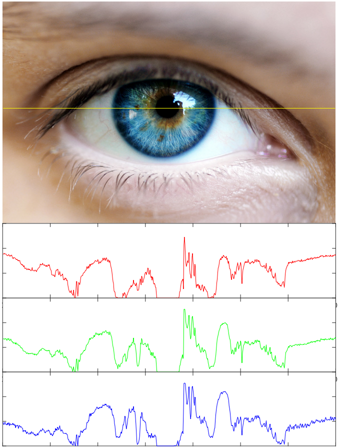
Image Differentiation and Image Pyramids

COMPSCI 527 — Computer Vision

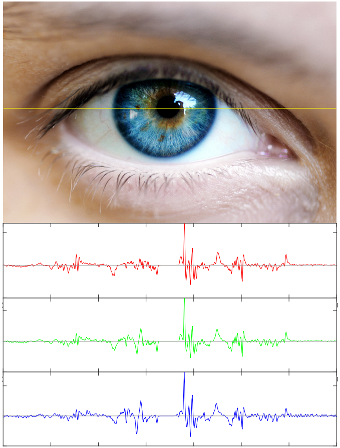
Outline

What Does Differentiating an Image Mean?

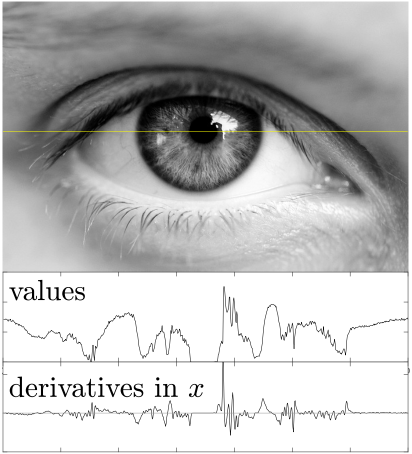
Values

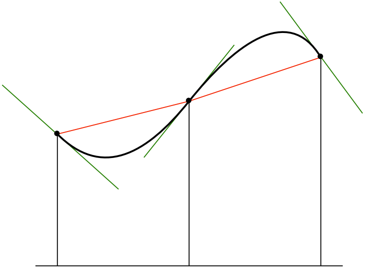


Derivatives in x



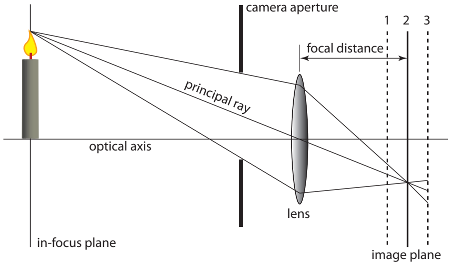
What Does Differentiating an Image Mean?



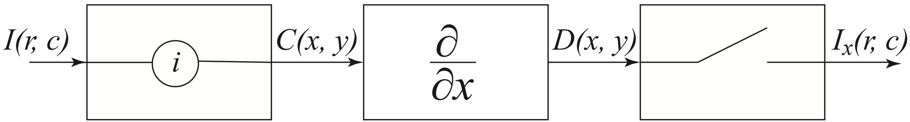


Can we reconstruct the black curve?

Cameras

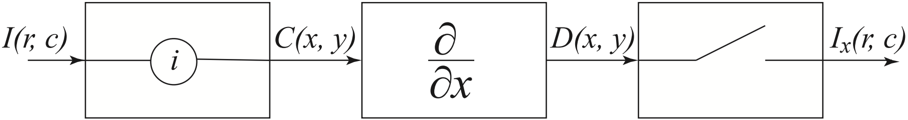

camera aperture
1
2
3
focal distance
principal ray
optical axis
lens
in-focus plane
image plane

A Conceptual Pipeline



- Somehow reconstruct the continuous sensor irradiance C from the discrete image array I
- Differentiate C to obtain D
- Sample the derivatives D back to the pixel grid
- Each would be hard to implement
- Surprisingly, *the cascade turns out to be easy!*

From Discrete Array to Sensor Irradiance



What would the transformation from I to C look like formally, if we could find one? Example: Linear interpolation

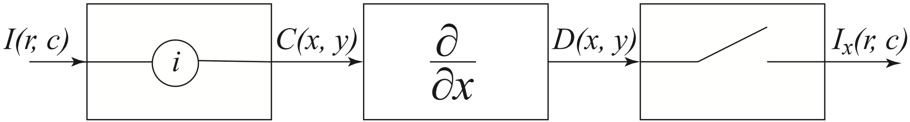
Linear Interpolation as a Hybrid Convolution

$$C(x, y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(i, j) P(x - j, y - i)$$

Gaussian Instead of Triangle

- Noise $\Rightarrow$: fit rather than interpolating
- Noise $\Rightarrow$: filter with a Gaussian
- $P(x, y) = G(x, y) \propto e^{-\frac{1}{2}\frac{x^2+y^2}{\sigma^2}}$

Differentiating

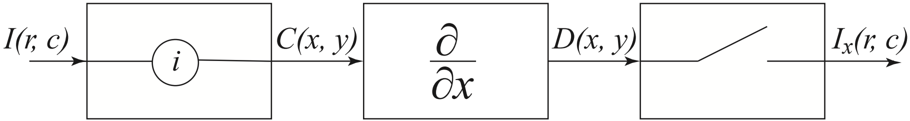


$C(x,y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(i,j)G(x-j, y-i)$
(still don't know how to do this, just plow ahead)
$D(x,y) = \frac{\partial C}{\partial x}(x,y) = \frac{\partial}{\partial x} \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(i,j)G(x-j, y-i)$
$D(x,y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(i,j)G_x(x-j, y-i)$

- We transferred the differentiation to G,
  and we know how to do *that*!

  (still don't know how to implement a hybrid convolution)

Sampling



$$D(x,y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(i,j) G_x(x-j, y-i)$$

- We are interested in the values of $D(x,y)$ *on the integer grid*: $x \to c$ and $y \to r$

$$I_x(r,c) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(i,j) G_x(c-j, r-i)$$

Wait! This is a standard, discrete convolution

We know how to do *that*!

To differentiate an image array, convolve it (discretely) with the (sampled, truncated) derivative of a Gaussian

The Derivatives of a 2D Gaussian

- The Gaussian function is separable:

  $G(x,y) \propto e^{-\frac{1}{2}\frac{x^2+y^2}{\sigma^2}} = g(x)\,g(y)$ where

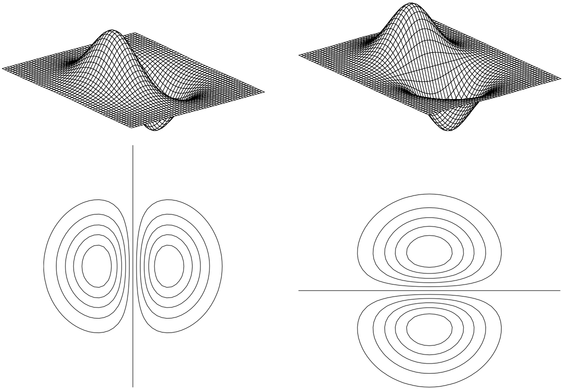
  $g(x) = e^{-\frac{1}{2}\frac{x^2}{\sigma^2}}$

  $G_x(x,y) = \frac{\partial G}{\partial x} = \frac{\partial g}{\partial x}\,g(y) = d(x)g(y)$

  $d(x) = \frac{dg}{dx} = -\frac{x}{\sigma^2}g(x)$
- Similarly, $G_y(x,y) = g(x)d(y)$
- Differentiate (smoothly) in one direction, smooth in the other
- $G_x(x,y)$ and $G_y(x,y)$ are separable as well

The Derivatives of a 2D Gaussian

$$G_x(x, y) = d(x)g(y) \text{ and } G_y(x, y) = g(x)d(y)$$

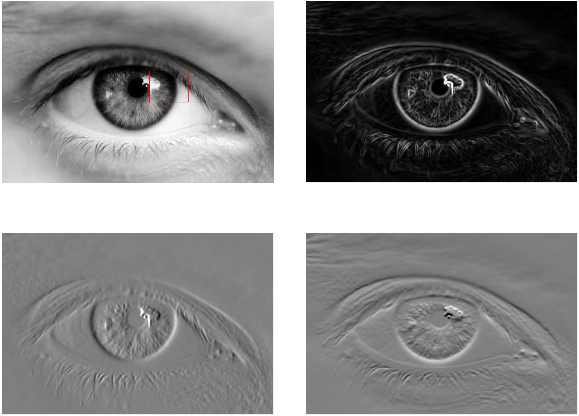


Normalization

- Can normalize $d(c)$ and $g(r)$ separately
- For smoothing, constants should not change:
- We want $k * g = k$ (we saw this before)
- For differentiation, a unit ramp should not change:
  $u(r, c) = c$ is a ramp
- We want $u * d = 1$ (see notes for math)

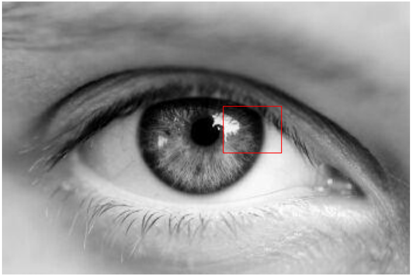
The Image Gradient

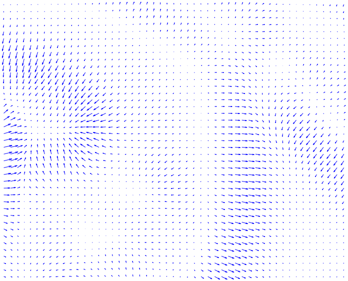
- Image *gradient*: $\nabla I(r,c) = \frac{\partial I}{\partial \mathbf{x}} = \mathbf{g}(r,c) = \left[\begin{array}{c} I_x(r,c) \\ I_y(r,c) \end{array} \right]$
- View 1: Two scalar images $I_x(r,c)$, $I_y(r,c)$



The Image Gradient

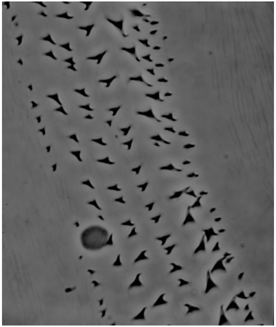
- View 2: One vector image $\mathbf{g}(r, c)$





- We can now measure changes of image brightness
- *Edges* are of particular interest

Image Pyramids and Scale



↑ smallest denticle we look for

- Scale:
  - Start with smallest template
  - Look for larger and larger occurrences
- Larger template ≈ smaller image!

Scale Budgets

- $n \times n$ image, $k \times k$ template, scaling $s > 1$
- Processing a large image with progressively larger templates with scales $s, s^2, s^3, \ldots$:
  $n^2(k^2 + k^2s^2 + k^2s^4 + \ldots) = n^2k^2(1 + s^2 + s^4 + \ldots)$
- Series diverges
- Processing progressively smaller images with a small template:
  $k^2(n^2 + n^2/s^2 + n^2/s^4 + \ldots) = k^2n^2(1 + 1/s^2 + 1/s^4 + \ldots)$
- Series converges to $k^2n^2s^2/(s^2 - 1)$
- For $s = 2$, the series converges to $k^2n^2 4/3$
- About 33% additional cost relative to processing the original image alone

Finer Scales

- Scaling down by $s = 2$ every time may be overly aggressive
- Let $\phi = 1/s$ be the *scaling factor*
- For $0 < \phi < 1$, image shrinks. For $\phi > 1$, the image grows larger
- How to downsample ($0 < \phi < 1$)?
- Two issues: aliasing and non-integer s

Aliasing

- Even when s is an integer, pure sampling is a bad idea: *(Spatial frequency) aliasing*
- Colors are sampled at locations on the pixel grid
- Nothing to do with the scene

Original

Sampled by $s = 30$, then magnified by 30

Downsampling = Smoothing + Sampling

- Smooth with a Gaussian blur kernel first, then sample

Original

Smoothed with $\sigma = 48$,

then sampled by $s = 30$, then magnified by 30

- We lose detail (blur), but that's the whole point
- True scale: 
- Every pixel in the low-resolution image is a weighted average of pixel values in the original image

Key Questions

- How much to smooth before resampling?
  - That is, where does $\sigma = 48$ come from for $\phi = 1/30$?
  - Lots of theory for the optimal multiplier
  - Depends on various factors (spectral properties of image and noise)
  - We use what works most of the time, empirically
  - Answer: $\sigma \approx 1.6\,s = 1.6/\phi$
- How to “take one out of every s pixels” when $s = 1/\phi$ is not an integer?

Bilinear Interpolation

- What does it mean to "take one out of every s pixels" when $s = 1/phi$ is not an integer?

$$\xi = \lfloor x \rfloor \quad , \quad \eta = \lfloor y \rfloor$$
$$\Delta x = x - \xi \quad , \quad \Delta y = y - \eta$$

$$\begin{aligned} I(\mathbf{x}) &= I(\xi, \eta)\,(1 - \Delta x)\,(1 - \Delta y) \\ &+ I(\xi + 1, \eta)\,\Delta x\,(1 - \Delta y) \\ &+ I(\xi, \eta + 1)\,(1 - \Delta x)\,\Delta y \\ &+ I(\xi + 1, \eta + 1)\,\Delta x\,\Delta y \end{aligned}$$

Abstracting Pyramid Operations

$J = \texttt{resize}(I, \phi)$:

- If $0 < \phi < 1$, image shrinks:

  Filter with $\sigma = 1.6/\phi$,
  then sample every $s = 1/\phi > 1$ pixels

- If $\phi \geq 1$, image grows:

  No filter. Just sample every $s = 1/\phi \leq 1$ pixels

- Pyramid operators: Pick a *single* value of $\phi \in (0, 1)$, then define

$$\texttt{down}(X) = \texttt{resize}(X, \phi)$$
$$\texttt{up}(X) = \texttt{resize}(X, 1/\phi)$$

- `up` is *not* the inverse of `down`:
  Cannot restore lost information

A Gaussian Pyramid ($\phi = 1/2$)