Problem Set 4

CMSC 733
Assigned Tuesday November 12, 2013
Due Tuesday, November 26, 2013

Stereo Correspondence. For this problem set you will solve the stereo correspondence problem using graph cuts, as described in class, and in the paper Fast approximate energy minimization via graph cuts, by Boykov, Veksler, and Zabih. In the main part of the problem set, you will implement the alpha-beta swap algorithm. For extra credit, you can also do alpha-expansion. You will implement everything just as they describe, except for one small simplification that I'll mention below. You do not need to implement code to compute the graph cuts; links to that are given below. Everything else should be implemented from scratch. (Note, it is not allowed to consult or make use of code that you might find on-line, beyond the code linked to below).

When turning in your problem set, please include a printout of your code (in addition to mailing a zip file), along with pictures showing the results of your algorithms.

To get started, download the maxflow code of Boykov and Kolmogorov from: http://vision.csd.uwo.ca/code/ The zip file you want is: maxflow-v3.01.zip. Note that this is C++ code.

Next, download Miki Rubinstein's Matlab interface to the maxflow code from: http://www.mathworks.com/matlabcentral/fileexchange/21310-maxflow. Note that per instructions in the readme file, you should place the C++ maxflow code in a subdirectory <lib_home>/maxflow-v3.0>. Follow the instructions in Rubenstein's README file to make and test the maxflow code.

Hint: The maxflow code is version 3.0.1. The Matlab interface expects it to be in a directory named maxflow-v3.0. Don't let this confuse you. As long as you name the directory maxflow-v3.0, they should work together fine.

For this assignment you will need to use sparse matrices. If you haven't used sparse matrices before, look at the documentation for them: `help sparse`. In particular, you will need to create a sparse matrix that would be really huge if it wasn't sparse. You won't be able to do this by creating the full matrix and then making it sparse. The Matlab code you've downloaded contains examples that should guide you in this.

1. **20 points** Write a function to compute the unary matching costs. This should have the form:

```
function dist = unary_cost(L,R,MAXD)
```

L is the left image, R is the right image. We assume that the images are rectified, so that pixels on L(j,:) match pixels in R(j,:). MAXD is the maximum possible disparity (you can use a value of 14 or less for this problem set). dist will contain the unary costs, aka the data term. Boykov et al. describe a method of computing this using interpolation, but we will use something much simpler. Just create dist so that:

dist(i,j,d) = (L(i,j) – R(i,j-d))^2.

Test this with a small pair of images and show that you are computing the correct values. You may assume that L and R are the same size.

2. **20 points** Create a function that will produce the pairwise cost for neighboring pixels. This can have the form:

```
function P = potts_cost(L,K)
```

Here L is the left image, and K is a constant (see Equation 20 in the paper). Note that the pairwise cost doesn't depend on the right image. P will be a 3D array. For example, P(:,:,1) could contain the pairwise cost for each pixel and the neighbor that is above it. With the Potts model, you only need to compute the cost for the case in which the pixels have different labels. If they have the same label, the cost is 0.

Test this function on a small image, and show that it produces correct results.

3. **10 points** Create a function that initializes the disparities with a reasonable starting point. A good idea is to start with the correspondences that optimize the unary costs, ignoring the pairwise costs. This could have the form:

```
function D = initialize(dist)
```

D will be the size of the images used to create dist. D(i,j) will contain the disparity at pixel (i,j).

4. **50 points** Now implement a function of the form:

```
function D = stereo(L,R)
```

that uses alpha-beta swaps to compute the disparity between two images. D should be the disparity map that you get, indicating correspondences. We provide two stereo pairs you can use to test your algorithm. A good way to visualize your results is to display them as images, by doing:

```
figure;
imagesc(D);
colormap(gray);
```

I1L and I1R are the left and right images of a random dot stereogram. We're not providing a solution, but you will know when you have the right answer. Tleft and Tright contain the two famous Tsukaba images that you can use. The web site also contains the results produced by my implementation. Note that my results are a little worse than those in the paper; this may be due to the lack of interpolation, as noted above, or maybe there's a small bug in my code. Let me know if you get results that look better than mine.:)

Also note that you will not be able to easily run your code on pictures you take yourself, since they would need to be rectified.

5. **Extra Credit (up to 10 points):** Implement the alpha-expansion algorithm. Compare this to alpha-beta swap. Can you explain the differences? Can you come up with a small, simple example so that one algorithm produces a result that is clearly better than the other (you could answer this last question even if you don't implement alpha-expansion)?

6. **Extra Credit (up to 10 points):** The version we have implemented does not allow for occlusion. Can you come up with a variation on graph cuts that does? Note that Boykov et al. refer to a paper by Kolmogorov and Zabih that handles occlusion. Implement a version with occlusion, and explain what you did. Analyze the extent to which this helps (it might not help much in the images that we give you. You might look at the Middlebury Stereo page for more images and ideas).