Canny Edge Detection
Real-time Drawing Assistance through Crowdsourcing
SIGGRAPH 2013
A. Limpaecher, N. Feltman, A. Treuille, and M. Cohen
CMU and Microsoft
Edge Detection

Code

\texttt{Jb = rgb2gray(J);}

\texttt{imagesc(Jb);axis image; colormap(gray);} 

\texttt{bw = edge(Jb,'canny');}
Numerical Image Filtering

Looping through all pixels

\[
[nr, nc] = \text{size}(Jb);
J\_out = \text{zeros}(nr, nc);
\]

for \( i = 1 : nr, \)
  for \( j = 1 : nc; \)
    if (\( i < nr \) \&\& (\( i > 1 \))
      \[ J\_out(i, j) = 2 \times Jb(i, j) - 0.8 \times Jb(i+1, j) - 0.8 \times Jb(i-1, j); \]
    else
      \[ J\_out(i, j) = Jb(i, j); \]
    end
  end
end

figure; imagesc(J\_out); colormap(gray)

Computation time: 0.050154 sec
Numerical Image Filtering
Convolution without Looping using meshgrid

$$\text{>> } [x,y] = \text{meshgrid}(1:5,1:3)$$

\[
\begin{bmatrix}
1 & 2 & 3 & 4 & 5 \\
1 & 2 & 3 & 4 & 5 \\
1 & 2 & 3 & 4 & 5 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 1 & 1 & 1 & 1 \\
2 & 2 & 2 & 2 & 2 \\
3 & 3 & 3 & 3 & 3 \\
\end{bmatrix}
\]
[x, y] = meshgrid(1:nc, 1:nr);
figure(1); imagesc(x); axis image; colorbar; colormap(jet);
figure(2); imagesc(y); axis image; colorbar; colormap(jet);
Convolution without Looping using meshgrid

\[
[x,y] = \text{meshgrid}(1:\text{nc},1:\text{nr});
\]

\[
y_{\text{up}} = y-1;
y_{\text{down}} = y+1;
\]

\[
y_{\text{up}} = \min(\text{nr},\max(1,y_{\text{up}})); \quad \% \text{keep } y_{\text{up}} \text{ index within legal range of } [1,\text{nr}]
y_{\text{down}} = \min(\text{nr},\max(1,y_{\text{down}}));
\]

\[
\text{ind}_{\text{up}} = \text{sub2ind}([\text{nr},\text{nc}],y_{\text{up}}(:),x(:)); \quad \% \text{create linear index}
\text{ind}_{\text{down}} = \text{sub2ind}([\text{nr},\text{nc}],y_{\text{down}}(:),x(:));
\]

\[
J_{\text{out}} = 2*J_b(:) - 0.8*J_b(\text{ind}_{\text{up}}) - 0.8*J_b(\text{ind}_{\text{down}}),
J_{\text{out}} = \text{reshape}(J_{\text{out}}, \text{nr}, \text{nc});
\]

\[
\text{figure}; \text{imagesc}(J_{\text{out}});\text{colormap(gray)}
\]

Computation time: 0.024047 sec
Convolution without Looping using meshgrid

\[ [x,y] = \text{meshgrid}(1:nc,1:nr); \]

\[ y_{\text{up}} = y - 1; \]
\[ y_{\text{down}} = y + 1; \]
\[ y_{\text{up}} = \min(nr, \max(1, y_{\text{up}})); \]  % keep \( y_{\text{up}} \) index within legal range of \([1,nr]\)
\[ y_{\text{down}} = \min(nr, \max(1, y_{\text{down}})); \]

\[ \text{ind}_\text{up} = \text{sub2ind}([nr,nc], y_{\text{up}}(:), x(:)); \]  % create linear index
\[ \text{ind}_\text{down} = \text{sub2ind}([nr,nc], y_{\text{down}}(:), x(:)); \]

\[ J_{\text{out}} = 2 * J_b(:) - 0.8 * J_b(\text{ind}_\text{up}) - 0.8 * J_b(\text{ind}_\text{down}); \]
\[ J_{\text{out}} = \text{reshape}(J_{\text{out}}, nr, nc); \]

Convolution without Looping using \text{meshgrid} \( x \) and \( y \) are subscript indice. 

\( J_{b_{xy}} \)
Convolution without Looping using meshgrid

\[
[x,y] = \text{meshgrid}(1:nc,1:nr);
\]

\[
y_{\text{up}} = y-1;
\]
\[
y_{\text{down}} = y+1;
\]
Convolution without Looping using meshgrid

\[
y_{\text{up}} = y-1;
y_{\text{down}} = y+1;
\]

\[
y_{\text{up}} = \min(nr, \max(1, y_{\text{up}})); \quad \% \text{keep } y_{\text{up}} \text{ index within legal range of } [1, nr]
y_{\text{down}} = \min(nr, \max(1, y_{\text{down}}));
\]

\[
y_{\text{up}} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix} 
y = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 & 3 \end{bmatrix}
y_{\text{down}} = \begin{bmatrix} 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 & 3 \end{bmatrix}
\]
Convolution without Looping using meshgrid

\[
y_{\text{up}} = \min(nr, \max(1, y_{\text{up}})); \quad \% \text{keep } y_{\text{up}} \text{ index within legal range of } [1, nr]
\]
\[
y_{\text{down}} = \min(nr, \max(1, y_{\text{down}}));
\]
\[
\text{ind}_{\text{up}} = \text{sub2ind}([nr, nc], y_{\text{up}}(:), x(:)); \quad % \text{create linear index}
\]
\[
\text{ind}_{\text{down}} = \text{sub2ind}([nr, nc], y_{\text{down}}(:), x(:));
\]

\[
\text{linear\_index} = \text{sub2ind}([n\_row, n\_col], \text{row\_subscript}, \text{col\_subscript})
\]
Convolution without Looping using meshgrid

\[
y_{\text{up}} = \min(n_r, \max(1, y_{\text{up}})); \quad \% \text{keep } y_{\text{up}} \text{ index within legal range of } [1, n_r]
\]
\[
y_{\text{down}} = \min(n_r, \max(1, y_{\text{down}}));
\]
\[
\text{ind}_{\text{up}} = \text{sub2ind}([n_r, n_c], y_{\text{up}}(:), x(:)); \quad \% \text{create linear index}
\]
\[
\text{ind}_{\text{down}} = \text{sub2ind}([n_r, n_c], y_{\text{down}}(:), x(:));
\]

\[
\text{linear\_index} = \text{sub2ind}([n_{\text{row}}, n_{\text{col}}], \text{row\_subscript}, \text{col\_subscript})
\]
Convolution without Looping using meshgrid

\[ J_{\text{out}} = 2 \cdot J_b(:) - 0.8 \cdot J_b(\text{ind\_up}) - 0.8 \cdot J_b(\text{ind\_down}); \]
\[ J_{\text{out}} = \text{reshape}(J_{\text{out}}, \text{nr}, \text{nc}); \]

figure; imagesc(J_out);colormap(gray)
With loop

Without loop

Computation time: 0.050154 sec

Computation time: 0.024047 sec
Canny Edge Detection

Objective: to localize edges given an image.

Binary image indicating edge pixels

\[
B(i,j) = \begin{cases} 
1 & \text{if } I(i,j) \text{ is edge} \\
0 & \text{if } I(i,j) \text{ is not edge}
\end{cases}
\]

Original image, I

Edge map image, B
Canny Edge Detection

1. Filter image by derivatives of Gaussian
2. Compute magnitude of gradient
3. Compute edge orientation
4. Detect local maximum
5. Edge linking
Overview

This project focuses on understanding image convolution and edge detection. Both the written part and the programming part are to be done individually. All Matlab functions should follow the names and arguments stated in the problems in order for them to run properly with the grading script. A test script will be provided shortly that will call your functions to ensure they will run inside the grading script.

To submit the assignment, submit a zip file containing all your codes and pdf files (for written part only) via Canvas.

Recall the definition of convolution, $J = I \otimes g$ in 1D as

$$J(i) = \sum_k I(i-k)g(k),$$  \hspace{1cm} (1)

and in 2D as

$$J(i,j) = \sum_{k,l} I(i-k, j-l)g(k,l).$$  \hspace{1cm} (2)

Typically, $I$ is an image, $g$ is a filter, and $J$ is the filter response of $I$ under $g$.

Programming Part

5. Edge detection

Write a Matlab function $E = \text{cannyEdge}(I)$

Compute the Canny edges. Canny edges are defined as local maxima of the image gradient. Following the steps described in the lecture notes:

(a) compute gradient magnitude and orientation,

(b) seek local maximum in the gradient orientation,

(c) continue search in the edge orientation of detected edge point.

We provide the framework of the program. You need to follow the framework and complete the functions $\text{findDerivatives}$, $\text{nonMaxSup}$ and $\text{edgeLink}$. We also include some visualization in the code which can be used for debugging. Rich
1) **Compute Image Gradient**

the first order derivative of Image I in x, and in y direction
Edge Detection, Step 1, Filter out noise and compute derivative:

\[
\frac{\delta}{\delta x} \otimes G
\]

Gradient of Gaussian
Edge Detection, Step 1,
Filter out noise and compute derivative:

\[ \otimes \left( \frac{\delta}{\delta x} \otimes G \right) \]

Image

Smoothed Derivative

\[ I_x \]
\[ I_y \]
Edge Detection, Step 1, Filter out noise and compute derivative:

In matlab:

```matlab
>> [dx,dy] = gradient(G); % G is a 2D gaussian
>> Ix = conv2(I,dx,'same'); Iy = conv2(I,dy,'same');
```
Edge Detection: Step 2
Compute the magnitude of the gradient

In Matlab:
>> Im = sqrt(Ix.*Ix + ly.*ly);
We know roughly where are the edges, but we need their precise location.
Finding the orientation of the edge

- The gradient of an image:
  \[ \nabla f = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right] \]

- The gradient points in the direction of most rapid change in intensity

- The image gradient direction is given by:
  \[ \nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}] \]

- The image gradient direction is given by:
  \[ \theta = \tan^{-1} \left( \frac{\partial f}{\partial y} / \frac{\partial f}{\partial x} \right) \]
  - how does this relate to the direction of the edge?
  \[ \theta_{edge} = \tan^{-1} \left( -\frac{\delta f}{\delta x} / \frac{\delta f}{\delta y} \right) \]
%% define image gradient operator
dy = [1;-1];
dx = [1,-1];

%% compute image gradient in x and y
AA_y = conv2(AA,dy,'same');
AA_x = conv2(AA,dx,'same');

Jy = AA_y(1:end-2,2:end-1);
Jy(1,:) = 0;

Jx = AA_x(2:end-1,1:end-2);
Jx(:,1) = 0;

%% display the image gradient flow
figure(3);clf;imagesc(J);colormap(gray);axis image;
hold on;
quiver(Jx,Jy);
quiver(-Jy,Jx,'r');
quiver(Jy,-Jx,'r');
[gx,gy] = gradient(J);
mag = sqrt(gx.*gx+gy.*gy);  imagesc(mag);colorbar
image gradient direction:
Edge orientation direction:
[gx, gy] = gradient(J);
th = atan2(gy, gx); % or you can use: [th, mag] = cart2pol(gx, gy);
imagesc(th.* (mag > 20)); colormap(hsv); colorbar
• Criteria for an “optimal” edge detector:
  
  • **Good detection**: the optimal detector must minimize the probability of false positives (detecting spurious edges caused by noise), as well as that of false negatives (missing real edges)
  
  • **Good localization**: the edges detected must be as close as possible to the true edges
  
  • **Single response**: the detector must return one point only for each true edge point; that is, minimize the number of local maxima around the true edge

Source: L. Fei-Fei
Discretized pixel locations

(Forsyth & Ponce)
Thesholding

(Forsyth & Ponce)
Non-maximum suppression along the line of the gradient

(Forsyth & Ponce)
Gradient direction

(Forsyth & Ponce)
Local maximum

(Forsyth & Ponce)
No intensity values at \( r \) and \( p \):
Interpolate these intensities using neighbor pixels.

Where is next edge point?
Where is next edge point?

we construct the tangent to the edge curve (which is normal to the gradient at that point) and use this to predict the next points
Where is next edge point?

we construct the tangent to the edge curve (which is normal to the gradient at that point) and use this to predict the next points
Edge Linking: Hysteresis

• Check that maximum value of gradient value is sufficiently large
  – drop-outs? use **hysteresis**
  • use a high threshold to start edge curves and a low threshold to continue them.
Edge Linking: Hysteresis

• Check that maximum value of gradient value is sufficiently large
  – drop-outs? use **hysteresis**
  • use a high threshold to start edge curves and a low threshold to continue them.
Edge Linking: Hysteresis

\[
\begin{bmatrix}
0 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 
\end{bmatrix}
\begin{bmatrix}
0 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 
\end{bmatrix}
\]
Edge Linking: Hysteresis

threshold_high  threshold_low  hysteresis

0 1 0 0 0 1 0 0 0 1 0 0
0 1 0 1 0 1 0 1 0 1 0 1
0 0 0 0 0 1 0 0 0 1 0 0
0 1 0 0 0 1 0 1 0 1 0 1
0 1 0 0 0 1 0 0 0 1 0 0

0 1 0 0 0 1 0 1
0 1 0 1
0 1 0 0
0 1 0 0
Canny Edge Detection

1. Filter image by derivatives of Gaussian
2. Compute magnitude of gradient
3. Compute edge orientation
4. Detect local maximum
5. Edge linking
Canny Edge Implementation

```matlab
img = imread ('Lenna.png');
img = rgb2gray(img);
img = double (img);

% Value for high and low thresholding
threshold_low = 0.035;
threshold_high = 0.175;

% Gaussian filter definition (https://en.wikipedia.org/wiki/Canny_edge_detector)
G = [2, 4, 5, 4, 2; 4, 9, 12, 9, 4;5, 12, 15, 12, 5;4, 9, 12, 9, 4;2, 4, 5, 4, 2];
G = 1/159.* G;

%Filter for horizontal and vertical direction
dx = [1 0 -1];
dy = [1; 0; -1];
```
Canny Edge Implementation

% % Convolution of image with Gaussian
Gx = conv2(G, dx, 'same');
Gy = conv2(G, dy, 'same');

% Convolution of image with Gx and Gy
Ix = conv2(img, Gx, 'same');
Iy = conv2(img, Gy, 'same');
Canny Edge Implementation

angle = atan2(Iy, Ix);

%% Edge angle conditioning
angle(angle<0) = pi+angle(angle<0);
angle(angle>7*pi/8) = pi-angle(angle>7*pi/8);

% Edge angle discretization into 0, pi/4, pi/2, 3*pi/4
angle(angle>=0&angle<pi/8) = 0;
angle(angle>=pi/8&angle<3*pi/8) = pi/4;
angle(angle>=3*pi/8&angle<5*pi/8) = pi/2;
angle(angle>=5*pi/8&angle<=7*pi/8) = 3*pi/4;

Continuous angle

Discretized angle
Canny Edge Implementation

%Calculate magnitude
magnitude = sqrt(lx.*lx+ly.*ly);
edge = zeros(nr, nc);

%% Non-Maximum Supression
dedge = non_maximum_suppression(magnitude, angle, edge);
edge = edge.*magnitude;

Gradient magnitude
Localized edge
Canny Edge Implementation

%%% Hysteresis thresholding
% for weak edge
threshold_low = threshold_low * max(edge(:));
% for strong edge
threshold_high = threshold_high * max(edge(:));
linked_edge = zeros(nr, nc);
linked_edge = hysteresis_thresholding(threshold_low, threshold_high, linked_edge, edge);
The rows of black and white squares are all parallel. The vertical zigzag patterns disrupt our horizontal perception.

Cornelia Fermüller
Figure 1: Type 1 Edge

This edge can be represented by a function. The picture below shows the gray value changes from black to white in the horizontal direction.

Figure 2

The edge is at the location of the inflection point on the curve, indicated by the black dot. If we were to smooth this image, it would look like this.
The edge is at the location of the inflection point on the curve, indicated by the black dot. If we were to smooth this image, it would look like this.

![Image of edge detection with inflection point]

**Figure 3**

As you can see, smoothing the image does not change the location of the edge.
The second case is a line on a background of different intensity.

![Type 2 Edge](image)

**Figure 4: Type 2 Edge**

This again can be represented as a function with two inflection points representing the edges at the boundaries of the line and the background regions.

![Function with inflection points](image)

**Figure 5**
When the image is smoothed the edges drift apart as shown.

Figure 6
The third case is a gray line between a bright and a dark region.

When this image is smoothed the edges at the boundary of the line move toward each other.

Figure 7: Type 3 Edge

Figure 8
If we smooth the image and then apply edge detection, we obtain edges as shown below.
The rows of black and white squares are all parallel. The vertical zigzag patterns disrupt our horizontal perception.
If we zoom in and look at the edges we notice that the added squares compensate for the drifting of the lines. There is still “waviness” to the edges, but it is too weak to be perceived.
Is Edge Detection Solved?
⇒ Edge detector

⇒ Human segmentation
Edge Formation Factors

Depth discontinuity

Illumination discontinuity

Surface color discontinuity

Surface normal discontinuity
Image Pyramid, CIS581
Different scale of image encodes different edge response.
Image Pyramids

Known as a Gaussian Pyramid [Burt and Adelson, 1983]
• In computer graphics, a *mip map* [Williams, 1983]
• A precursor to *wavelet transform*
Image sub-sampling

Throw away every other row and column to create a $1/2$ size image - called *image sub-sampling*
Image sub-sampling

1/2

1/4 (2x zoom)

1/8 (4x zoom)

Why does this look so bad?
Sampling

Good sampling:
• Sample often or,
• Sample wisely

Bad sampling:
• see aliasing in action!
Gaussian pre-filtering

Solution: filter the image, *then* subsample

- Filter size should double for each $\frac{1}{2}$ size reduction. Why?
Subsampling with Gaussian pre-filtering

Solution: filter the image, *then* subsample

- Filter size should double for each $\frac{1}{2}$ size reduction. Why?
- How can we speed this up?