Poisson Image Editing

This lab focuses on the gradient domain blending based on poisson equation. The goal of this part is to create a blended image that is partially cloned from a (source) image. You can learn the application of poisson equation and how to solve a sparse linear system. We will follow the technique described in the following paper:

'Poisson Image Editing", Perez, P., Gangnet, M., Blake, A. SIGGRAPH 2003

It discusses the *importing gradients* method in section 3, which you are strongly encouraged to read prior to starting this part of the project. The goal is seamlessly blend two images together automatically given the blending region. As shown in the figure 1, the red line mark the blending region.

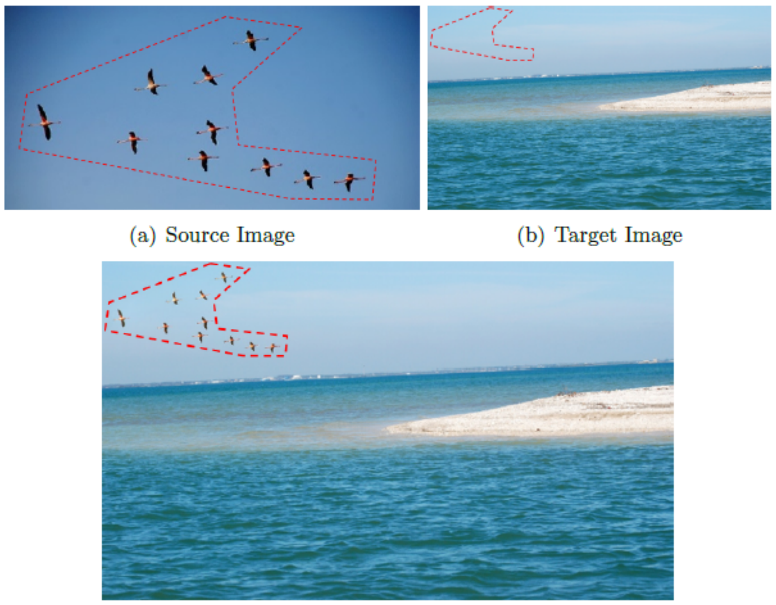
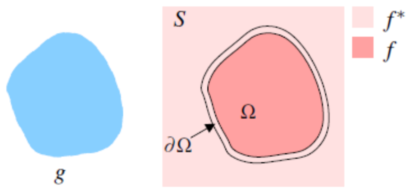


Figure 1: Gradient Domain Blending

Let's first define the image we're changing as the target image, the image we're cutting out and pasting as the source image, the pixels in target image that will be blended with source image as the replacement pixels. The key idea of the gradient domain blending is to apply the gradient of the source image to the target image's replacement pixels, but keep other pixels. For continua image function, it can summarize as the following equation:

$$\min_f \iint_\Omega |\nabla f - \nabla g|^2, \qquad \text{with} \quad f|_{\partial\Omega} = f^\star|_{\partial\Omega},$$

where f is the blending image function, $f^\star$ is the target image function, g is the source image function, Ω is the blending region, and $\partial\Omega$ is the boundary of blending region, that is $\partial\Omega = \{p \in S \setminus \Omega : N_p \cap \Omega \neq \varnothing\}$, where S is the source image domain and N_p is the set of 4-connected neighbors of pixel p in S.



In the discrete pixel grid, the previous equation admits the form,

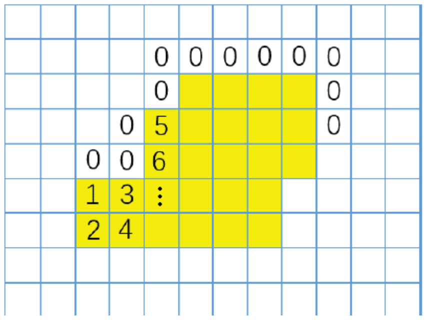
$$\min_{f|_\Omega} \sum_{p\in\Omega} \sum_{q\in N_p\cap\Omega} ((f_p - f_q) - (g_p - g_q))^2, \quad \text{with} f|_{\partial\Omega} = f^\star|_{\partial\Omega}$$

The solution $\{f_p\}_{p\in\Omega}$ of the above minimization problem satisfies the following set of linear equations:

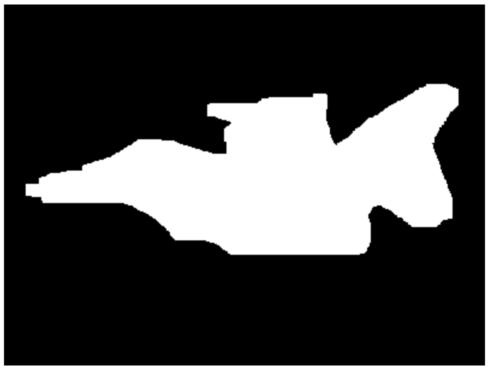
$$|N_p| f_p - \sum_{q\in N_p \cap \Omega} f_q = \sum_{q\in N_p \cap \partial\Omega} f^\star_q + \sum_{q\in N_p} (g_p - g_q), \qquad \forall p \in \Omega$$

Our task is to solve all $\{f_p\}_{p\in\Omega}$ from the set of linear equations. If we form all $\{f_p\}_{p\in\Omega}$ as a vector x, then the set of linear equations can be converted to the form $Ax = b$, whose solution can be efficiently computed with in `Matlab` using the command: `x = A\b` .

The replacement pixels' intensity are solved by the linear system $Ax = b$. But not all the pixels in target image need to be computed. Only the pixels mask as 1 in the mask image will be used to blend. In order to reduce the number of calculations, you need to index the replacement pixels so that each element in x represents one replacement pixel. As shown in the figure below, the yellow locations are the replacement pixels (indexed from top to bottom and from left to right).



The images for the following lab are attached below:



Your Function

Save Reset MATLAB Documentation (https://www.mathworks.com/help/)

```
function [seamless,A] = PoissonImageEditing;

% read images
target= im2double(imread('target_3.jpg'));
source= im2double(imread('source_3.jpg'));
```

```matlab

% mask image
mask=imread('mask_3.bmp');

% image offsets
row_offset=20;
col_offset=40;


% N: Number of pixels in the mask
N=sum(mask(:));

% enumerating pixels in the mask
mask_id = zeros(size(mask));
mask_id(mask) = 1:N;

% neighborhood size for each pixel in the mask
[ir,ic] = find(mask);

Np = zeros(N,1);

for ib=1:N

    i = ir(ib);
    j = ic(ib);

    Np(ib)=  double((row_offset+i> 1))+ ...
             double((col_offset+j> 1))+ ...
             double((row_offset+i< size(target,1))) + ...
             double((col_offset+j< size(target,2)));
end


% compute matrix A

% your CODE begins here
% start with Np along diag
% add at most 4 -1s
% in case less than four, boundary values are already added (to b!)

i = 1:N;
j = 1:N;
A = sparse(i,j,Np,N,N,4*N);

for p = 1:N
  % get row and column of current pixel
  row = ir(p);
  col = ic(p);

  % setup row vector to enter in (sparse) matrix
  v = A(p,:);



  % now check four cases
  % check right (same row, col+1)
  % for id:
  %   numeral_in_mask = sub2ind(size(mask),row,col+1)
  %   id = mask_id(numeral_in_mask)
  if mask(row,col+1) ~= 0
    numeral_in_mask = sub2ind(size(mask),row,col+1);
    right_id = mask_id(numeral_in_mask);
    v(right_id) = -1;
  end
  % check up (row+1, same col):
  if mask(row+1,col) ~= 0
    numeral_in_mask = sub2ind(size(mask),row+1,col);
    up_id = mask_id(numeral_in_mask);
    v(up_id) = -1;
    v(up_id) = -1;
  end
  % check left (same row, col-1):
  if mask(row,col-1) ~= 0
    numeral_in_mask = sub2ind(size(mask),row,col-1);
    left_id = mask_id(numeral_in_mask);
    v(left_id) = -1;
  end
  % check down (row-1, same col):
  if mask(row-1,col) ~= 0
    numeral_in_mask = sub2ind(size(mask),row-1,col);
    down_id = mask_id(numeral_in_mask);
    v(down_id) = -1;
  end

  % update A:

  A(p,:) = v;

end

% your CODE ends here



% output intialization
seamless = target;


for color=1:3 % solve for each colorchannel

    % compute b for each color
    b=zeros(N,1);

    for ib=1:N

    i = ir(ib);
    j = ic(ib);


      if (i>1)
          b(ib)=b(ib)+ target(row_offset+i-1,col_offset+j,color)*(1-mask(i-1,j))+...
                          source(i,j,color)-source(i-1,j,color);
      end

      if (i<size(mask,1))
          b(ib)=b(ib)+  target(row_offset+i+1,col_offset+j,color)*(1-mask(i+1,j))+ ...
                           source(i,j,color)-source(i+1,j,color);
      end

      if (j>1)
          b(ib)= b(ib) +  target(row_offset+i,col_offset+j-1,color)*(1-mask(i,j-1))+...
                           source(i,j,color)-source(i,j-1,color);
      end


      if (j<size(mask,2))
          b(ib)= b(ib)+ target(row_offset+i,col_offset+j+1,color)*(1-mask(i,j+1))+...
                         source(i,j,color)-source(i,j+1,color);
      end




    end


     % solve linear system A*x = b;
    % your CODE begins here
    % your CODE begins here

    x = A\b;
    % your CODE ends here






    % impaint target image

     for ib=1:N
           seamless(row_offset+ir(ib),col_offset+ic(ib),color) = x(ib);
     end

end


figure(1), imshow(target);
figure(2), imshow(seamless);
```

Code to call your function

Reset

```
[seamless,A] = PoissonImageEditing;
```

Run Function

Previous Assessment: All Tests Passed

Submit

✓ **Test 1**