Texture flattening with Poisson image editing

This project focuses on the gradient domain blending based on poisson equation. The goal of this part is to flatten the texture of an image (or of a selected region of an image) based on the Poisson equation. We will follow the technique described in the following paper:

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

It discusses the *texture flattening* method in section 4, which you are strongly encouraged to read prior to starting this part of the project.

Recall from par 1, that 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:

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

In the previous part of this project, the guidance field depended, partly or wholly, on the gradient field of a source image g. Alternatively, in-place image transformations can be defined by using a guidance field depending entirely on the original image (target image). In the texture flattening approach, the image gradient $\nabla f^\star$ is passed through a sparse sieve that retains only the most salient features:

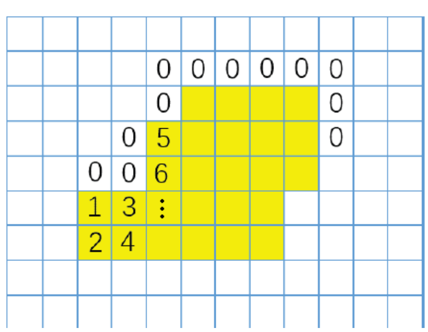
$$v(x) = M(x)\nabla f^\star(x)$$

where M is a binary mask turned on at a few locations of interest. A good choice $f$or M is an edge detector, in which case the discrete counterpart is given by

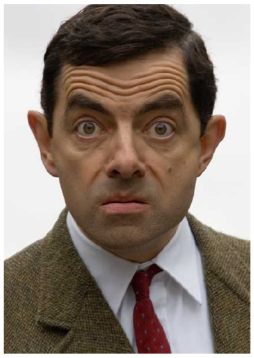
$$v_{pq} = \begin{cases} f^\star_p - f^\star_q, & \text{if } M_p = 1 \text{ or } M_q = 1 \\ 0, & \text{otherwise} \end{cases}$$

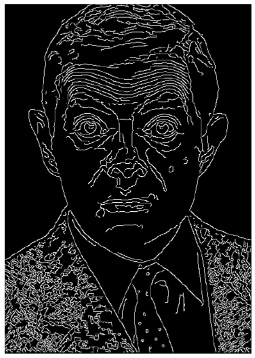
for all $p \in \Omega$ and $q \in N_p$.

As before, 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).



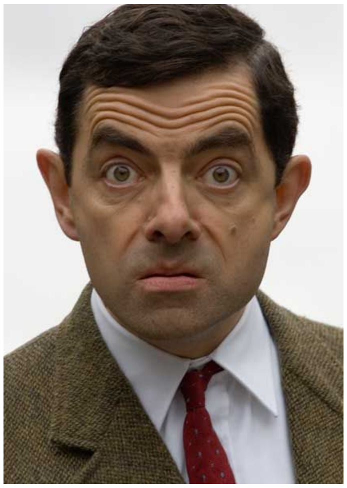
Your results should look like this:

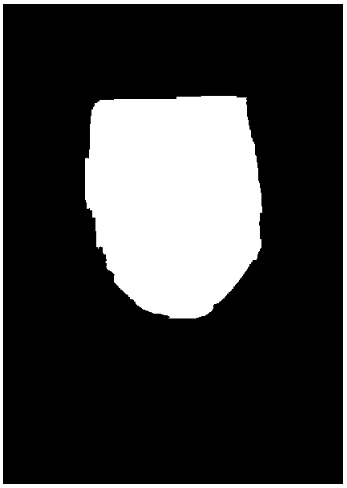

Original


Edge map


Flattened texture

The images for the assignment are attached below:





Your Function

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

```
function I = TextureFlattening

% read image and mask
target = im2double(imread('bean.jpg'));
mask = imread('mask_bean.bmp');

% edge detection
Edges = edge(rgb2gray(target),'canny',0.1);



N=sum(mask(:));  % N: Number of unknown pixels == variables



% YOUR CODE STARTS HERE

  % 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);

```

```matlab
for ib=1:N

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

    Np(ib)=  double((i> 1))+ double((j> 1))+ ...
             double((i< size(target,1))) + double((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
I = 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(i,j,color)-target(i-1,j,color);
          Mp = Edges(i,j);
          Mq = Edges(i-1,j);
          M = Mp*Mq;
          vpq = M*fpq;
          b(ib)=b(ib)+ target(i-1,j,color)*(1-mask(i-1,j))+ vpq;
      end

      if (i<size(mask,1))
          fpq = target(i,j,color)-target(i+1,j,color);
          Mp = Edges(i,j);
          Mq = Edges(i+1,j);
          M = Mp*Mq;
          vpq = M*fpq;
          b(ib)=b(ib)+  target(i+1,j,color)*(1-mask(i+1,j))+vpq;
      end

      if (j>1)
          fpq = target(i,j,color)-target(i,j-1,color);
          Mp = Edges(i,j);
          Mq = Edges(i,j-1);
          M = Mp*Mq;
          vpq = M*fpq;
          b(ib)= b(ib) +  target(i,j-1,color)*(1-mask(i,j-1))+vpq;
      end


      if (j<size(mask,2))
          fpq = target(i,j,color)-target(i,j+1,color);
          Mp = Edges(i,j);
          Mq = Edges(i,j+1);
          M = Mp*Mq;
          vpq = M*fpq;
          b(ib)= b(ib)+ target(i,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
       I(ir(ib), ic(ib),color) = x(ib);
     end

end
```

```
% YOUR CODE ENDS HERE

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

end
```

Code to call your function

Reset

```
I = TextureFlattening;
```

Run Function

Your request timed out. If the problem persists, check your code for an infinite loop or improve your code's efficiency.

Previous Assessment: Incorrect

Submit

Test 1