Filtering and Edge Detection

Local Neighborhoods

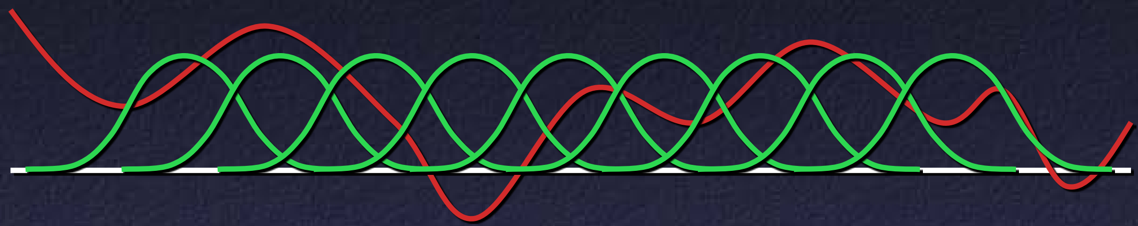
- Hard to tell anything from a single pixel
  - Example: you see a reddish pixel. Is this the object's color? Illumination? Noise?
- The next step in order of complexity is to look at local neighborhood of a pixel

Linear Filters

- Given an image *In*(x,*y*) generate a new image *Out*(x,*y*):
  - For each pixel (x,*y*), *Out*(x,*y*) is a linear combination of pixels in the neighborhood of *In*(x,*y*)
- This algorithm is
  - Linear in input intensity
  - Shift invariant

Discrete Convolution

- This is the discrete analogue of convolution



$$f(x) * g(x) = \int_{t=-\infty}^{\infty} f(t)\, g(x-t)\, dt$$

- Pattern of weights = "filter kernel"
- Will be useful in smoothing, edge detection

Example: Smoothing

Original: Mandrill

Smoothed with Gaussian kernel

Gaussian Filters

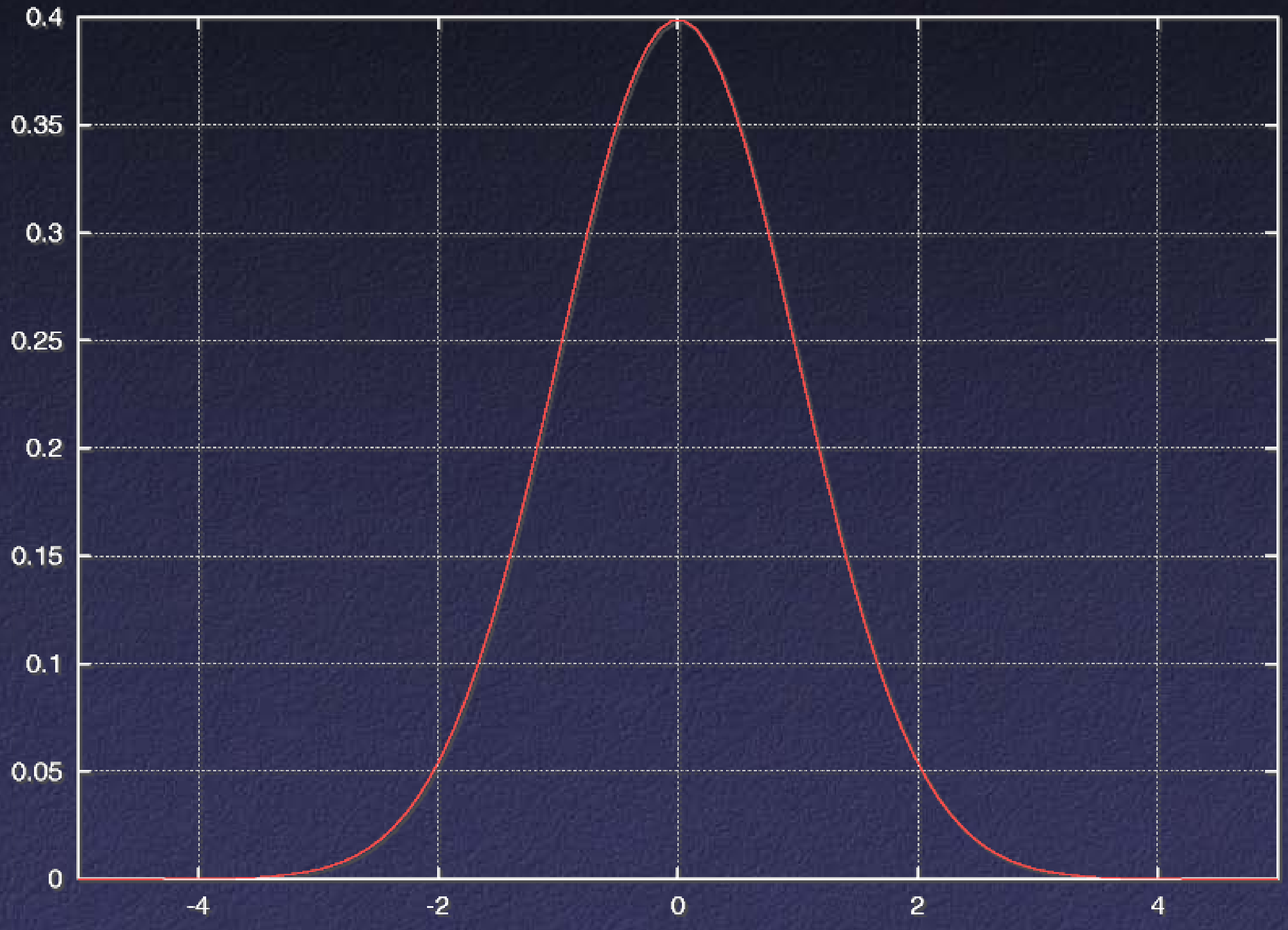
- One-dimensional Gaussian

$$G_1(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

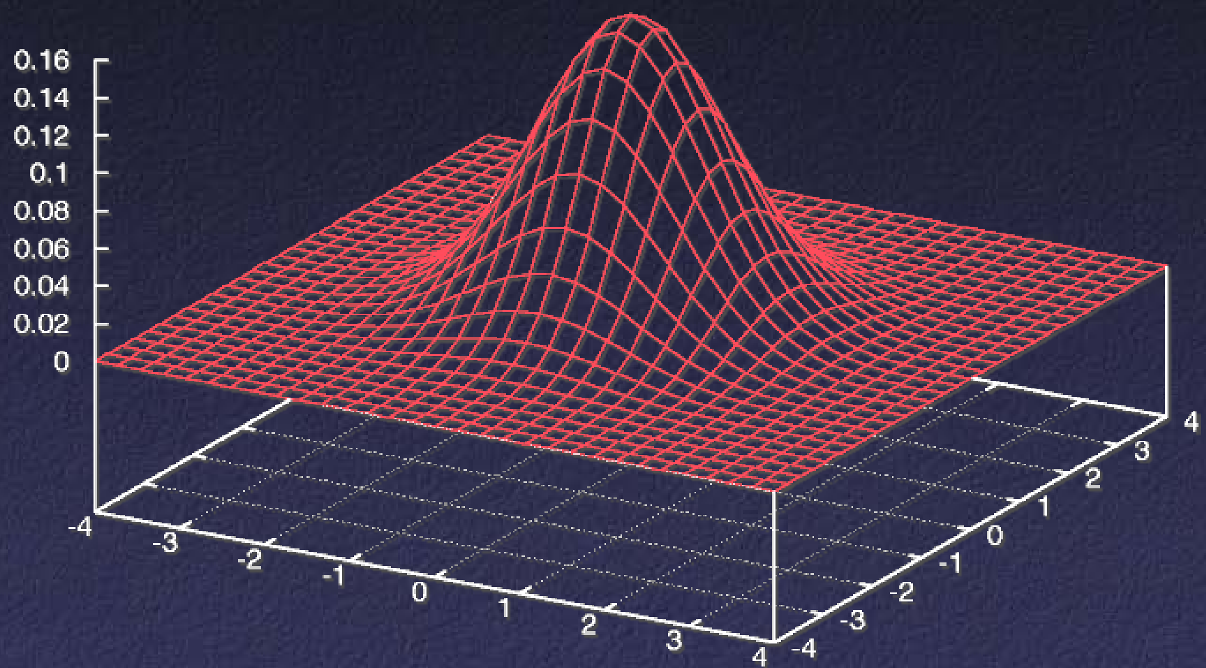
- Two-dimensional Gaussian

$$G_2(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Gaussian Filters



Gaussian Filters



Gaussian Filters

- Gaussians are used because:
  - Smooth
  - Decay to zero rapidly
  - Simple analytic formula
  - Central limit theorem: limit of applying (most) filters multiple times is Gaussian
  - Separable:

$$G_2(x,y) = G_1(x)\ G_1(y)$$

Computing Discrete Convolutions

$$Out(x,y) = \sum_i \sum_j f(i,j) \cdot In(x-i, y-j)$$

- What happens near edges of image?
  - Ignore (*Out* is smaller than *In*)
  - Pad with zeros (edges get dark)
  - Replicate edge pixels
  - Wrap around
  - Reflect
  - Change filter

Computing Discrete Convolutions

$$Out(x,y) = \sum_i \sum_j f(i,j) \cdot In(x-i, y-j)$$

- If *In* is $n \times n$, f is $m \times m$, takes time $O(m^2 n^2)$
- OK for small filter kernels, bad for large ones

Fourier Transforms

- Define *Fourier transform* of function *f* as

$$\mathrm{F}(\omega) = \mathcal{F}\big(f(x)\big) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) e^{i\omega x} dx$$

- F is a function of frequency – describes how much of each frequency *f* contains
- Fourier transform is invertible

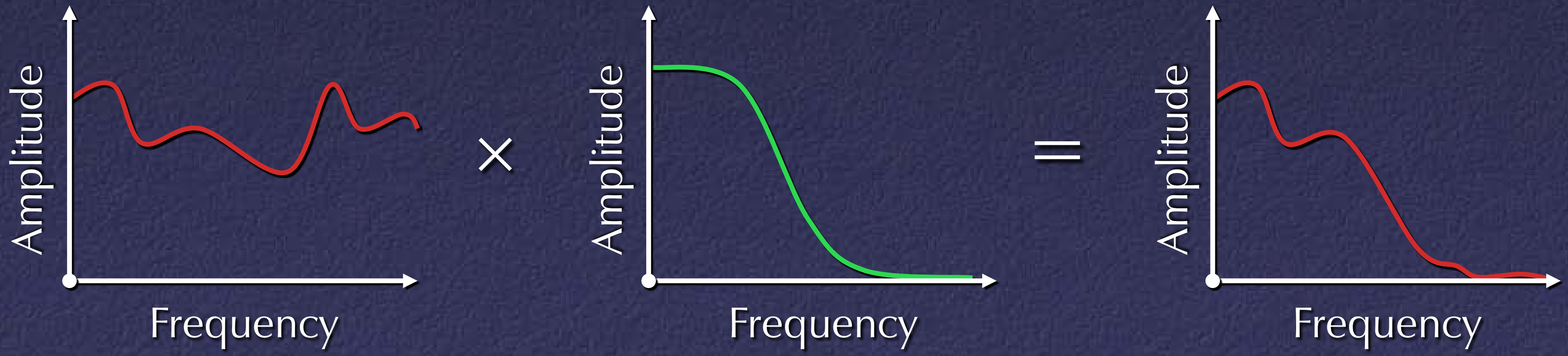
Fourier Transform and Convolution

- Fourier transform turns convolution into multiplication:

$$\mathcal{F}\big(f(x) * g(x)\big) = \mathcal{F}\big(f(x)\big)\,\mathcal{F}\big(g(x)\big)$$

Fourier Transform and Convolution

- Useful application #1: Use frequency space to understand effects of filters
  - Example: Fourier transform of a Gaussian is a Gaussian
  - Thus: attenuates high frequencies



Fourier Transform and Convolution

- Useful application #2: Efficient computation
  - Fast Fourier Transform (FFT) takes time

    $$O(n \log n)$$

  - Thus, convolution can be performed in time

    $$O(n \log n + m \log m)$$

  - Greatest efficiency gains for large filters

Edge Detection

- What do we mean by edge detection?
- What is an edge?

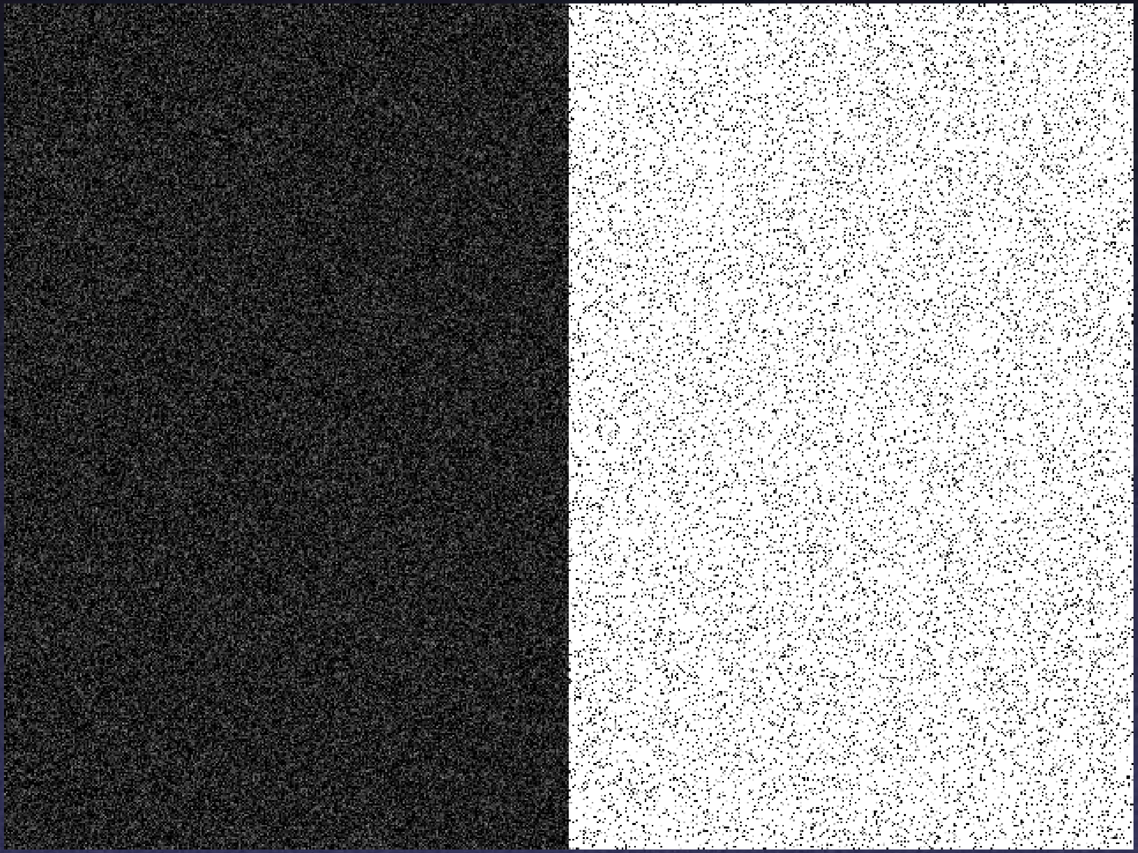
What is an Edge?

Edge easy to find

What is an Edge?

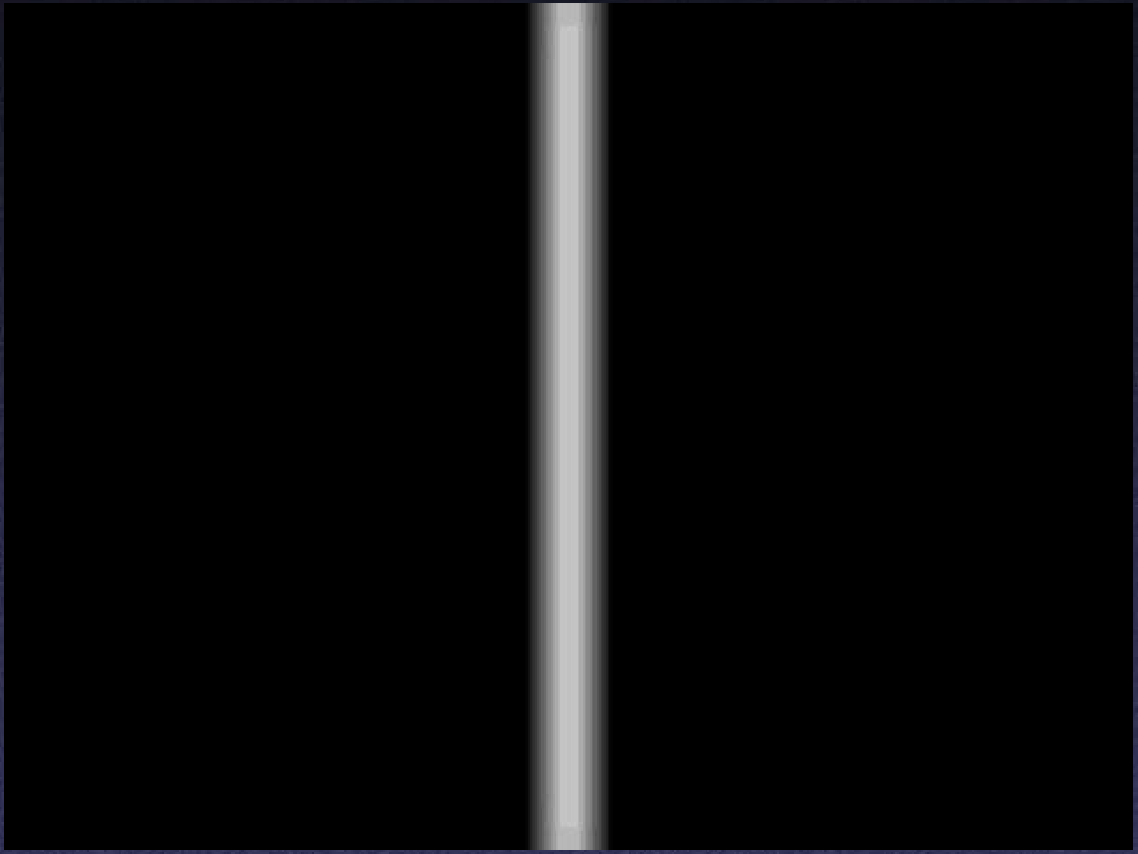
Where is edge? Single pixel wide or multiple pixels?

What is an Edge?



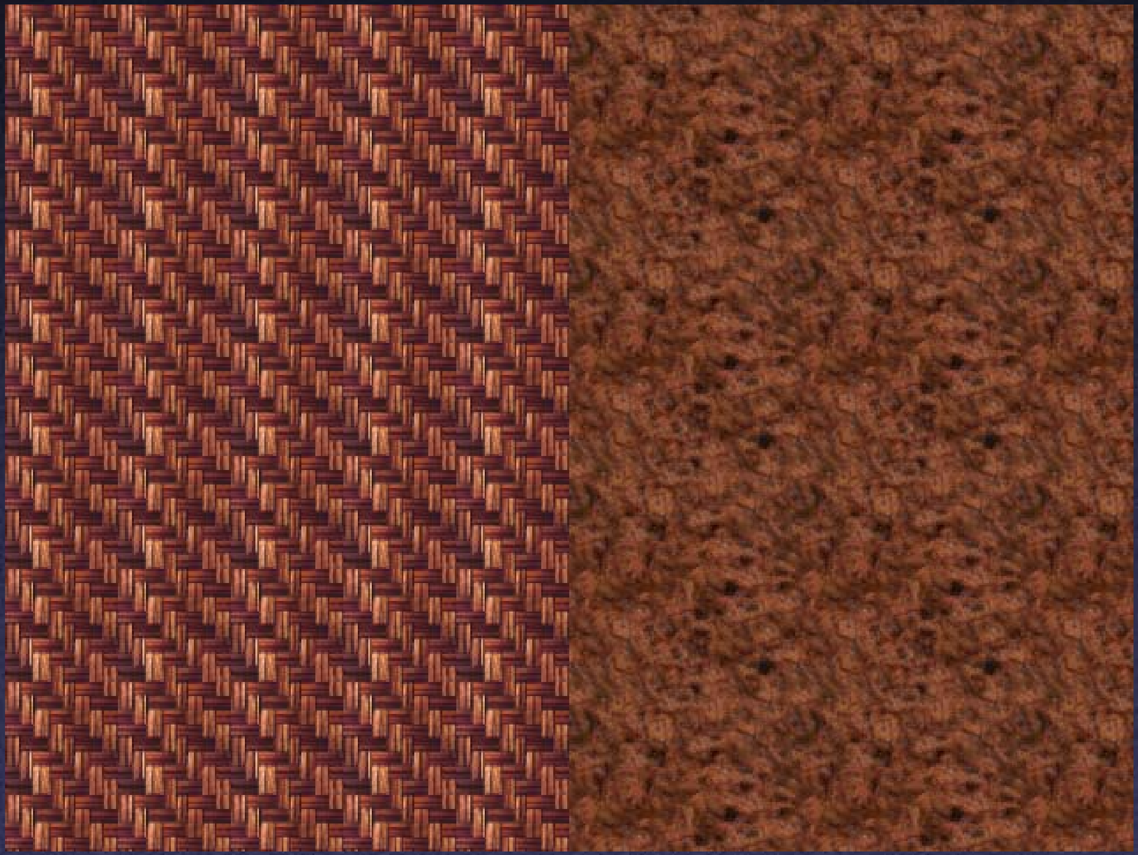
Noise: have to distinguish noise from actual edge

What is an Edge?



Is this one edge or two?

What is an Edge?



Texture discontinuity

Formalizing Edge Detection

- Look for strong step edges

$$\frac{dI}{dx} > \tau$$

- One pixel wide: look for *maxima* in dI / dx
- Noise rejection: smooth (with a Gaussian) over a neighborhood of size σ

Canny Edge Detector

- Smooth
- Find derivative
- Find maxima
- Threshold

Canny Edge Detector

- First, smooth with a Gaussian of some width σ

Canny Edge Detector

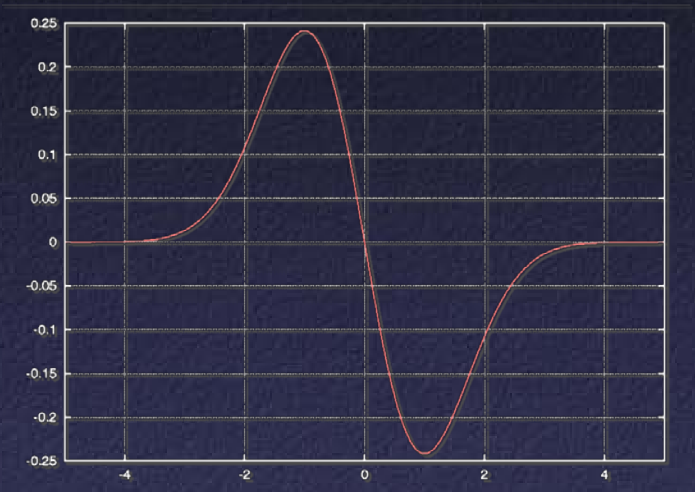
- Next, find "derivative"
- What is derivative in 2D? Gradient:

$$\nabla f(x, y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)$$

Canny Edge Detector

- Useful fact #1: differentiation "commutes" with convolution

$$\frac{d}{dx}\left(f * g\right) = \frac{df}{dx} * g$$

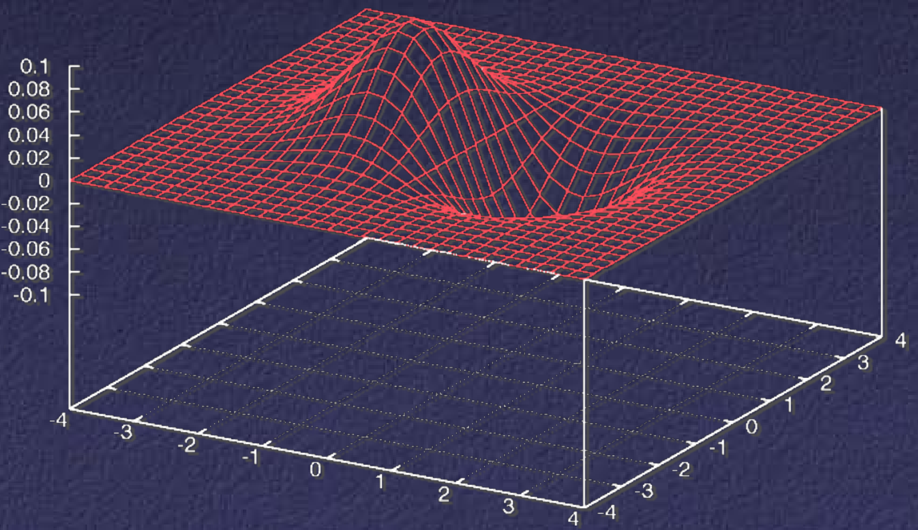


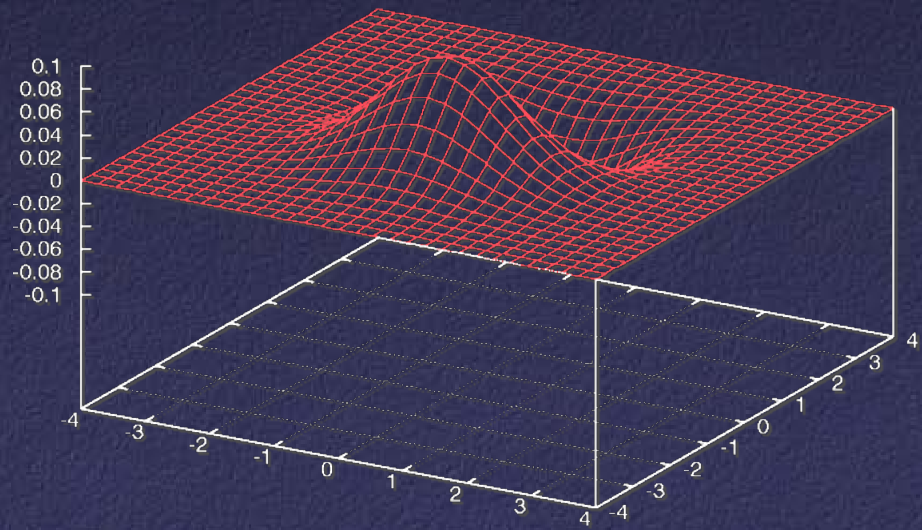
- Useful fact #2: Gaussian is separable: $G_2(x, y) = G_1(x)G_1(y)$

Canny Edge Detector

- Thus, combine first two stages of Canny:

$$\nabla\left(f(x,y) * G_2(x,y)\right) = \begin{bmatrix} f(x,y) * \left(G_1'(x)G_1(y)\right) \\ f(x,y) * \left(G_1(x)G_1'(y)\right) \end{bmatrix} = \begin{bmatrix} f(x,y) * G_1'(x) * G_1(y) \\ f(x,y) * G_1(x) * G_1'(y) \end{bmatrix}$$





Canny Edge Detector

Original Image

Smoothed Gradient Magnitude

Canny Edge Detector

- Nonmaximum suppression
  - Eliminate all but local maxima in *magnitude* of gradient
  - At each pixel look along *direction* of gradient: if either neighbor is bigger, set to zero
  - In practice, quantize direction to horizontal, vertical, and two diagonals
  - Result: “thinned edge image”

Canny Edge Detector

- Final stage: thresholding
- Simplest: use a single threshold
- Better: use two thresholds
  - Find chains of edge pixels, all greater than τ_{low}
  - Each chain must contain at least one pixel greater than τ_{high}
  - Helps eliminate dropouts in chains, without being too susceptible to noise
  - "Thresholding with hysteresis"

Canny Edge Detector

Original Image

Edges

Other Edge Detectors

- Can build simpler, faster edge detector by omitting some steps:
  - No nonmaximum suppression
  - No hysteresis in thresholding
  - Simpler filters (approx. to gradient of Gaussian)
    - Sobel: $\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$ $\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$
    - Roberts: $\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$ $\begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}$

Second-Derivative-Based Edge Detectors

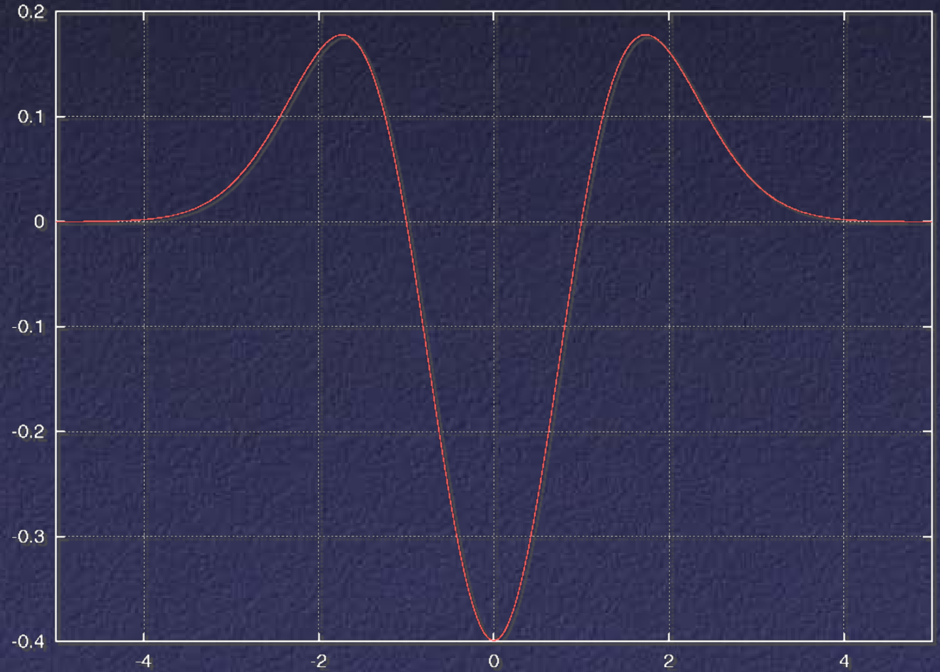
- To find local maxima in derivative, look for zeros in second derivative
- Analogue in 2D: Laplacian

$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

- Marr-Hildreth edge detector

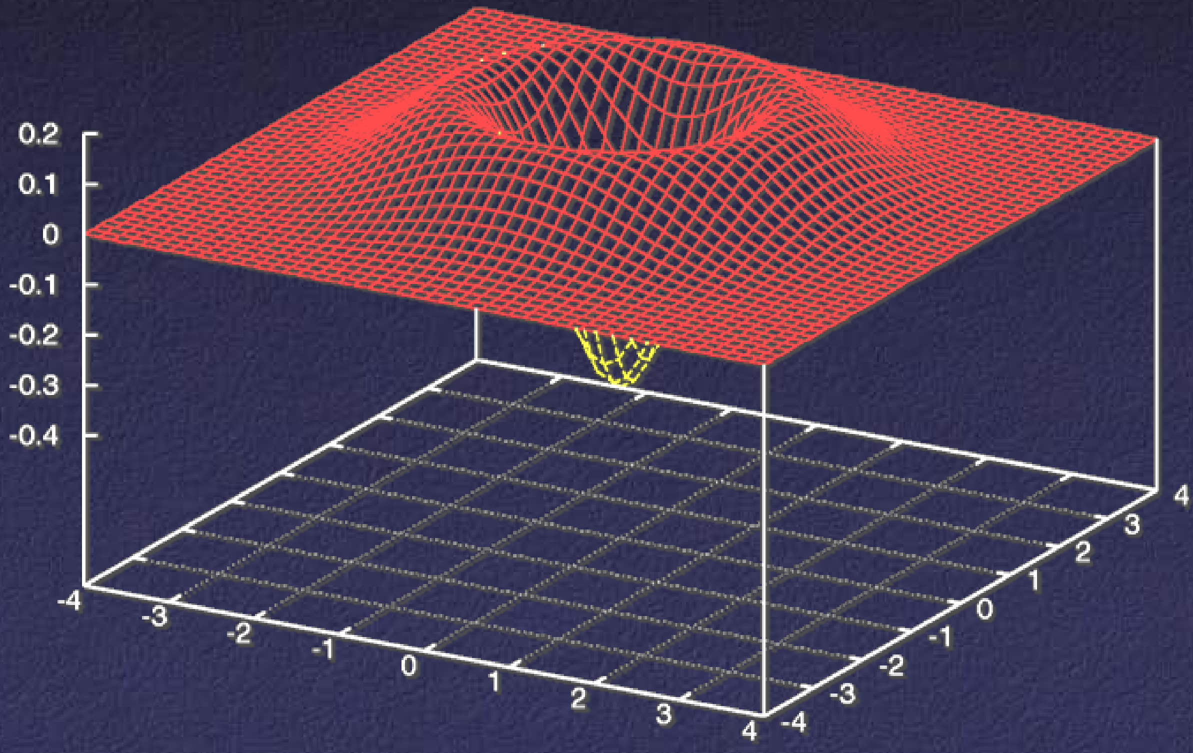
LOG

- As before, combine Laplacian with Gaussian smoothing: Laplacian of Gaussian (LOG)



LOG

- As before, combine Laplacian with Gaussian smoothing: Laplacian of Gaussian (LOG)



Problems with Laplacian Edge Detectors

- Local minimum vs. local maximum
- Symmetric – poor performance near corners
- Sensitive to noise
  - Higher-order derivatives = greater noise sensitivity
  - Combines information along edge, not just perpendicular