# CS 766: Computer Vision
# Homework #2
# Image Segmentation using Mean Shift

Due: Tuesday, October 17, 2006

## General Information

- This homework is based on the mean shift segmentation algorithm; papers describing this method are in

  `~cs766-1/public/html/fall06/hw/hw2/papers/`

- Provided MATLAB functions are in

  `~cs766-1/public/html/fall06/hw/hw2/code/`

- Provided test data (images and `pts.mat`) are in

  `~cs766-1/public/html/fall06/hw/hw2/data/`

- Do not print out your MATLAB code listing. Please submit your code to the `hw2` handin directory.

## Mean Shift Clustering

The Mean Shift algorithm clusters an $n$-dimensional data set (i.e., each data point is described by a feature vector of $n$ values) by associating each point with a peak of the data set's probability density. For each point, Mean Shift computes its associated peak by first defining a spherical window at the data point of radius $r$ and computing the mean of the points that lie within the window. The algorithm then shifts the window to the mean and repeats until convergence, i.e., until the shift is less than a threshold (e.g., 0.01). At each iteration the window will shift to a more densely populated portion of the data set until a peak is reached, where the data is equally distributed in the window. Implement the peak searching processes as the function

```
function peak = findpeak(data,idx,r)
```

where `data` is an $n \times p$ matrix containing $p$ data points, each point is defined by an $n$-dimensional column vector of feature values; `idx` is the column index of the data point for which we wish to compute its associated density peak; and `r` is the search window radius.

Implement the `meanshift` function, which calls `findpeak` for each point and then assigns a label to each point according to its peak. This function should have the syntax:

```
function [labels,peaks] = meanshift(data,r)
```

where `labels` is a $1 \times p$ vector that has an entry for each data point of `data` storing its associated cluster label, and `peaks` is an $n \times p$ matrix storing the density peaks found using `meanshift` for each data point. Peaks should be compared after each call to `findpeak` and similar peaks should be merged. For your implementation, consider two peaks to be the same if the distance between them is $\leq r/2$. Also, if the peak associated with a data point is found to already exist in `peaks`, then for simplicity its computed peak is discarded and it is given the label of the already existing peak in `peaks`.

## Speeding up the Algorithm

As described so far, the Mean Shift algorithm is too slow to be used for image segmentation where each pixel is a data point. Therefore, you should incorporate the following two speedups into your implementation. Upon finding a peak, the first speedup will be to associate each data point that is at a distance $\leq r$ from the peak with the cluster defined by that peak. This speedup is known as the "basin of attraction" and is based on the intuition that points that are within one window size distance from the peak will, with high probability, converge to that peak.

The second speedup is based on a similar principle, where points that are within a distance of $r/c$ of the search path are associated with the converged peak, where $c$ is some constant value. Use $c = 4$ for this assignment.

Incorporate both of the above speedups into your implementation of `meanshift`. To realize the second speedup you will need to modify `findpeak` as follows:

```
function [peak,cpts] = findpeakopt(data,idx,r)
```

where `cpts` is a vector storing a 1 for each point that is within a distance $r/4$ from the path, and a 0 otherwise.

Since MATLAB is optimized for matrix operations, not loops, try to avoid using loops in your functions whenever possible. Instead, use matrix manipulation. For example, if you want to select all points that have been labeled "1," instead of writing a for loop you could write:

```
found = find(labels == 1);
currdata = data(:,found);
```

which uses the `find` command to return the indexes of the elements of `labels` that match the Boolean expression, which in this case is "== 1". Then `data(:,found)` selects those columns whose indexes are listed in the vector `found`.

Here's another MATLAB hint: say you are trying to find points in a matrix $\mathbf{A}$ that are closer than $r$ to the point of index $idx$ in $\mathbf{A}$. You can do it with a for loop this way:

```
for i=1:size(A,2)
       R(i)=norm(A(:,i)-A(:,idx));
   end
   find(R<r)
```

But this is way too slow in MATLAB. The following is much faster:

```
n = size(A,2)
   find(r>sqrt(sum((A - repmat(A(:,idx),1,n)).^2,1)))
```

Debug and test your algorithm, both with and without the two speedups, using the provided dataset `pts.mat` which contains a set of 3D points that belong to two 3D Gaussian distributions. Compute results using $r = 1$, 2, and 4 and plot each result separately using the provided function `plot3Dclusters`.

## Image Segmentation

Next, build upon your implementation so that it can be used to perform image segmentation. To do so, implement the function

```
function segIm = meanshiftSegment(im,r)
```

where `im` is an input image or, more generally, an image feature matrix, and `r` is the parameter associated with the Mean Shift algorithm. The output segmented image is then constructed using the cluster labels and peak values. That is, the output image is constructed by assigning a different color value (from the peak value) to each label and coloring pixels in the output image accordingly.

Note that Mean Shift clusters use the Euclidean distance metric. Unfortunately, Euclidean distance in RGB color space does not correlate well to perceived difference in color by people. For example, in the green portion of the spectrum large distances are perceived as the same color, whereas in the blue part of the spectrum a small distance may represent a large change in perceived color (see Figures 6.13 and 6.14 of Forsyth and Ponce). For this reason you should use the non-linear LUV color space. In this space Euclidean distance better models the perceived difference in color. In `meanshiftSegment` cluster the image data in LUV color space by first converting the RGB color vectors to LUV using the provided MATLAB function `rgb2luv`. Then convert the resulting cluster centers back to RGB using the provided MATLAB function `luv2rgb`.

## Experiments

Segment the images `sunset.bmp` and `terrain.bmp` using $r = 5$ and 10. Given an input image file, you'll need to read the image into MATLAB and then convert it into the matrix form required by `data`. If the feature vector is simply the 3 LUV color values at a pixel, then you'll have to create a $3 \times p$ matrix where $p$ is the number of pixels in the input image. If we want to include spatial position information as well, we can define the feature vector as a 5D vector specifying the LUV and x,y coordinates of each pixel. For each value of $r$, run your algorithm using (1) just the LUV color values (i.e., a 3D feature vector), and (2) LUV+position values (i.e., a 5D feature vector). Create a color output image for each combination of $r$ and feature vector definition, resulting in 4 output segmentation images. Also run your algorithm on at least one other test image using your own choice of $r$ and feature vector definition. One source for possible test images is the dataset of images available at

`http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/`

## Questions to Answer

Based on your experiments and the resulting segmentations, answer the following questions. What effect does varying $r$ and the feature vector seem to have on the resulting segmentations? What effect does adding position information have on the resulting segmentations? What are the advantages and disadvantages of using each type of feature vector? Can you suggest any extensions that might avoid many of the situations where single regions are over-segmented into multiple regions? Hand in hardcopy of the output images and the answers to these questions.

A final note: Although more efficient implementations exist, the above simplified implementation of Mean Shift may take several minutes to run. Debug using small images.

## Comparison with the Normalized Cut Algorithm

Use the MATLAB implementation of the Normalized Cut method for image segmentation given at

`http://www.seas.upenn.edu/~timothee/software_ncut/software.html`

and run it using the same test images you used earlier. Comment on the differences between the two methods and for what types of images you might prefer to use one or the other.

## Extra Credit Problem: 2D Skeletonization using Shock Graphs

One useful way of modeling 2D shapes is by a skeleton description, initially studied by H. Blum with the medial axis transform (MAT). For extra credit, implement a method for computing a generalization of the MAT called a **shock graph**. The algorithm you are to implement is given in Section 3.3 of the paper "Robust and Efficient Skeletal Graphs" by P. Dimitrov, C. Phillips and K. Siddiqi, *Proc. Computer Vision and Pattern Recognition Conf., Vol. 1*, 2000, 417-423. This paper is called `siddiqi-cvpr00.pdf` in the directory

`http://www.cs.wisc.edu/~dyer/cs766/readings/`

The steps of the algorithm that you are to implement are now summarized.

## Step 1: Compute the Euclidean Distance Transform

The first step of the shock graph construction algorithm is to compute the Euclidean Distance Transform (EDT). Do this in Matlab by first reading in your image file using the function `imread`. Next, compute the EDT using the Matlab function `bwdist` to produce a real-valued Euclidean distance transform array `D`. Note: `bwdist` assumes that object pixels have value 0 and background pixels have any non-zero value.

In order to reduce the effects of boundary aliasing due to the digital boundary of the object, smooth `D` slightly by blurring it using a Gaussian kernel. Create this kernel using the Matlab function `fspecial`. For example, to create a 3 x 3 kernel with sigma = 0.5, do `G = fspecial('gaussian',[3 3],0.5)`. Experiment with at least two different kernel sizes and values of the parameter sigma to obtain the best final results you can. (For example, use a 3 x 3 kernel with sigma=0.5 and use a 5 x 5 kernel with sigma=1.0.) Specify the final values you used in your documentation. Convolve your computed kernel with `D` using the Matlab function `imfilter`. The result should be a real-valued 2D array `SD` containing smoothed values of `D` for every pixel. For more information on creating and applying a Gaussian filter, see

```
http://www.cs.wisc.edu/~dyer/cs766/hw/hw2/HowToCreateFilter.pdf
```

Create a separate, intermediate-result image that contains 255 (white) at each background pixel and the `SD` value, rounded to the nearest integer, at each object pixel. Linearly scale the SD values so that the largest value becomes 0 (black) and an SD value of 0 becomes 255 (white). You can do this in Matlab using the functions `stretchlim` and `imadjust`. (The reason for inverting the background gray level is so that the printed images look nicer.) Save this intermediate-result image in an image file using `imwrite`. Print out and turn in this image. (If you prefer, you can create, display and print this image within Matlab and never create an intermediate image file.)

## Step 2: Compute the Gradient Vector Field

Using the real-valued array `SD`, the second step of the Dimitrov et al. shock graph algorithm is to compute the gradient vector field, `delD` from `SD`. One way to compute `delD` would be to use a simple finite difference approximation, but this blurs across singularities, which is precisely where the skeletal points lie. (If you want to try out this method for curiosity and comparison purposes, you can use the Matlab function `gradient`.

For this assignment you are to compute `delD` as the unit vector `-PQ/`$\|$`PQ`$\|$ where `P` is a given object pixel and `Q` is (one of) its nearest background point(s). `Q` can be computed at the same time as the EDT using `bwdist` as follows: `[D,L] = bwdist(IMAGE)` Used in this way, `L` is an array of type `double` of the same size as `IMAGE` specifying for each pixel (one of) its closest background pixel(s). Use the value in `L` (after converting it from its raster-scan index value to image coordinates) to define the point `Q` needed to compute `delD`. Create an image of this vector field using Matlab's `quiver` function and print a copy of this image.

## Step 3: Compute the Flux

The goal of this step is to compute the flux (also known as the divergence) defined by Eq. (3) in the paper by Dimitrov et al. This is done as described in the algorithm in the paper by summing over all eight neighbors of each point the values computed by the inner product of `delD` and a unit outward normal from the center point to the current 8-neighbor. Since in our case `delD` is undefined for background pixels, modify this step to compute instead the **average flux** over those 8-neighbors that are in the object. Note: You may want to use the Matlab function `divergence` and compare your result with theirs; your code should use your own implementation ofthis step, however.

Hint: For those of you who are a little rusty on your linear algebra, given two vectors in $\mathcal{R}^2$, $\mathbf{u} = [u_1, u_2]^T$ and $\mathbf{v} = [v_1, v_2]^T$, the cosine of the angle between $\mathbf{u}$ and $\mathbf{v}$ is equal to the inner product (aka dot product) of the unit vectors $\mathbf{u}/\|\mathbf{u}\|$ and $\mathbf{v}/\|\mathbf{v}\|$. The length (or norm) of a vector $\mathbf{u} = [u_1, u_2]^T$ is defined as $\|\mathbf{u}\| = \sqrt{u_1^2 + u_2^2}$. Given two vectors, their inner product is defined as $\mathbf{u} \cdot \mathbf{v} = <\mathbf{u}, \mathbf{v}> = \mathbf{u}^T\mathbf{v} = u_1v_1 + u_2v_2$, which is a scalar value. In this assignment $\mathbf{u}$ and $\mathbf{v}$ correspond to the vectors $\mathbf{N_i}$ and `delD`($\mathbf{P_i}$). `Flux(P)` is the average of these (up to eight) scalar values.

## Step 4: Thin

The final step is to compute the skeletal points by thinning, i.e., removing object points in order by their flux values. Strong negative flux corresponds to skeletal points. Thinning must not remove endpoints or change the topology at any point during this process. The method for doing this is given in the algorithm in the paper. You can create a heap class in Matlab for this part.

Create an output image where background pixels are white (255), object pixels are middle-gray (128), and the skeleton pixels are black (0). Print and hand in this image.

## What to Hand In

Copy your source code into your handin directory `hw2`. Hand in in hardcopy form (1) a description of any parameters you used and their values, (2) hardcopy images of the blurred EDT, the vector field, and the final skeleton results for at least the following two test images:

```
~cs766-1/public/images/hw2/europeanhare.gif
~cs766-1/public/images/hw2/girl.gif
```

## For More Information

The following papers may also be consulted for other details related to this algorithm. They are all located in the directory `http://www.cs.wisc.edu/~dyer/cs766/readings/`

1. F. Leymarie and M. Levine, Fast raster scan distance propagation on the discrete rectangular lattice, *Proc. Computer Vision, Graphics and Image Processing: Image Understanding* 55, 1992, 84-94. Pdf: `leymarie-cvgip92.pdf`

2. K. Siddiqi et al., The Hamilton-Jacobi skeleton, *Proc. Int. Conf. Computer Vision*, 1999. Pdf: `siddiqi-iccv99.pdf`

3. K. Siddiqi et al., The Hamilton-Jacobi skeleton, *Int. J. Computer Vision* 48(3), 2002, 215-231. Pdf: `siddiqi-ijcv02.pdf`

4. S. Bouix and K. Siddiqi, Divergence-based medial surfaces, *Proc. European Conf. Computer Vision*, 2000. Pdf: `siddiqi-eccv00.pdf`

5. K. Siddiqi et al., Geometric shock-capturing ENO schemes for subpixel interpolation, computation, and curve evolution, *Graphical Models and Image Processing* 59(5), 1997, 278-301. Pdf: `siddiqi-eno.pdf`