Edge Detection

1 Why Detect Edges?
   - Information reduction
     - Replace image by a cartoon in which objects and surface markings are outlined ⇒ create line drawing description
     - These are the most informative parts of the image
   - Biological plausibility
     - Initial stages of mammalian vision systems involve detection of edges and local features
   - Applications
     - Object recognition, stereo, texture analysis, motion analysis, image enhancement, image compression

What Causes Image Intensity Changes?

1 Many types of physical events cause intensity changes
   - Surface reflectance discontinuity - change in the fraction of light incident on the surface that is reflected to viewer
   - Illumination discontinuity - shadow
   - Surface orientation (normal) discontinuity
   - Depth discontinuity - at occluding contour, where surface orientation is perpendicular to line of sight
What type of real-world edge is this contrast border?

Need global information too.
What Does an Edge Look Like?

1. Step

2. Ramp

3. Roof

4. Line (bar)
fine scale high threshold

coarse scale, high threshold
1 Sum activity from an array of oriented simple cells
A “hypercolumn” = complete set of orientations

Craik-O’Brien-Cornsweet Illusion
Koffka Ring

Luminance Differences Affect our Perceptions

1. Artists use the technique of “equiluminance” to blur outlines and suggest motion. We cannot perceive the edges of objects where object and background have the same luminance. If parts of a painting are equiluminant, their positions become ambiguous. They may seem to shift position or to float.

2. Equiluminant colors have special properties, e.g., they can make a painting appear unstable.
Equiluminant Colors

1 An object that can be seen by both subdivisions of the visual system will be perceived accurately. But if the two subdivisions are not balanced in their response to an object, it may look peculiar.

1 For example, an object defined by equiluminant colors can be seen by the What system but is invisible (or poorly seen) by the Where system. It may seem flat, it may seem to shift position, or it may seem to float ambiguously because there is too little luminance contrast to provide adequate information about its three-dimensional shape, its location in space, or its motion (or lack of it).

1 Conversely, something defined by very low contrast contours is seen by the Where system but not the What system and may seem to have depth and spatial organization but no clear shape.

Detail from Richard Anuszkiewicz’s Plus Reversed, 1960

The red and blue seem to move around because they are equiluminant
Edge Detection Goals

1. **Good detection**: Low false alarm rate and low false dismissal rate ⇒ maximize signal-to-noise (S/N) ratio
2. **Good localization**: Mark point closest to “center” of true edge ⇒ minimize distance between marked point and center
3. **Uniqueness**: Only one response to a single edge
4. **Good property measurement**: Orientation, contrast, etc.

Edge Operator Properties

1. **Shift invariant** (translation invariant, position invariant)
   - If \( g(x,y) = \text{Op}[f(x,y)] \) then \( g(x-a, y-b) = \text{Op}[f(x-a, y-b)] \)
2. **Isotropic** (rotation invariant) vs. non-isotropic
3. **Linear** vs. non-linear
   - \( \text{Op}[a f_1(x,y) + b f_2(x,y)] = a \text{Op}[f_1(x,y)] + b \text{Op}[f_2(x,y)] = a g_1(x,y) + b g_2(x,y) \)
4. **Scale** (operator neighborhood size)
   - \( g(x,y) = \text{Op}[f(x+a, y+b), \ \forall -k < a, b < k] \)
5. **Convolution** (linear and shift-invariant)
Edge and Local Feature Detection Methods

1. Gradient-based edge detection
2. Edge detection by function fitting
3. Second derivative edge detectors
4. Edge linking and the construction of the chain graph

1D Edge Detection

1. An ideal edge is a step function

\[ f(x) \]

\[ r(x) \]
1D Edge Detection

The first derivative of $I(x)$ has a peak at the edge

The second derivative of $I(x)$ has a zero crossing at the edge

More realistically, image edges are blurred and the regions that meet at those edges have noise or variations in intensity

- Blur - high first derivatives near edges
- Noise - high first derivatives within regions that meet at edges
Edge Detection in 2D

1. Let $I(x,y)$ be the image intensity function. It has derivatives in all directions
   - $\frac{\partial I(x,y)}{\partial x} = \lim_{\Delta x \to 0} \frac{I(x+\Delta x,y) - I(x,y)}{\Delta x} = \frac{I(u+1,v) - I(u,v)}{1}$
   - Gradient of $I(x,y)$ is a vector $\nabla I(x,y) = [\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}]^T$ specifying the direction of greatest rate of change in intensity (i.e., perpendicular to the edge’s direction)
   - From gradient can determine the direction in which the first derivative is highest, and the magnitude of the first derivative in that direction
     - Magnitude = $\sqrt{(\frac{\partial I}{\partial x})^2 + (\frac{\partial I}{\partial y})^2}$
     - Direction = $\tan^{-1}(\frac{\frac{\partial I}{\partial y}}{\frac{\partial I}{\partial x}})$

Computing First Derivative

1. To compute first derivative in direction $\theta$, calculate from linear combination of derivatives from any two non-collinear directions
   - $x = x'\cos \theta - y'\sin \theta$
   - $y = x'\sin \theta + y'\cos \theta$
   - $\frac{\partial I}{\partial x'} = \frac{\partial I}{\partial x} \frac{\partial x}{\partial x'} + \frac{\partial I}{\partial y} \frac{\partial y}{\partial x'}$
   - $= \frac{\partial I}{\partial x} \cos \theta + \frac{\partial I}{\partial y} \sin \theta$
   - $\frac{\partial I}{\partial y'} = -\frac{\partial I}{\partial x} \sin \theta + \frac{\partial I}{\partial y} \cos \theta$
   - $(\frac{\partial I}{\partial x'})^2 + (\frac{\partial I}{\partial y'})^2 = (\frac{\partial I}{\partial x})^2 + (\frac{\partial I}{\partial y})^2$
   - So, sum of squares of first derivative is isotropic, non-linear
   - Similarly, all derivatives of odd order raised to an even power are isotropic
Gradient

1 Gradient = $[\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}]^T$

1 What direction is first derivative a maximum?
   - Set $\frac{\partial^2 I}{\partial \theta^2} (\frac{\partial I}{\partial x}) = 0$, and solve for $\theta$
     $\Rightarrow \frac{\partial^2 I}{\partial \theta^2} (\frac{\partial I}{\partial x} \cos \theta + \frac{\partial I}{\partial y} \sin \theta) = 0$
     $\Rightarrow \theta = \tan^{-1} (\frac{\partial I}{\partial y} / \frac{\partial I}{\partial x})$

1 Gradient direction is perpendicular to “edge direction”
1 Gradient magnitude is isotropic

$$|\nabla I| = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2}$$

Edge Detection in 2D

1 With a digital image, the partial derivatives are replaced by finite differences:
   - $\Delta I = I(u+1, v) - I(u, v)$
   - $\Delta I = I(u, v) - I(u, v+1)$

1 An alternative (Sobel)
   - $\Delta_{sobel_x} I = I(u+1, v+1) + 2I(u+1, v) + I(u+1, v-1) - I(u-1, v+1) - 2I(u-1, v) - I(u-1, v-1)$
   - $\Delta_{sobel_y} I = I(u-1, v-1) + 2I(u, v-1) + I(u+1, v-1) - I(u-1, v+1) - 2I(u, v+1) - I(u+1, v+1)$

1 Roberts’s “Cross”
   - $\Delta I = I(u, v) - I(u+1, v-1)$
   - $\Delta I = I(u, v-1) - I(u+1, v)$
   - $\Delta I = I(u, v) - I(u+1, v)$

\[\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\]
Gradient Operator Example

1. $I = 0 0 0 1 2 3 4 4 4 8 8 8 3 3 3$
2. $\Delta_x = -1 1$
3. $\Delta_x I = * 0 0 1 1 1 1 0 0 4 0 0 -5 0 0$

Figure 1: Detail of an eye (top left), the corresponding gradient field (top right), the magnitude of the gradient (bottom left), and the magnitude of the image intensity (bottom right). The area in the box is analyzed further in the text.

Figure 2: The image intensity function (left) is the box shown in figure 1, and the corresponding gradient magnitude (right).

Figure 3: The same plots as in figure 2 but without smoothing. Noise is very visible.
### Directional Edge Operators

1. **Kirsch 8-direction masks**

   \[ k_0 = \begin{pmatrix} -3 & 0 & 5 \\ -3 & 5 & 5 \\ -3 & -3 & -3 \end{pmatrix} \]

   \[ k_1 = \begin{pmatrix} -3 & 0 & 5 \\ 0 & 5 & 5 \\ -3 & -3 & -3 \end{pmatrix} \]

   \[ k_2 = \begin{pmatrix} -3 & 0 & -3 \\ -3 & 5 & 5 \\ -3 & -3 & -3 \end{pmatrix} \]

   \[ k_3 = \begin{pmatrix} -3 & 0 & 3 \\ -3 & 5 & 5 \\ -3 & -3 & -3 \end{pmatrix} \]

1. **Gradient magnitude** = \( \max_{k_n} k_n \)

1. **Gradient direction** = \( 45^\circ \arg \max_{k_n} k_n \)
Directional Edge Operators

1. Robinson masks

\[
\begin{bmatrix}
-1 & 0 & 1 \\
-2 & -1 & 0 \\
0 & 1 & 2
\end{bmatrix}
\]

1. Nevatia and Babu used 6 5x5 masks, detecting orientations at multiples of 30°

Edge Detection in 2D

1. How do we combine the directional derivatives to compute the gradient magnitude?
   - Use the root mean square (RMS) as in the continuous case
   - Sum of the absolute values of the directional derivatives
   - Maximum of the absolute values of the directional derivatives

1. Advantages of the latter
   - Avoids computing square root (although this can be done using table lookups)
   - Keeps result in the same range as the original image
   - Gives a magnitude that is invariant with respect to orientation of the edge in the image (\(\Delta_{\text{sobel,x}}\) overestimates diagonal edges and \(\Delta_x\) overestimates horizontal and vertical edge magnitudes)
Finite Differences and Noise

1. Finite difference filters respond strongly to noise
   - obvious reason: image noise results in pixels that look very different from their neighbors
2. Generally, the larger the noise the stronger the response

1. What is to be done?
   - intuitively, most pixels in images look quite a lot like their neighbors
   - this is true even at an edge; along the edge they’re similar, across the edge they’re not
   - suggests that smoothing the image should help, by forcing pixels different from their neighbors (=noise pixels?) to look more like their neighbors

Finite Differences Responding to Noise

Increasing noise ->
(this is zero mean additive gaussian noise)
Combining Smoothing and Differentiation - Fixed Scale

1. Local operators like the Roberts operator give high responses to any intensity variation
   - local surface texture
   - noise from the sensing process
2. If the picture is first smoothed by an averaging process, then these local variations are removed and what remains are the “prominent” edges
3. Example: \( I_{2x2}(u,v) = 1/4[I(u,v) + I(u+1,v) + I(u,v+1) + I(u+1,v+1)] \)

Smoothing - Basic Problems

1. What function should be used to smooth or average the image before differentiation?
   - **Mean** smoothing (aka box filters or uniform smoothing or averaging)
     - easy to compute (linear)
     - for large smoothing neighborhoods assigns too much weight to points far from an edge
   - **Median** smoothing
     - doesn’t blur corners
     - unaffected by outliers
   - **Gaussian** smoothing
     - \[ [1/(2\pi\sigma^2)] e^{-(u^2 + v^2)/2\sigma^2} \]
Example: Smoothing by Averaging
The Mystery of the Mona Lisa Smile

1. Margaret Livingstone has conjectured that the elusive smile is because of the differences in resolution in the fovea and the periphery of the retina.
Changing small details can affect the mood we perceive. A few pixels of change switches her from happy to sad. Adding random “noise” also changes our perception of her expression.
Smoothing and Convolution

1. The convolution of two functions, \( f(x) \) and \( g(x) \), is defined as

\[
h(x) = \int g(x') f(x-x') \, dx' = g(x) * f(x)
\]

1. When \( f \) and \( g \) are discrete and when \( g \) is nonzero only over a finite range \([-n, n]\), this integral is replaced by the summation

\[
h(u) = \sum g(i) f(u-i)
\]

Example of 1D Convolution

\[
h(4) = \sum_{i=-2}^{2} g(i) f(4-i)
\]

\[
= g(-2) f(6) + g(-1) f(5) + g(0) f(4) + g(1) f(3) + g(2) f(2)
\]

Note: Values in \( h \) have been divided by \( \sum g = 13 \)
Smoothing and Convolution

1. These integrals and summations extend simply to functions of two variables:

\[ h(u,v) = g(u,v) * f(u,v) = \sum \sum g(i,j) f(u-i, v-j) \]

1. Convolution computes the weighted sum of the gray levels in each \( n \times n \) neighborhood of the image, \( f \), using the “kernel” matrix of weights, \( g \)

1. Convolution is a **linear** operator because

\[ g*(af_1 + bf_2) = a(g*f_1) + b(g*f_2) \]

---

Convolution

- \( f \)
- \( h = h(x,y) \)
- \( g \) (kernel matrix of weights)

\[ g_{1,1} \quad g_{1,2} \quad g_{1,3} \]
\[ g_{2,1} \quad g_{2,2} \quad g_{2,3} \]
\[ g_{3,1} \quad g_{3,2} \quad g_{3,3} \]

Matrices of gray levels:

\[ f_{1,1} \quad f_{1,2} \quad f_{1,3} \]
\[ f_{2,1} \quad f_{2,2} \quad f_{2,3} \]
\[ f_{3,1} \quad f_{3,2} \quad f_{3,3} \]
2D Convolution

\[ h(5,5) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} g(i, j) f(5-i, 5-j) \]

\[ = g(-1,-1) f(6,6) + g(-1,0) f(6,5) + g(-1,1) f(6,4) \]
\[ + g(0,-1) f(5,6) + g(0,0) f(5,5) + g(0,1) f(5,4) \]
\[ + g(1,-1) f(4,6) + g(1,0) f(4,5) + g(1,1) f(4,4) \]

Smoothing and Convolution

4.2. LINEAR SYSTEMS

\[ h[i,j] = A p_1 + B p_2 + C p_3 + D p_4 + E p_5 + F p_6 + G p_7 + H p_8 + I p_9 \]
Properties of Convolution

1. Commutative
2. Associative
3. Cascadable
4. Linear, shift-invariant
5. Any linear, shift-invariant operation can be implemented as a convolution
6. Convolution in spatial domain = multiplication in frequency domain. I.e., \( g = f * h \) iff \( G = F \cdot H \), where \(*\) denotes convolution and \( F = \mathcal{F}(f) \)
7. An alternative for computing \( g \): \( g = \mathcal{F}^{-1}(F \cdot H) \)

Combining Smoothing and Edge Detection

1. Let \( S = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \)
2. \( \Delta_x = \begin{bmatrix} -1 & 1 \end{bmatrix} \)
3. Smooth, then differentiate:
   \( \Delta_x * (S * I) \), where \( I \) is original image and \(*\) is convolution
   \[ = (\Delta_x * S) * I \]
   \[ = \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \]
4. \( \Delta_x * S = \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} = \Delta_{\text{Prewitt}_x} \)
   \[ = \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \]
To smooth an image using a Gaussian filter, \( e^{-(u^2 + v^2)/2\sigma^2} \), we must

- Choose an appropriate value for \( \sigma \), which controls how quickly the Gaussian falls to near 0
  - Small \( \sigma \) produces filter which drops to near 0 quickly - can be implemented using small digital array of weights
  - Large \( \sigma \) produces a filter which drops to near 0 slowly - will be implemented using a larger size digital array of weights

- Determine the size weight array needed to adequately represent that Gaussian
  - To represent 98.76% of the area under the Gaussian using \( \sigma = \sigma_0 \) use a mask of size \( 5\sigma_0 \times 5\sigma_0 \)
  - In practice, size of mask is often closer to size \( 3\sigma \times 3\sigma \)
  - Weight array needs to be of odd size \( (2k+1) \times (2k+1) \) to allow for symmetry

To smooth an image using a Gaussian filter we must

- Sample the Gaussian by integrating it over the square pixels of the array of weights and multiplying by a scale factor to obtain integer weights

- Because we have truncated the Gaussian, the weights will not sum to 1.0 x scale factor
  - In “flat” areas of the image we expect our smoothing filter to leave the image unchanged
  - But if the filter weights do not sum to 1.0 x scale factor, it will either amplify (> 1.0) or de-amplify (< 1.0) the image
Gaussian Smoothing

1. Gaussian smoothing steps
   - Normalize the weight array by dividing each entry by the sum of the all of the entries (integral equal to 1)
   - Convert to integers

1. Advantages of Gaussian filtering
   - Rotationally symmetric (for large filters)
   - Filter weights decrease monotonically from central peak, giving most weight to central pixels
   - Simple and intuitive relationship between size of $\sigma$ and size of objects whose edges will be detected in image.
   - The Gaussian is separable

Building Discrete Gaussian Kernels

1. Given $\sigma$ and $w$, compute real-valued weights for $w \times w$ kernel array
1. Divide all entries by minimum weight and round result to nearest integer, obtaining kernel $G$
1. Compute normalization factor $n = \sum$ weights
1. $I' = 1/n \ (G * I)$
A function \( g(x, y) \) is **separable** if \( g(x, y) = g_1(x) \cdot g_2(y) \)

The Gaussian function is separable:

\[
e^{-\left(\frac{x^2 + y^2}{2 \sigma^2}\right)} = e^{-\frac{x^2}{2 \sigma^2}} \cdot e^{-\frac{y^2}{2 \sigma^2}}
\]

First convolve the image with a 1D horizontal filter

Then convolve the result of the first convolution with a 1D vertical filter

For a \( k \times k \) Gaussian filter, 2D convolution requires \( k^2 \) multiplications and \( k^2 - 1 \) additions per pixel

But using the separable filters, we reduce this to \( 2k \) multiplications and \( 2k - 1 \) additions per pixel
Advantages of Gaussians

1. Cascading Gaussians: Convolution of a Gaussian with itself is another Gaussian
   - So, we can first smooth an image with a small Gaussian
   - Then, we convolve that smoothed image with another small Gaussian and the result is equivalent to smoother the original image with a larger Gaussian
   - If we smooth an image with a Gaussian having standard deviation $\sigma$ twice, then we get the same result as smoothing the image with a Gaussian having standard deviation $\sqrt{2}\sigma$

Combining Smoothing and Differentiation - Fixed Scale

1. Non-maxima suppression - Retain a point as an edge point if
   - its gradient magnitude is higher than a threshold
   - its gradient magnitude is a local maximum in the gradient direction
2. “Lateral inhibition” process
Summary of Basic Edge Detection Steps

1. Smooth the image to reduce the effects of local intensity variations
   - Choice of smoothing operator practically important
2. Differentiate the smoothed image using a digital gradient operator that assigns a magnitude and direction of the gradient at each pixel
   - Mathematically, we can apply the digital gradient operator to the digital smoothing filter, and then just convolve the differentiated smoothing filter to the image
   - Requires using a slightly larger smoothing array to avoid border effects
3. Threshold the gradient magnitude to eliminate low contrast edges

1. Apply a non-maximum suppression step to thin the edges to single pixel wide edges
   - Smoothing will produce an image in which the contrast at an edge is spread out in the neighborhood of the edge
   - Thresholding operation will produce thick edges
Canny Edge Operator

1. Design operator that is optimal with respect to good detection, good localization, and unique response criteria

   **Good detection criterion**: Maximize SNR, which is the ratio of the step size of a step edge over the standard deviation of the 0-mean, additive Gaussian noise

   **Good localization criterion**: Maximize inverse distance of detected edge from true edge

   Optimal 1D step-edge detector = detector that maximizes the product of the two terms above subject to third criterion

   First derivative of a Gaussian is within 20% of optimal

Directional Derivatives

1. Isotropic operators smooth **across** edges
   ⇒ Poor localization

   Instead, average pixels **along** edge direction only
   ⇒ Use a set of directional derivatives at a large number of orientations

   $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$, $\begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}$ etc.
Canny Edge Operator

1. Gaussian smooth image: \( G * I \)

1. Estimate gradient direction (i.e., edge normal), \( n \), at each pixel:
\[
\mathbf{n} = \frac{\nabla(G * I)}{\|\nabla(G * I)\|}
\]

1. Compute first derivative in the gradient direction, \( n \). That is, \( \partial G / \partial n * I \)

1. Suppress non-maxima
   - If value is not a local max in direction \( n \), then change value to 0
   - Or, find zero-crossings in direction \( n \). I.e., zero-crossings of
     \[
     \partial[\partial G / \partial n] * I / \partial n = \partial^2 G * I / \partial n^2
     \]

Canny Edge Operator

1. To get rid of spurious edges created by noise (false positives) and to fill in missing edges (false negatives) use **hysteresis thresholding**:
   - Define two thresholds, \( t_{\text{high}} \) and \( t_{\text{low}} \)
   - Mark all pixels with edge magnitude > \( t_{\text{high}} \) as edge pixels
   - For each marked pixel, search in edge direction, both ways, marking all pixels with edge magnitude > \( t_{\text{low}} \) as edge pixels. Stop search when a pixel is reached with edge magnitude < \( t_{\text{low}} \)
Laplacian Edge Detectors

1. Directional second derivative in direction of gradient has a zero crossing at gradient maximum
2. Can approximate directional second derivative with Laplacian:
   \[ \nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} \]
3. Laplacian is lowest order linear isotropic operator
4. Digital approximation (2nd forward difference) is
   \[ \nabla^2 I(u,v) = \left[ I(u+1,v) + I(u-1,v) + I(u,v+1) + I(u,v-1) \right] - 4I(u,v) \]
   \[ = \left[ I(u+1,v) - I(u,v) \right] - \left[ I(u,v) - I(u-1,v) \right] + \left[ I(u,v+1) - I(u,v) \right] - \left[ I(u,v) - I(u,v-1) \right] \]

\[
\begin{bmatrix}
0 & 1 & 0 \\
1 & -4 & 1 \\
0 & 1 & 0
\end{bmatrix}
\]

Laplacian Examples

1. \( I = \ldots 2 \ 2 \ 2 \ 8 \ 8 \ 8 \ldots \)
2. \( \nabla^2 = 1 \ -2 \ 1 \)
   \[ \Rightarrow \nabla^2 I = \ldots 0 \ 0 \ 0 \ 6 \ -6 \ 0 \ 0 \ 0 \ldots \]

1. \( I = \ldots 2 \ 2 \ 2 \ 5 \ 8 \ 8 \ 8 \ldots \)
   \[ \Rightarrow \nabla^2 I = \ldots 0 \ 0 \ 0 \ 3 \ 0 \ -3 \ 0 \ 0 \ldots \]
Problems with Laplacian of Small Size

1. Responds most strongly at isolated points (spots)  
   ⇒ noise sensitive
2. Responds strongly at lines and endpoints of lines
3. Responds relatively less at edges
4. Ramp edges detected at two ends, not once at center
Using Laplacian for Edge Enhancement

1. Given a blurred image $g \approx I' = \frac{1}{5} \begin{bmatrix} I \ast 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 \end{bmatrix}$

1. $\nabla^2 I \propto I - I'$

1. $g - k\nabla^2 g \approx I' - k(-\frac{1}{5} (I - I'))$

1. $= \frac{k}{5} I + I'(1 - \frac{k}{5})$

1. $\approx I$ when $k=5$

1. $\implies$ Can deblur an image by subtracting a multiple of its Laplacian

1. Simulates Mach Band effect in human visual system

1. Analogous to unsharp masking technique in photography

Mach Bands

1. In 1865, Mach discovered that we perceive an edge (change of intensity) at lines that separate surface regions with identical intensity on both sides but different intensity derivatives

1. It is caused by the lateral inhibition of the receptors in the eyes
Mach Bands

Actual brightness

Perceived by you

Laplacian Edge Detectors

1. Laplacians are also combined with smoothing for edge detectors
   - Take the Laplacian of a Gaussian-smoothed image - called the Laplacian-of-Gaussian (LoG), Mexican Hat operator, Difference-of-Gaussians (DoG), Marr-Hildreth, $\nabla^2 G$ operator
   - Locate the zero-crossing of the operator
     - these are pixels whose LoG is positive and which have neighbor’s whose LoG is negative or zero
   - Usually, measure the gradient or directional first derivatives at these points to eliminate low contrast edges
\[ \nabla^2 G_{\sigma}(x, y) = \frac{1}{2\pi\sigma^4} \left( 2 - \frac{x^2 + y^2}{\sigma^2} \right) e^{-\frac{x^2 + y^2}{2\sigma^2}} \]

Laplacian of Gaussian (LoG)
LoG Properties

1. Linear, shift invariant $\Rightarrow$ convolution
2. Circularly symmetric $\Rightarrow$ isotropic
3. Size of LoG operator approximately $6\sigma \times 6\sigma$
4. LoG is separable
5. LoG $\approx G_{\sigma_1} - G_{\sigma_2}$, where $\sigma_1 = 1.6\sigma_2$

1. Analogous to spatial organization of receptive fields of retinal ganglion cells, with a central excitatory region and an inhibitory surround

---

Zero-crossing Detection

if $((\nabla^2 G(p) < -T) \land
(\exists q \text{ neighbor } q : \nabla^2 G(q) > T))$

$\lor ((\nabla^2 G(p) > T) \land
(\exists q : \nabla^2 G(q) < -T))$

then mark $p$ as an edge point
Receptive Field

- Region on retina that influences activity of a ganglion cell
- Firing rate of neuron
- Excitatory center, inhibitory surround

Ganglion cells detect differences in luminance at a border, line, or spot

Retina is NOT a spot-meter
Lateral Inhibition Explanation

B < A
B inhibited by surround

D > E
D less inhibited by surround

Hermann Grid

1. Illustrates lateral inhibition
Visual illusions

Lateral Inhibition Explanation

- Inhibited less by white stripes
- Inhibited more by white stripes
The Scale Space Problem

1. Usually, any single choice of $\sigma$ does not produce a good edge map
   - A large $\sigma$ will produce edges from only the largest objects, and they will not accurately delineate the object because the smoothing reduces shape detail
   - A small $\sigma$ will produce many edges and very jagged boundaries of many objects

2. Scale-space approaches
   - Detect edges at a range of scales $[\sigma_1, \sigma_2]$
   - Combine the resulting edge maps
     - Trace edges detected using large $\sigma$ down through scale space to obtain more accurate spatial localization
Figure 3. Several signals, with their maximum-slope descriptions. These are "toplot" descriptions, generated automatically and without thresholds. You should compare the descriptions to your own freehand "toplot" sketches. (The tiny sine and square waves are synthetic signals.)

Figure 4. A signal with its internal tree, represented as a rectangular translation of subspaces. Each rectangle in a scale, indicating an interval on the signal, and the scale interval over which the signal interval exists, translates the (x,y)-plane. See fig. 4 for an illustration of the tree.

This internal tree may be viewed in two ways: as describing the signal simultaneously at all scales, or as generating a family of single-scale descriptions, each defined by a subset of nodes in the tree that cover the x-axis. On the second interpretation, one moves across the family...
Pyramids

1. Very useful for representing images
2. Pyramid is built by using multiple copies of image
3. Each level in the pyramid is 1/4 the area of previous level, i.e., each dimension is 1/2 resolution of previous level
Gaussian Pyramid
Gaussian Pyramid

1. Multiresolution, low-pass filter
2. Hierarchical convolution
   - $G_0$ = input image
   - $G'_k(u, v) = \sum\sum w(m, n) G_{k-1}(u-m, v-n)$ ; smooth
   - $G_k(u, v) = G'_k(2u, 2v), \ 0 < u, v < 2^{N-k}$ ; sub-sample
3. $w$ is a small (e.g., 5 x 5) separable generating kernel, e.g., $1/16 \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix}$
4. Cascading $w$ equivalent to applying large kernel
   - Effective size of kernel at level $k = 2M(2^k - 1) + 1$, where $w$ has width $2M+1$
   - Example: Let $M=1$. If $k=1$ then equivalent size = 5; $k=2$ then equivalent size = 13; $k=3$ then equivalent size = 27

Gaussian Pyramids

$$g_l(u,v) = \sum_{m=-2}^{2} \sum_{n=-2}^{2} w(m,n) g_{l-1}(2u+m,2v+n)$$

$$g_l = REDUCE \left[ g_{l-1} \right]$$
Reduce (1D)

\[ g_l(u) = \sum_{m=-2}^{2} \hat{w}(m) g_{l-1}(2u+m) \]

\[ g_l(2) = \hat{w}(-2) g_{l-1}(4-2) + \hat{w}(-1) g_{l-1}(4-1) + \hat{w}(0) g_{l-1}(4) + \hat{w}(1) g_{l-1}(4+1) + \hat{w}(2) g_{l-1}(4+2) \]

\[ g_l(2) = \hat{w}(-2) g_{l-1}(2) + \hat{w}(-1) g_{l-1}(3) + \hat{w}(0) g_{l-1}(4) + \hat{w}(1) g_{l-1}(5) + \hat{w}(2) g_{l-1}(6) \]
Gaussian Pyramids

\[ g_{l,n}(u,v) = \sum_{p=-2}^{2} \sum_{q=-2}^{2} w(p,q) g_{l,n-1}(\frac{u-p}{2}, \frac{v-q}{2}) \]

\[ g_{l,n} = \text{EXPAND}[g_{l,n-1}] \]

Expand (1D)

\[ g_{l,n}(u) = \sum_{p=-2}^{2} \hat{w}(p) g_{l,n-1}(\frac{u-p}{2}) \]

\[ g_{l,n}(4) = \hat{w}(-2) g_{l,n-1}(\frac{4-2}{2}) + \hat{w}(-1) g_{l,n-1}(\frac{4-1}{2}) + \]

\[ \hat{w}(0) g_{l,n-1}(\frac{4}{2}) + \hat{w}(1) g_{l,n-1}(\frac{4+1}{1}) + \hat{w}(2) g_{l,n-1}(\frac{4+2}{2}) \]

\[ g_{l,n}(4) = \hat{w}(-2) g_{l,n-1}(1) + \hat{w}(0) g_{l,n-1}(2) + \hat{w}(2) g_{l,n-1}(3) \]
Expand

**Gaussian Pyramid**

\[
g_{l,1} = \text{EXPAND}[g_{1,1}]
\]

**Convolution Mask**

\[
[w(-2), w(-1), w(0), w(1), w(2)]
\]
Convolution Mask

1. Separable
   \[ w(m, n) = \hat{w}(m)\hat{w}(n) \]

2. Symmetric
   \[ \hat{w}(u) = \hat{w}(-u) \]
   \[ [c, b, a, b, c] \]

Convolution Mask

1. The sum of mask should be 1:
   \[ a + 2b + 2c = 1 \]

2. All nodes at a given level must contribute the same total weight to the nodes at the next higher level:
   \[ a + 2c = 2b \]
Convolution Mask

\[ \hat{w}(0) = a \]
\[ \hat{w}(-1) = \hat{w}(1) = \frac{1}{4} \]
\[ \hat{w}(-2) = \hat{w}(2) = \frac{1}{4} - \frac{a}{2} \]

\[ a = A \Rightarrow \text{Gaussian} \quad a = 0.5 \Rightarrow \text{Triangular} \]

Algorithm

1. Apply 1D mask to alternate pixels along each row of image
2. Apply 1D mask to each pixel along alternate columns of resultant image from previous step
Laplacian Pyramids

1. Similar to results of edge detection
2. Most pixels are 0
3. Can be used for image compression:

\[ L_1 = g_1 - EXPAND[g_2] \]
\[ L_2 = g_2 - EXPAND[g_3] \]
\[ L_3 = g_3 - EXPAND[g_4] \]
Laplacian Pyramid

1. Computes a set of bandpass filtered versions of image
   \[ L_k = G_k - w \ast G_k \]
   \[ = G_k - \text{Expand}(G_{k+1}) \]
   \[ L_N = G_N \] (apex of Laplacian pyr = apex of Gaussian pyr)
1. Separates features by their scale (size)
1. Enhances features
1. Compact representation
1. \[ \sum L_k = (G_0 - G_1) + (G_1 - G_2) + ... + (G_{N-1} - G_N) + G_N \]
   \[ = G_0 \]
Gaussian and Laplacian Pyramids

Coding using the Laplacian Pyramid

• Compute Gaussian pyramid

\[ g_1, g_2, g_3, g_4 \]

• Compute Laplacian pyramid

\[ \begin{align*}
L_1 &= g_1 - \text{EXPAND}[g_2] \\
L_2 &= g_2 - \text{EXPAND}[g_3] \\
L_3 &= g_3 - \text{EXPAND}[g_4] \\
L_4 &= g_4
\end{align*} \]

• Code Laplacian pyramid
Decoding using Laplacian pyramid

1. Decode Laplacian pyramid
2. Compute Gaussian pyramid from Laplacian pyramid:
   \[ g_4 = L_4 \]
   \[ g_3 = \text{EXPAND}[g_4] + L_3 \]
   \[ g_2 = \text{EXPAND}[g_3] + L_2 \]
   \[ g_1 = \text{EXPAND}[g_2] + L_1 \]
3. \( g_1 \) is reconstructed image

Image Compression

Laplacian Pyramid

After Quantization
Image Compression

Image Compositing by Pyramid Blending

1. **Given**: Two $2^n \times 2^n$ images
2. **Goal**: Create an image that contains left half of image A and right half of image B
3. **Algorithm**
   - Compute Laplacian pyramids, $LA$ and $LB$, from images $A$ and $B$
   - Compute Laplacian pyramid $LS$ by copying left half of $LA$ and right half of $LB$. Pyramid nodes down the center line = average of corresponding $LA$ and $LB$ nodes ⇒ blend along center line
   - Expand and sum levels of $LS$ to obtain output image $S$
Combining Apple & Orange

Combining Apple & Orange (using Pyramids)
Image Compositing from Arbitrary Regions

1. Given: Two $2^n \times 2^n$ images and one $2^n \times 2^n$ binary mask
2. Goal: Output image containing image A where mask=1, and image B where mask=0
3. Algorithm:
   - Construct Laplacian pyramids $LA$ and $LB$ from images A and B
   - Construct Gaussian pyramid $GR$ from mask $R$
   - Construct Laplacian pyramid $LS$:
     \[ LS_k(u, v) = GR_k(u, v) \cdot LA_k(u, v) + (1 - GR_k(u, v)) \cdot LB_k(u, v) \]
   - Expand and sum levels of $LS$ to obtain output image $S$

---

Pyramid Processing

- Hierarchical Structures for Image Representation
- Scene Structure occurs at many Sizes
- Match to Resolution and Scale

**Ex** $N \times N$ image; $M \times M$ template

Correlation requires $M^2N^2$ ops

To detect at finer scale:
(a) Increase scale of template by $S$: $S^2M^2N^2$ ops
(b) Decrease scale of image by $S$: $M^2N^2 / S^2$ ops

2 approaches differ in cost by $S^4$
- Fast Feature Detection

  * Hierarchical, Fine-to-Coarse Operators

  Gaussian Pyramid
  Feature Pyramids (Texture, Motion)
  Image Moment Pyramids

  * Complex Features over large neighborhoods defined in terms of simpler features in small neighborhoods

  * Hierarchical Grouping and Model-fitting
- Coarse-to-Fine Hierarchical Search
  
  * Selectively process only relevant areas and scales
  
  * Iterative refinement
  
  * Variable resolution analysis

Computing change energy within selected frequency
Oriented ("Steerable") Pyramids

1. Laplacian pyramid is orientation-independent
2. Apply an oriented filter to determine orientations at each layer
   - by clever filter design, we can simplify synthesis
   - this represents image information at a particular scale and orientation
From "Shiftable MultiScale Transforms," by Simoncelli et al., IEEE Transactions on Information Theory, 1992
Analysis

Laplacian Pyramid Layer

B_1

B_2

B_3

B_4

Oriented Pyramid Levels

Synthesis

Oriented Pyramid Levels

B_3

B_6

B_5

B_3

B_6

Laplacian Pyramid Layer
Edge Detection by Function Fitting: Facet Model

1. **General approach**
   - Fit a function to each pixel’s neighborhood of the image
   - Use the gradient of the function as the digital gradient of the image neighborhood

1. **Example**: Fit a plane to a 2 x 2 neighborhood
   - $z = ax + by + c$; $z$ is gray level
   - Gradient is then $(a^2 + b^2)^{1/2}$
   - Neighborhood points are $I(x,y)$, $I(x+1,y)$, $I(x,y+1)$ and $I(x+1,y+1)$
   - Need to minimize $E(a,b,c) = \sum \sum [a(x+i) + b(y+j) + c - I(x+i,y+j)]^2$
   - Solve this and similar problems by:
     - Differentiating with respect to $a,b,c$, set results to 0, and
     - Solve for $a,b,c$ in resulting system of equations

\[
\frac{\partial E}{\partial a} = \sum \sum 2[a(x+i) + b(y+j) + c - I(x+i,y+j)](x+i)
\]
\[
\frac{\partial E}{\partial b} = \sum \sum 2[a(x+i) + b(y+j) + c - I(x+i,y+j)](y+j)
\]
\[
\frac{\partial E}{\partial c} = \sum \sum 2[a(x+i) + b(y+j) + c - I(x+i,y+j)]
\]
1. It is easy to verify that
   - $a = \frac{[I(x+1,y) + I(x+1,y+1) - I(x,y) - I(x,y+1)]}{2}$
   - $b = \frac{[I(x,y+1) + I(x+1,y+1) - I(x,y) - I(x+1,y)]}{2}$
1. $a$ and $b$ are the partial derivatives with respect to $x$ and $y$

\[
a = \begin{array}{cc}
-1 & 1 \\
-1 & 1
\end{array}
\]
\[
b = \begin{array}{cc}
1 & 1 \\
-1 & -1
\end{array}
\]
Edge Detection by Function Fitting

1. Could also fit a higher order surface than a plane
   - With a second order surface we could find the (linear) combination of pixel values that corresponds to the higher order derivatives, which can also be used for edge detection
2. Would ordinarily use a neighborhood larger than 2 x 2
   - Better fit
   - For high degree functions need more points for the fit to be reliable

Edge Linking and Following

1. Group edge pixels into chains and chains into large pieces of object boundary.
   - can use the shapes of long edge chains in recognition
     1. slopes
     1. curvature
     1. corners
2. Basic steps
   - thin connected components of edges to one pixel thick
   - find simply connected paths
   - link them at corners into a graph model of image contours
   - compute local and global properties of contours and corners
Finding Simply Connected Chains

1. **Goal:** create a graph-structured representation (chain graph) of the image contours
   - vertex for each junction in the image
   - edge connecting vertices corresponding to junctions that are connected by a chain; edge labeled with chain

Creating the Chain Graph

1. **Algorithm:** given binary image, E, of thinned edges
   - create a binary image, J, of junctions and end points
     - points in E that are 1 and have more than two neighbors that are 1 or exactly one neighbor that is a 1
   - create the image \( E-J = C(\text{chains}) \)
     - this image contains the chains of E, but they are broken at junctions
   - perform a connected component analysis of C. For each component store in a table T:
     - its end points (0 or 2)
     - the list of coordinates joining its end points
   - For each point in J:
     - create a node in the chain graph, G, with a unique label
Creating the Chain Graph

1. For each chain in C
   - if that chain is a closed loop (has no end points)
     1. choose one point from the chain randomly and create a new node in G corresponding to that point
     1. mark that point as a “loop junction” to distinguish it from other junctions
     1. create an edge in G connecting this new node to itself, and mark that edge with the name of the chain loop
   - if that chain is not a closed loop, then it has two end points
     1. create an edge in G linking the two points from J adjacent to its end points

Creating the Chain Graph

1. Data structure for creating the chain graph
1. Biggest problem is determining for each open chain in C the points in J that are adjacent to its end points
   - sophisticated solution might use a hierarchical data structure, like a k-d tree, to represent the points in J
   - more direct solution is to create image J in which all 1’s are marked with their unique labels
   - For each chain in C
     1. Examine the 3 x 3 neighborhood of each end point of C in J
     1. Find the name of the junction or end point adjacent to that end point from this 3 x 3 neighborhood
Finding Internal “Corners” of Chains

1. Chains are only broken at junctions
   - But important features of the chain might occur at internal points
   - Example: closed loop corresponding to a square - would like to find the natural corners of the square and add them as junctions to the chain graph (splitting the chains at those natural corners)

2. Curve segmentation
   - Similar to image segmentation, but in a 1D form
     - Local methods, like edge detectors
     - Global methods, like region analyzers

Local Methods of Curve Segmentation

1. Natural locations to segment contours are points where the slope of the curve is changing quickly
   - These correspond, perceptually, to “corners” of the curve

1. To measure the change in slope we are measuring the **curvature** of the curve
   - Straight line has 0 curvature
   - Circular arc has constant curvature corresponding to 1/r

1. Can estimate curvature by fitting a simple function (circular arc, quadratic function, cubic function) to each neighborhood of a chain, and using the parameters of the fit to estimate the curvature at the center of the neighborhood
Formulas for Curvature

1. Consider moving a point, P, along a curve
   - Let T be the unit tangent vector as P moves
     1. T has constant length (1)
     1. The direction of T, φ, changes from point to point unless the curve is a straight line
     1. Measure this direction as the angle between T and the x-axis

   \[ T = \frac{dR}{ds}, \text{s distance along curve} \]

\[ \phi = \tan^{-1} \left( \frac{dy}{dx} \right) \]

Formulas for Curvature

1. The \textbf{curvature}, \( \kappa \), is the instantaneous rate of change of \( \phi \) with respect to s, distance along the curve
   - \( \kappa = \frac{d\phi}{ds} \)
   - \( ds = (dx^2 + dy^2)^{1/2} \)
   - \( \phi = \tan^{-1} dy/dx \)

\[ T = \frac{dR}{ds}, \text{s distance along curve} \]
Formulas for Curvature

1. Now

\[ \frac{d\phi}{dx} = \frac{\frac{d^2y}{dx^2}}{1 + \left(\frac{dy}{dx}\right)^2} \]

and

\[ \frac{ds}{dx} = \sqrt{1 + \left(\frac{dy}{dx}\right)^2} \]

so

\[ \kappa = \frac{d\phi}{ds} = \frac{\frac{d\phi}{dx}}{\frac{ds}{dx}} = \frac{\frac{d^2y}{dx^2}}{\left[1 + \left(\frac{dy}{dx}\right)^2\right]^{3/2}} \]

Example - Circle

1. For the circle

- \( s = a\theta \)
- \( \phi = \theta + \pi/2 \)
- so \( \kappa = \frac{d\phi}{ds} = \frac{d\theta}{ad\theta} = \frac{1}{a} \)
Local Methods of Curve Segmentation

1. There are also a wide variety of heuristic methods to estimate curvature like local properties
   - For each point, \( p \), along the curve
   - Find the points \( k \) pixels before and after \( p \) on the curve \( (p^k, p^k) \)
     and then measure
       1. the angle between \( pp^k \) and \( pp^k \)
       2. the ratio \( s/t \)

1. Similar problems to edge detection
   - what is the appropriate size for \( k \)?
   - how do we combine the curvature estimates at different scales?
   - boundary problems near the ends of open curves - not enough pixels to look out \( k \) in both directions