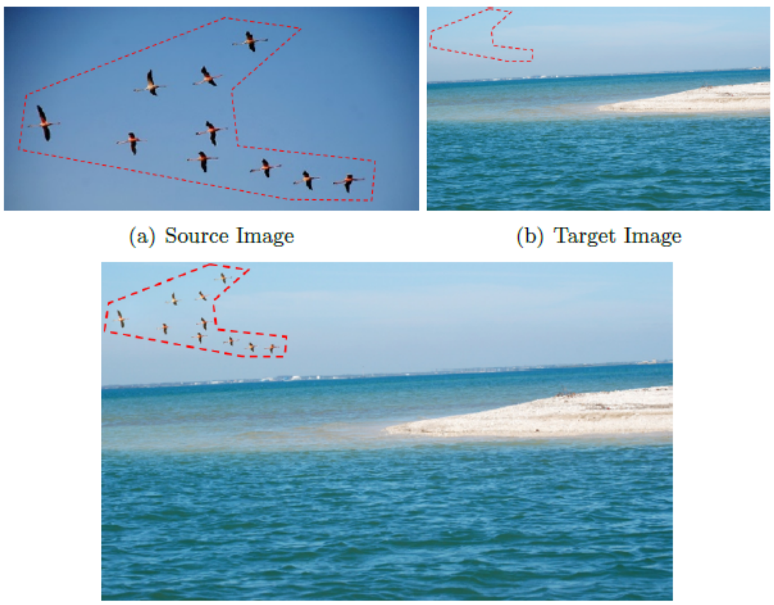
Poisson image editing - Mixing gradients

This project 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 the 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 *mixing 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.



(a) Source Image (b) Target Image

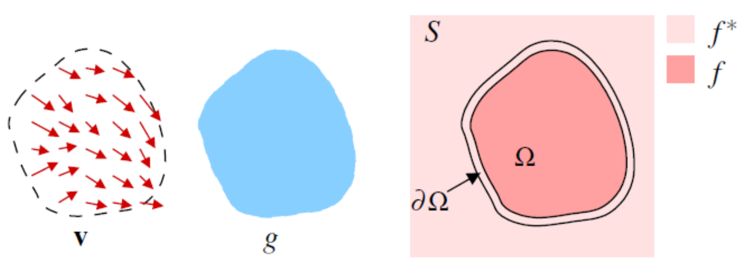
(c) Result

Figure 1: Gradient Domain Blending

Let's first define the image we're changing as the *target i*mage, 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 a so-called guidance vector field v, which might be or not the gradient field of a source function g, to the target image's replacement pixels, but keep other pixels. For continua image function, it can be summarized as the following equation:

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

where f is the blending image function, $f^\star$ is the target image function, v is the guidance vector field, 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:

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

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

$$|N_p| f_p - \sum_{q \in N_p \cap \Omega} f_q = \sum_{q \in N_p \cap \partial\Omega} f^\star_q + \sum_{q \in N_p} v_{pq} \qquad \forall p \in \Omega$$

With the importing gradients method we described in week 7's lab, no trace of the destination image $f^\star$ is kept inside Ω. However, there are situations where it is desirable to combine properties of $f^\star$ with those of g, for example to add objects with holes, or partially transparent ones, on top of a textured or cluttered background. According to the mixing gradients approach, at each point of Ω, we retain the stronger of the variations in $f^\star$ or in g, using the following guidance field (mixing gradients method):

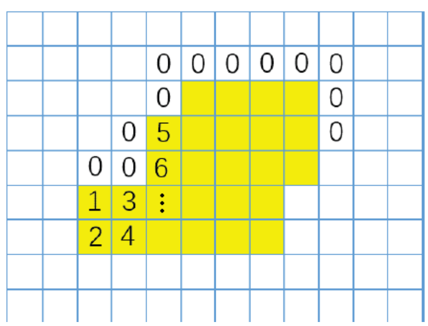
$$v(x) = \begin{cases} \nabla f^\star(x), & \text{if } |\nabla f^\star(x)| \geq |\nabla g(x)| \\ \nabla g(x), & \text{if } |\nabla f^\star(x)| < |\nabla g(x)| \end{cases}$$

for all $x \in \Omega$. The discrete counterpart, is simply, given by

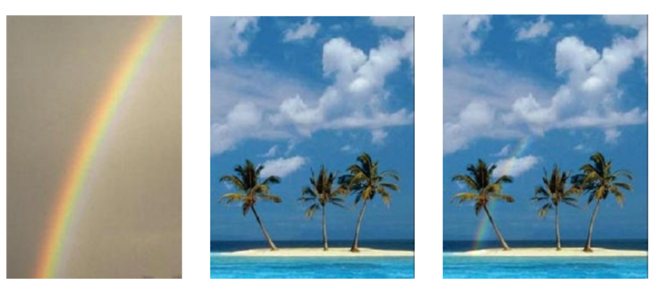
$$v_{pq} = \begin{cases} f^\star_p - f^\star_q, & \text{if } |f^\star_p - f^\star_q| \geq |g_p - g_q| \\ g_p - g_q, & \text{if } |f^\star_p - f^\star_q| < |g_p - g_q| \end{cases}$$

for all $p \in \Omega$ and $q \in N_p$.

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).

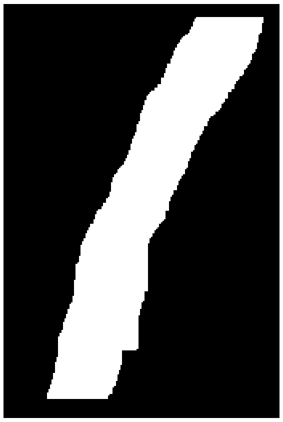
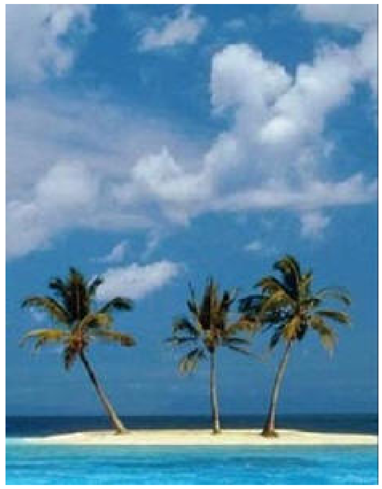


Your results should look like this:



Source Target Edited

The images for this assignment are attached below:

Your Function

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

```
function I = PoissonMixingGradients

% read images
target= im2double(imread('target_2.jpg'));
source= im2double(imread('source_2.jpg'));
mask=imread('mask_2.bmp');

row_offset=130;
col_offset=10;

source_scale=0.6;

source =imresize(source,source_scale);
mask =imresize(mask,source_scale);


% YOUR CODE STARTS HERE

  % N: Number of (nonzero!) 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
  % find gets row,column of nonzero elements
  [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
```

```
% 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;
  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

% 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)
```

```
      fpq = target(row_offset+i,col_offset+j,color)-...
            target(row_offset+i-1,col_offset+j,color);
      gpq = source(i,j,color)-source(i-1,j,color);
      wpq = [fpq,gpq];
      % find max abs
      [~,idx] = sort(abs(wpq));
      vpq = wpq(idx(2));
      b(ib)=b(ib)+ target(row_offset+i-1,col_offset+j,color)*(1-mask(i-1,j))+...
                      vpq;
  end

  if (i<size(mask,1))
      fpq = target(row_offset+i,col_offset+j,color)-...
            target(row_offset+i+1,col_offset+j,color);
      gpq = source(i,j,color)-source(i+1,j,color);
      wpq = [fpq,gpq];
      % find max abs
      [~,idx] = sort(abs(wpq));
      vpq = wpq(idx(2));
      b(ib)=b(ib)+  target(row_offset+i+1,col_offset+j,color)*(1-mask(i+1,j))+ ...
                       vpq;
  end

  if (j>1)
      fpq = target(row_offset+i,col_offset+j,color)-...
            target(row_offset+i,col_offset+j-1,color);
      gpq = source(i,j,color)-source(i,j-1,color);
      wpq = [fpq,gpq];
      % find max abs
      [~,idx] = sort(abs(wpq));
      vpq = wpq(idx(2));
      b(ib)= b(ib) +  target(row_offset+i,col_offset+j-1,color)*(1-mask(i,j-1))+...
                       vpq;
  end


  if (j<size(mask,2))
      fpq = target(row_offset+i,col_offset+j,color)-...
            target(row_offset+i,col_offset+j+1,color);
      gpq = source(i,j,color)-source(i,j+1,color);
      wpq = [fpq,gpq];
      % find max abs
      [~,idx] = sort(abs(wpq));
      vpq = wpq(idx(2));
      b(ib)= b(ib)+ target(row_offset+i,col_offset+j+1,color)*(1-mask(i,j+1))+...
                     vpq;
  end




end


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

x = A\b;
% your CODE ends here






% impaint target image

 for ib=1:N
       seamless(row_offset+i(ib),col_offset+j(ib),color)=x(ib);
```

```
            seamless(row_offset+ir(ib),col_offset+ic(ib),color) = x(ib);
        end

    end

    I = seamless;
    figure(1), imshow(target);
    figure(2), imshow(I);

% YOUR CODE ENDS HERE




end
```

Code to call your function

Reset

```
I = PoissonMixingGradients;
```

Run Function

Previous Assessment: All Tests Passed

Submit

Test 1