Centre for Mathematical Sciences, September 2010

Image Analysis, Laboratory session 3

Preparations

Read the material about graph cuts (lecture notes and Szeliski 5.5).

Log in

Log in as usual. Don't forget to start Matlab in the library where the files for this session are located (maxflow directory) or, alternatively, add a path to the directory using `addpath`. If you have forgotten where the directory is located you can download the files from the course home page again.

Image Segmentation

A very simple yet useful model for image segmentation is the following: the foreground is modelled as having a constant grey level μ_1 and the background as being constant equal to μ_0. The segmentation task is to find a curve γ such that the region Γ inside the curve is foreground and the exterior is background. See Figure 1. Finding the best possible curve results in an optimization problem and is the topic of this computer session.

A simple version of the Mumford-Shah functional takes the length of γ into account:

$$E(\gamma) = \lambda \,\mathrm{length}(\gamma) + \iint_{\Gamma} (I(\boldsymbol{x}) - \mu_1)^2 d\boldsymbol{x} + \iint_{\Omega\setminus\Gamma} (I(\boldsymbol{x}) - \mu_0)^2 d\boldsymbol{x} \tag{1}$$

We would like to find the curve γ which minimizes $E(\gamma)$. We will do this with a discrete approximation of E. From now on, we view I as a discrete image with n pixels. Let θ be an indicator variable for the foreground, i.e.:

$$\theta_i = \begin{cases} 1, & \text{if } i \text{ is foreground} \\ 0, & \text{if } i \text{ is background .} \end{cases} \tag{2}$$

Let i and j be two image pixel locations. We say that i **is a neighbor of** j if they are adjacent, either horizontally or vertically. We write this as $j \in \mathcal{N}_i$.

```
M=5; %Width of image
N=6; %Height of image
n = M*N; %Number of image pixels
```

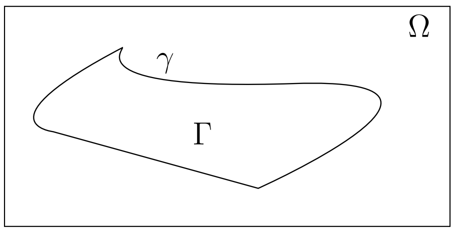

Figure 1: Image segmentation

```
Neighbors = edges4connected(M,N);
i=Neighbors(:,1);
j=Neighbors(:,2);
A = sparse(i,j,1,n,n);
```

This creates a matrix A such that $\mathsf{A}_{ij} \neq 0 \iff j \in \mathcal{N}_i$. We can plot this 5×6 image along with the neighborhood structure with

```
[X Y] = ndgrid(1:M,1:N);
draw_graph(X(:),Y(:),A)
```

Question 1. *Does* $12 \in \mathcal{N}_{17}$*?*

We can now approximate (3) by

$$E(\theta) = \frac{\lambda}{2}\sum_{i=1}^{n}\sum_{j\in\mathcal{N}_i}\mathbb{I}(\theta_i \neq \theta_j) + \sum_{i=1}^{n}\theta_i(I(i)-\mu_1)^2 + \sum_{i=1}^{n}(1-\theta_i)(I(i)-\mu_0)^2. \qquad (3)$$

$\mathbb{I}$ is the indicator function, i.e. $\mathbb{I}(\theta_i \neq \theta_j)$ is equal to 1 if θ_i and θ_j are different. Thus the first double summation counts the total number of separated neighbors.

Question 2. *How does* (3) *approximate* (1)*?*

Minimizing (3) is an example of a **minimum cut** problem (see lecture notes). There exists very efficient methods to solve it exactly. We will use the software written by Boykov and Kolmogorov, which is ubiquitous in the image analysis community. The first task is to load an image (feel free to use your own favourite one).

```
I = imread('cameraman.tif');
I = double(I)/255;
[M N] =size(I);
n = M*N; %Number of image pixels
```

Next, choose some values for μ_1, μ_0 and λ to specify what foreground and background looks like and to decide how important a short curve length is.

```
mu0 = ???
mu1 = ???
lambda = ???
```

Create the neighbors and the sparse matrix in the same manner as before. Set $\mathsf{A}_{ij} = \lambda \iff j \in \mathcal{N}_i$. (Don't try to draw this graph with `draw_graph` – it's too big)

```
A = ???
```

Now we handle the other two sums in (3). They will be represented with s and t connections in the graph (see the lecture notes to refresh your memory). In the software package we are going to use, these are represented as a separate $n \times 2$ matrix:

```
T = [ (I(:)-mu1).^2 (I(:)-mu0).^2];
T = sparse(T);
```

Finally, we solve the minimum cut problem:

```
tic
[E Theta] = maxflow(A,T);
Theta = reshape(Theta,M,N);
Theta = double(Theta);
toc
```

And we can now view the output:

```
imshow(Theta);
```

Write a function so that you may easily try the code with different values of the parameters:

```
function Lab = segment_image(I,mu0,mu1,lambda)
        ...
end
```

Question 3. *What are some good values of μ_0 and μ_1?*

Question 4. *What are some good values of λ? Above which value does the solution no longer consist of two different regions?*

Sub-pixel edge detection

When a high accuracy is needed in the calculations, it is not always sufficient to use only integer pixel locations. In order to obtain sub-pixel precision we have to be able to interpolate image intensities in decimal coordinates, i.e. to assign a gray-level intensity to the position in the image matrix with coordinates e.g. $(10.8, 23.2)$. There are many interpolation methods, e.g. pixel replication, linear-, cubic-, spline-, Gaussian- and sinc-interpolation. In this computer laboratory exercise, we will use Gaussian interpolation. This method gives also a smoothing of the image, which usually is desirable in many applications. The intensity at a point is calculated as the sum of all intensities in the image, weighted using a Gaussian function with centre at the point where we would like to calculate the intensity:

$$I(x) = \sum_k i(k) g_b(x-k) \ ,$$

where g_b denotes a Gaussian function with width b. The width determines the level of smoothing. The derivative of an image at a point given by 'decimal coordinates' can be calculated from

$$I'(x) = \sum_k i(k) g_b'(x-k) \ .$$

For one-dimensional signals, edges can be defined as local maxima in the derivative of the intensities. These can be determined e.g. using Newtons method. A routine for edge detection for a row at sub-pixel precision is implemented in `rowedges.m`. Try this routine at several rows in an image, e.g. using the script `edgepos.m`

```
bild = double(imread('cameraman.tif')); %For example!

b=1.2
threshold=10;
edgedata=[];
for rownr=180:260;
  row=bild(rownr,:);
  edgeposfine=rowedges(row,b,threshold);
  edgedata=[edgedata [edgeposfine;rownr*ones(size(edgeposfine))]];
end
figure
colormap(gray(256));
hold on;
imagesc(bild);
rita(edgedata,'r+');
zoom on
```

Zoom in at different edges and study the quality of the edge detection.

If time permits

Construct a scheme to automatically find good values of μ_0 and μ_1 by iterating the following scheme:

1. Segment the image using μ_0 and μ_1
2. Set μ_0 and μ_1 to the mean of their respective region

Question 5. *Does this iterative scheme always converge to the same values of μ_0 and μ_1?*