Edge Detection

Radim Šára

Center for Machine Perception
Department of Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague

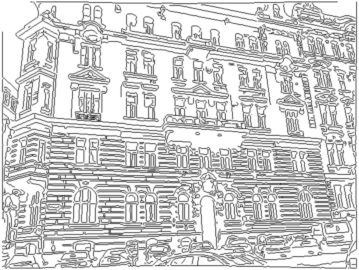
http://cmp.felk.cvut.cz
mailto:sara@cmp.felk.cvut.cz

rev. October 29, 2014

Digital Image Processing Course

What are Image Edges?

source image



Canny edges

What are Image Edges?

In short: Edges are image loci of strong intensity changes.

- **historical view:** edges constitute an image sketch
  - (some) edges do capture shape

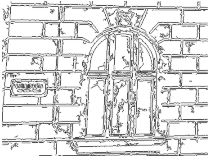
    they coincide with occlusion boundaries, shadow boundaries, material changes
  - neurophysiological and psychophysical studies consider edges important for perception
  - this school of thought peaked with part-based 3D representations (Geons)

Pablo Picasso, La Sieste 1919

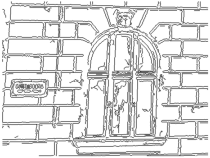
- **modern view:** edges are useful image structures, along with, ridges, corners, junctions, regional segments, etc.
  - edges are well localized in image
  - edges are stable features accross views

Want Edges? Easy. But Wait . . . What Edges?

```
sigma = 0.8;
im = rgb2gray(imread('facade.jpg'));
edg = edge(im, 'canny', [0.02, 0.12], sigma);
imagesc(-edg)
axis image
```

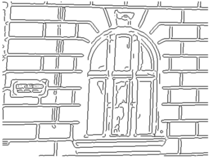


$\sigma = 0.1$, accurate but cluttered

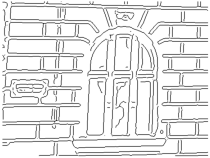


$\sigma = 0.5$

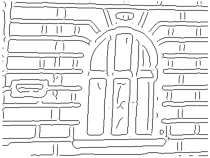
$\sigma = 1.0$



$\sigma = 1.5$



$\sigma = 2.0$

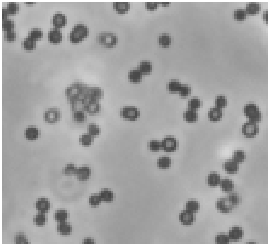


$\sigma = 3.0$, clean but inaccurate

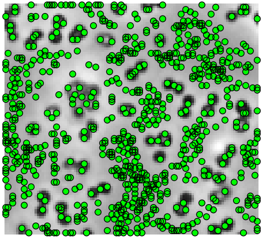
- as σ increases, false positives decrease but accuracy decreases too
- what is this σ?

The σ is the **Scale**: A Universal 'Problem' in Vision.

- let us study an easier problem than edge detection: circular blob detection
- a detector can be based on detecting local image minima



input image



local minima in 3×3 neighborhood

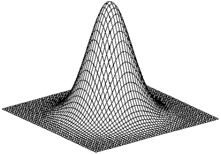
```
d = nonmaxsup(-im, [3 3]); % detect minima in 3x3 ngh
[i,j] = find(d);           % locate the yes responses
imagesc(im)                % show image
point([j,i]')              % overlay detections over the image
```

- . . . oops
- Is this idea bad? No.
- the problem is that we have to filter-out noise

A Better Cell Detector

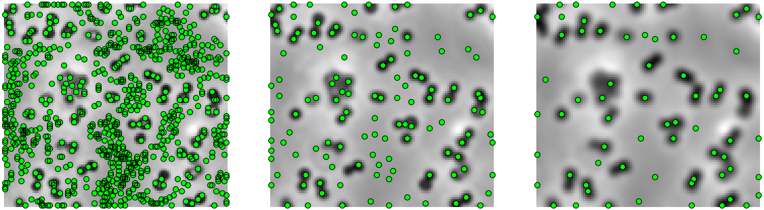
- we define a circular template of radius σ: 2D Gaussian of variance σ^2

$$f(x,y) = \exp\left[-\frac{x^2+y^2}{2\sigma^2}\right]$$



- similarity with a template is done by image convolution

```
msksize = 1+2*round(3*sigma/2);
h = fspecial('gaussian', msksize, sigma); % Gaussian kernel
img = imfilter(im,h,'replicate');         % convolution
d = nonmaxsup(-img,[3 3]);
```



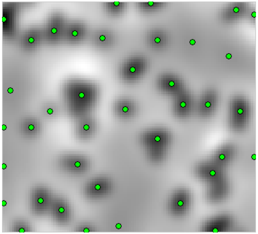
no filtering $\sigma = 0$ | $\sigma = 1.0$ | $\sigma = 2.0$

Scale Space

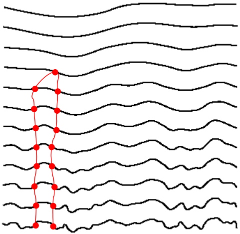
Q: So, what σ do we choose?
A: All of them.

- the σ is called scale and it defines the level of detail we are interested in
- if we do not know what level of detail is useful for a given task we work with a range of scales simultaneously
- minima at a coarser scale tend to be close to the minima at finer scale see video
- in 1D there is actually a nice inheritance structure

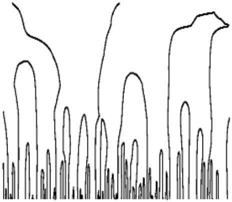
no such thing is guaranteed in 2D images but most of the time it is



video: detection in scale space



scale space for a 1D signal



scale-space events (edges)

Edge Detection: The General Scheme

1. choose scale σ and filter the image
2. compute a local image edginess feature

   toolbox:

   - convolutional edge templates
   - differential invariants

3. detect edges

   toolbox:

   - thresholding
   - non-maxima suppression
   - linking to chains

Vertical Edge Detection

- the simplest vertical edge template

$$\texttt{edge} = \begin{array}{|c|c|}\hline -1 & 1 \\ \hline \end{array}$$

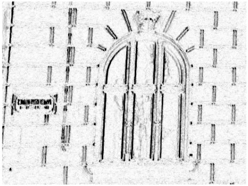
- it lacks the central point at which we want to detect the edge, hence we use

$$\texttt{edge\_c} = \begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline \end{array}$$

result will be placed here

```
sigma = 1;
h = fspecial('gaussian', msksize, sigma);
img = imfilter(im, h, 'replicate');          % scale selection
edge_c = [-1, 0, 1];
dx = abs(imfilter(img, edge_c, 'replicate')); % local edginess
```

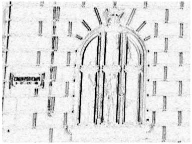
input image

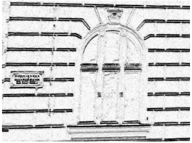


dx (larger is darker)

Directional Derivative


input image


dx


dy

- convolution with edge_c is equivalent to

$$I_x(x,y) = I(x+1,y) - I(x-1,y)$$

- this is the discretized derivative in the x-direction $\partial\hat{I}(x,y)/\partial x$

$\hat{I}(x,y)$ is the continuous image over continuous domain

- basic result on continuous scalar functions says that a directional derivative in arbitrary direction φ is

$$\mathrm{d}_\varphi\, I(x,y) = \frac{\partial I(x,y)}{\partial x}\cos\varphi + \frac{\partial I(x,y)}{\partial y}\sin\varphi = \nabla I(x,y)\cdot\mathbf{n} \qquad (1)$$

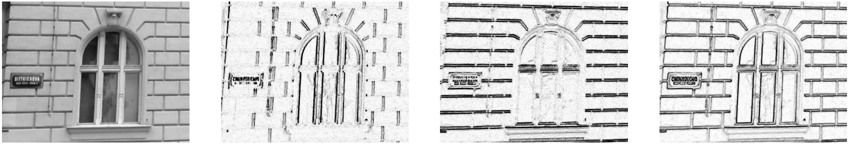
where $\mathbf{n} = (\cos\varphi,\, \sin\varphi)$ and '$\cdot$' is the dot product of two vectors

- we pretend this is possible in discrete images too and define image gradient as

$$\nabla I(x,y) = \big(I_x(x,y),\ I_y(x,y)\big)$$

Image Gradient: Direction and Magnitude

- directional derivatives are computed with the help of two auxiliary images using (1)



input image dx dy g

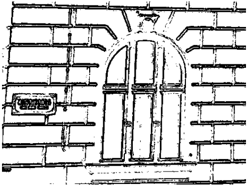
```
sigma = 1;
h = fspecial('gaussian', msksize, sigma);
img = imfilter(im, h, 'replicate');                % scale selection
edge_c = [-1, 0, 1];
dx = abs(imfilter(img, edge_c,  'replicate')); % horizontal difference
dy = abs(imfilter(img, edge_c', 'replicate')); % vertical difference

g = hypot(dx,dy);                                  % gradient magnitude
nx = dx./g;                                        % gradient direction
ny = dy./g;
```

- gradient magnitude is the desired local edginess, invariant to rotation
- gradient direction is the orientation of the edge (its normal vector)

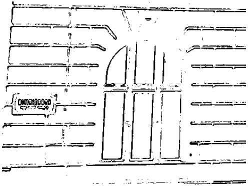
Detecting Edges

- so far, we only have the edginess image (image gradient magnitude)
- greater magnitude = more promising edge
- **detection = yes/no decision per pixel**
- what about classification by thresholding?

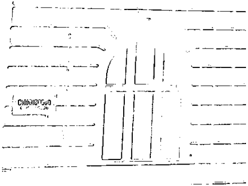
```
g = hypot(dx,dy);    % gradient magnitude
thr = 0.1;
edges = (g > thr);   % edge detection (or is it?)
```



thr = 0.1



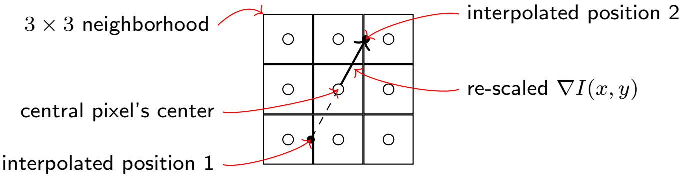
thr = 0.2

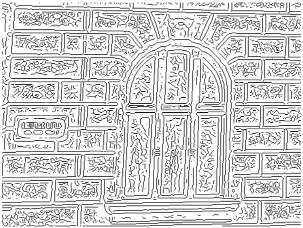


thr = 0.3

- we have got new problems:
  1. no single threshold seems to work
  2. edges are not thin (single-pixel wide)
- what about the idea of non-maximum/minimum suppression?

- **idea:** check if the gradient magnitude at (x, y) is locally maximal
  1. interpolate gradient magnitude between pixels at opposite directions (in black dots)
  2. compare their magnitudes with $g = \|\nabla I(x, y)\|$
  3. if g is greater, then the central pixel (x, y) is an edge pixel

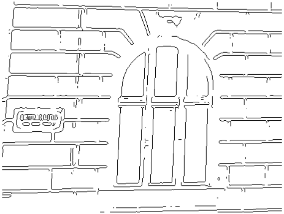




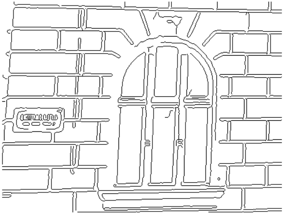
- looks fine but. . . very cluttered
- $\Rightarrow$ we still need some thresholding

Canny Edge Detector: Conditioned Thresholding

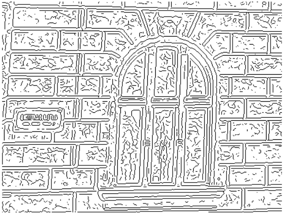
- Canny proposed two thresholds and detection conditioned on strong edges
  1. the upper threshold selects reliable edges
  2. then edges are grown from the reliable ones
  3. growth stops at the lower threshold



upper threshold applied



conditional growth



lower threshold applied

Canny detector criticism

- resulting edges do not possess any strong properties

Marr-Hildreth Edge Detector

- Canny detector looks at local gradient magnitude maxima: locations where gradient change vanishes
- there is a more principled view of this:
  Zero-crossings of the second-order derivative in the gradient direction
- these occur at approximately the locations of vanishing image Laplacian

$$\Delta I(x, y) = I_{xx}(x, y) + I_{yy}(x, y) = 0$$

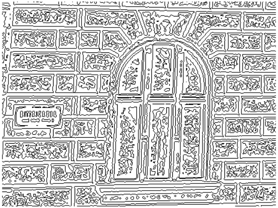
more precisely at $I_x^2\, I_{xx} + 2\, I_x\, I_y\, I_{xy} + I_y^2\, I_{yy} = 0$ [Lindeberg]

- this is another ‘feature’ that is computed with the help of a 2D convolution kernel

L =

1	4	1
4	-20	4
1	4	1

$\Delta I(x, y)$ at $\sigma = 1.5$



zero-crossings of $\Delta I(x, y)$

- edge pixels are zero crossings of the result

implement as sign changes in 3×3 neigborhood

- automatically single-pixel wide
- thresholding possible after zero-crossing edgels are traced-out to edges

Some Notes

- both Laplacian filter and Gaussian filter are linear operators
- they are associative (also commutative)

$$L * G * I = (L * G) * I = (G * L) * I$$

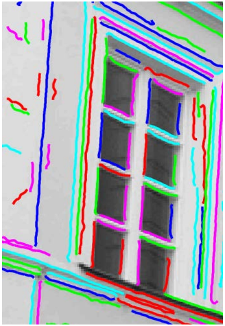
- the $L * G$ is the Laplacian of Gaussian (LoG)
- this is why the detector is often called the LoG edge detector
- LoG zero-crossings have strong topological properties: they are closed and nested curves that touch at saddle points
- LoG is approximately computed by subtracting two Gaussian filtered images, one with σ and the other with $\sigma + \delta\sigma$ — this is called DoG – Difference of Gaussians
- zero-crossings of the 2^{nd} fundamental form carry important information on image structure
- regions they encompass are of either elliptic or hyperbolic curvature, ie. not arbitrary

Criticisms and Discussion

- LoG edges are not accurate at sharp corners
- if this is a problem, use the 2^{nd} fundamental form
- but corners are not the goal of edge detection
- for that we have corner detectors — Harris, Foerstner, etc

Chaining Edgels

- what we got are free edgels – edge elements
- we need to chain them to get actual edges
- this example shows
  - Marr-Hildreth detector
  - zero-crossings chained to straight line segments
  - that minimize the total curvature of each segment



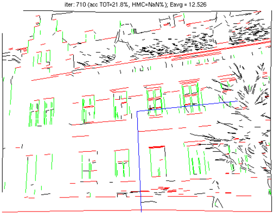
What Are Edges Good for?

- useful geometric entities, used for:
  - image correspondence (registration, stereo, tracking)
  - image interpretation (e.g. geons)

Example: Automatic image rectification

- perspective camera with known principal point, square pixel, unknown focal length
- perspective distortion of a plane is a special homography in this case 4-parameter mapping
- finite vanishing points corresponding to orthogonal directions determine the homography
- images can then be rectified

input image



detected edges
classified to horizontal and vertical

rectified image

Wrap-Up

We have visited some notions and principles

- edge as an image primitive
- convolution as a filter (Gaussian) and/or template (edge)
- scale
- non-maxima (non-minima) suppression

Thank You