Computational Photography
Project 4, Fall 2014

October 30, 2014

This project is about image manipulation using the gradient domain and graph cut optimization. The project needs to be turned in by uploading on ILIAS. The deadline is November 13, 14:00.

Poisson Image Editing

The goal of this part is to implement Poisson image editing and demonstrate it using different scenarios.

1. **(2 points)** Implement a Matlab function that solves the Poisson equation. Your function should take the following inputs: a greyscale *target image*, the desired *gradient field*, and a *binary mask*. The output is a greyscale image. To deal with color images, you will need to invoke the Poisson solver separately for each color channel.

   Your Poisson solver should assume that the target, the gradient field, and the mask have the same pixel resolution. The mask determines for each pixel whether it is a boundary pixel or an unknown pixel that should be solved for. Boundary pixels are simply copied from the target to the output. The values of the unknown pixels are computed using Gauss-Seidel iteration. Note that the gradient field does not necessarily need to be defined over the whole image, but only in that region of the mask that indicates unknown pixels.

2. **(1 point)** Show an example of seamless cloning, i.e., gradient domain cut-and-paste. First you need to find suitable source and target images. Then you need to define a mask that indicates the region of the target that should be replaced by the corresponding region of the source. Use Photoshop or some other tool to paint the mask. Finally, you need to compute the gradient field using the source image.

3. **(1 point)** Demonstrate an example of gradient mixing. Select two images that are suitable as foreground and background. Compute the desired gradient field by combining the gradient fields of the two images by selecting the gradient with the larger magnitude at each pixel. Use the background image as the target, and a mask that fixes only the boundary pixels of the target.

4. **(1 point)** Show an example of highlight removal. Use an image with suitable highlights and compute its gradient field. Compute the desired gradients by scaling the input

gradients using alpha compression. Use the original image as the target, and a mask that fixes only the boundary pixels.

Turn-in your matlab function written in assignement 1 (.m-file!). Create a matlab file which runs your experiments described in assignements 2 to 4. Organize this file in cells and dont't forget to include all input images needed.

Image Segmentation using Graph Cut Optimization

In this part of the project you will implement an interactive image segmentation tool using graph cut opimization.

1. **(1 point)** Implement an interface that allows a user to draw rough foreground and background regions onto an image. Use `imfreehand` to let the user draw the regions. Use `input` to obtain interactive keyboard input to let the user indicate foreground or background regions.

2. **(1 point)** Construct color models for the foreground and background regions. Use the following Matlab code to extract pixel colors from the masked regions. The code assumes the input image is stored in `img`, and the binary masks for foreground and background are stored in `fmask` and `bmask`.

```
% get all pixel colors in a 3 x (#pixels) array
colors = reshape(shiftdim(img,2),3,size(img,1)*size(img,2));
% get all pixel coordinates in a 2 x (#pixels) array
x = zeros(size(img,1),size(img,2),2);
[x(:,:,1) x(:,:,2)] = meshgrid(1:size(img,2),1:size(img,1));
x = reshape(shiftdim(x,2),2,size(img,1)*size(img,2));
% get the foreground and background masks in a 1 x (#pixels) array
fmask = reshape(fmask,1,size(img,1)*size(img,2));
bmask = reshape(bmask,1,size(img,1)*size(img,2));
% get the indices of the foreground and background pixels
[tt ttt findices] = intersect(x', (x.*[fmask; fmask])', 'rows');
[tt ttt bindices] = intersect(x', (x.*[bmask; bmask])', 'rows');
% extract only labeled foreground and background pixels
fcolors = colors(:,findices);
bcolors = colors(:,bindices);
```

   You can now use the data in `fcolors` and `bcolors` to build Gaussian mixture models (GMMs) for the foreground and background colors. This is achieved with the `gmdistribution` function. Two or three Gaussian components should be sufficient. The estimated probability density function (pdf) can be accessed using the `pdf` command. Type `help gmdistribution/pdf` to get more information.

   Visualize the mean colors estimated by the GMMs using the `mu` attribute of the GMM returned by `gmdistribution`. Also show the pdf's for background and foreground over the input image, i.e., show two grayscale images, where in each image the value at each pixel is either the probability for the pixel to belong to the foreground, or to the background.

3. (**2 points**) Set up and solve a graph cut optimization to obtain an image segmentation. First, get a graph cut implementation for Matlab from `http://vision.ucla.edu/~brian/gcmex.html`. The main function is `GCMex`. Study the example in `GCMex_test.m` to see how it is used. Type `help gcmex` to get a description of the parameters for this function. Here is an excerpt from the help text:

```
GCMEX(CLASS, UNARY, PAIRWISE, LABELCOST,EXPANSION)

Runs a minimization starting with the labels for each node defined
by CLASS, with unary potentials UNARY and the structure of the
graph and pairwise potentials defined by PAIRWISE. LABELCOST
determines data costs in terms of the labels of adjacent nodes.

Parameters:
 CLASS:: A 1xN vector which specifies the initial labels of each
   of the N nodes in the graph
 UNARY:: A CxN matrix specifying the potentials (data term) for
   each of the C possible classes at each of the N nodes.
 PAIRWISE:: An NxN sparse matrix specifying the graph structure and
   cost for each link between nodes in the graph.
 LABELCOST:: A CxC matrix specifying the fixed label cost for the
   labels of each adjacent node in the graph.
 EXPANSION:: A 0-1 flag which determines if the swap or expansion
   method is used to solve the minimization. 0 == swap,
   1 == expansion. If ommitted, defaults to swap.
```

You should consider the following notes on these parameters:

- `CLASS`: Use a vector consisting of zeros. Note that the length `N` is the number of pixels in the image.
- `UNARY`: The data term contains the negative logarithm of the pdf's for foreground and background at each pixel. For binary segmentation, there are two classes (foreground and background), therefore this is a $2 \times N$ matrix. Note that the pdf for some pixels may be zero, such that the logarithm is not defined. Therefore, add a small constant $\epsilon \approx 0.00001$ to the pdf.

  The data term should also include the information about the user specified masks. For pixels belonging to the foreground mask, set the data term for the foreground label to zero, and for the background label to $-\log(\epsilon)$. This means that the cost for assigning a background label to a pixel that was masked as foreground by the user will be $-\log(\epsilon)$, which is a large number. The data term for pixels belonging to the background mask is analogous.
- `PAIRWISE`: This parameter stores the smoothness term in a sparse $N \times N$ matrix. For each pair of interacting pixels, this matrix contains the penalty if the labels

of the pixels do not agree. Note that most of the matrix elements are zero, since only pixels in an 8-neighborhood interact. Therefore, it would be wasteful to store the full matrix.

Instead, you should first compute a list of interacting pixels, and store them in arrays. You will need three arrays, two for the indices of each pair of interacting pixels p and q, and one for the penalty. The penalty is

$$\gamma e^{-\beta \|c_p - c_q\|^2},$$

where $\|c_p - c_q\|$ is the Euclidean distance between the colors of pixels p and q. The purpose of the factor γ is to balance the contribution of the data and smoothness terms. You need to adjust it manually. Usually values between 5 and 30 should work well. The parameter β should be set to one-half of the color variance over the whole image. You can compute it using

```
avg = sum(colors,2)/size(colors,2);
beta = sum(sum( (colors-repmat(avg,1,size(colors,2))).* ...
(colors-repmat(avg,1,size(colors,2))) )) / size(colors,2);
beta = 1/(2*beta);
```

Finally, you construct a sparse matrix from these arrays using `sparse`, and you pass the sparse matrix as the `PAIRWISE` parameter to `GCMex`. Note that writing directly into the sparse matrix as in the `GCMex_test` example is extremely slow and should be avoided!

For debugging purposes, it is also useful to visualize the smoothness term. For this purpose, compute the sum of smoothness penalties at each pixel by summing over the columns of the `PAIRWISE` matrix. You can do this with the `sum` command. You get a vector of N elements (i.e., the number of pixels), which you can show as an image.

- `LABELCOST`: For each pair of labels, this matrix stores a cost factor. Given a pair of interacting pixels with associated labels, the smoothness cost is given by the corresponding entry in `LABELCOST` multiplied by the value in `PAIRWISE`. The smoothness cost should be zero if the interacting pixels have the same label. Therefore, the diagonal of the `LABELCOST` matrix is zero. For all other combinations of labels, the value stored in `PAIRWISE` should be applied. Therefore, all other elements in `LABELCOST` are one. Hence, use `labelcost = [0 1; 1 0];`.

4. **(1 point)** Demonstrate your segmentation tool using at least two examples. For each visualize the user provided masks, the means of the GMM color models, the foreground and background pdf's, the sum of smoothness costs at each pixel, and the final segmentation.

Turn-in the code written in the assignements 1 to 3 (including the graph cut implementation) and a PDF with the results of your experiments in assignement 4. Give also a short discription, how to start and run your image segmentation tool.

Bonus: Digital Photomontage

(**1 point**)The goal of this bonus assignment is to implement a simplified version of the interactive digital photomontage application as discussed in the lecture. The simplification is to combine just two images, as opposed to an arbitrary number of images. The restriction to two images is necessary because, unfortunately, the Matlab graph cut solver does not support the more sophisticated cost functions to handle an arbitrary number of images.

Your application should take two images of the same size as input. It should allow the user to mark regions in both images that he wants to composite in the output image. Your application will then compute a seamless collage combining the two input images.

Your method should use gradient domain compositing and graph cut segmentation to achieve this task. It will first find the best cut to combine the images, then copy the gradient field of one image into the other, and finally reconstruct a seamless collage.

Note that, in this application, the graph cut labels have a different meaning than before, where they denoted background and foreground. Instead they tells us which pixels are selected from which image. Therefore, for a pixel the user has selected in one image, the data term should be zero if this image is selected at the pixel, and it should be large if the other image is selected. For pixels that were not marked by the user, the data term is always zero. For the smoothness term between pixel p and q we recommend the following penalty,

$$||I_1(p) - I_2(p)|| + ||I_1(q) - I_2(q)||,$$

where $||I_1(p) - I_2(p)||$ denotes the Euclidean distance between the color of pixel p in image 1 and image 2. This smoothness term penalizes color differences between the two images and tries to avoid visible seams.

For more detailed explanations and different smoothness and data terms we refer to the original research paper, which you can find at `http://grail.cs.washington.edu/projects/photomontage/`. It is worth having a look at the webpage for inspiration! They provide some interesting examples, the source code of their implementation, and also an executable for Microsoft Windows.

Turn-in the matlab code, a short description how to use your application, and your example results (including the input images) that you created with your application.