the region we wanted to fill.

However, there are situations where it is desirable to combine properties of the target image A with those of the source image B, for example to add objects with holes, or partially transparent ones, on top of a textured or cluttered background.

With the mixing gradients algorithm, we retain the stronger of the variations in image A or in image B, using the following guidance field:

$$\mathbf{v}(\mathbf{x}) = \begin{cases} \nabla A\,(x) & \text{if } |\nabla A\,(x)\ | > |\nabla B(x)| \\ \nabla B(x) & \text{otherwise} \end{cases}$$

Optional: Mixing gradients (II)

```
for nC = 1: nChannels

    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;

    driving_on_dst = zeros(size(dst(:,:,1)));

    if mixing_gradients

        % --- start optional: mixing gradients

        drivingGrad_i_dst = G1_DiBwd(dst(:,:,nC), param.hi) - G1_DiFwd(dst(:,:,nC), param.hi);
        drivingGrad_j_dst = G1_DjBwd(dst(:,:,nC), param.hj) - G1_DjFwd(dst(:,:,nC), param.hj);
        driving_on_dst_aux = drivingGrad_i_dst + drivingGrad_j_dst;

        driving_mix = cat(3, driving_on_src(mask_src(:)), driving_on_dst_aux(mask_dst(:)));
        [~, idx] = max(abs(driving_mix), [], 3);

        driving_on_dst(mask_dst(:)) = driving_on_src(mask_src(:)).*(idx==1) + driving_on_dst_aux(mask_dst(:)).*(idx==2);

        % --- end optional: mixing gradients
    else
        driving_on_dst(mask_dst(:)) = driving_on_src(mask_src(:));
    end

    param.driving = driving_on_dst;

    dst1(:,:,nC) = G1_Poisson_Equation_Axb(dst(:,:,nC), mask_dst, param);
end
```

Optional: Mixing gradients (III)

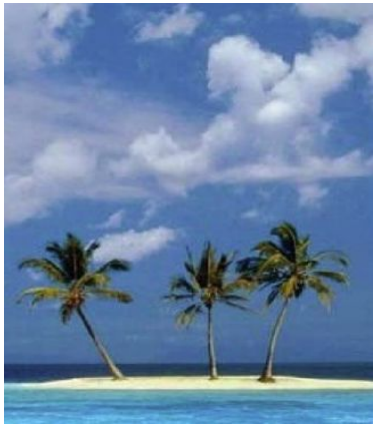
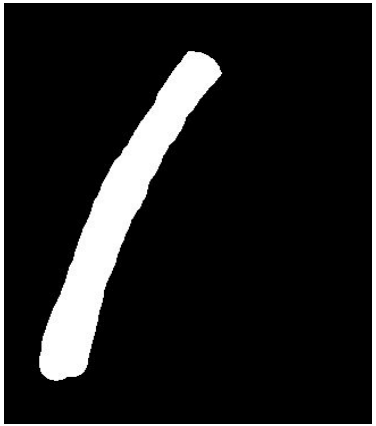


Image A

Image B



Mask

Simple Poisson Editing

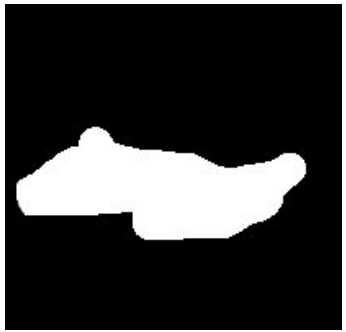
Mixing Gradients

Optional: Mixing gradients (IV)

Image A

Mask A

Image B



Mask B

Simple Poisson Editing

Mixing Gradients

Optional: Gradient Descent

```
function [x] = grad_descent(A, b, x)

r = b - A*x;
err = norm(r);

tolerance = 1e-2;
i = 1;
while err > tolerance
    q = A*r;
    alpha = (transpose(r)*r) /(transpose(q)*r);
    r = b - A*x;
    x = x + alpha*r;

    err = norm(r);
    fprintf('iter %03d, error: %.6f\n', i, err);

    i = i+1;
end
```

Result

Discussion & Conclusions

After seeing the results, we can conclude that Poisson editing is a good tool when it comes to solving the problem of seamless editing and seamless cloning. This technique, though, has some limitations; for instance: the detail (gradient) of the target image is completely obliterated in the areas to be filled with information of the source image.

Another issue is the case of the source image being transparent or with holes that should reveal detail of the target image, situation in which again the detail in the target image is deleted.

A solution that has proved to overcome these limitations is the mixing of gradients, where the strongest gradient (between target and source images) in an inpainted area is the one to be kept.