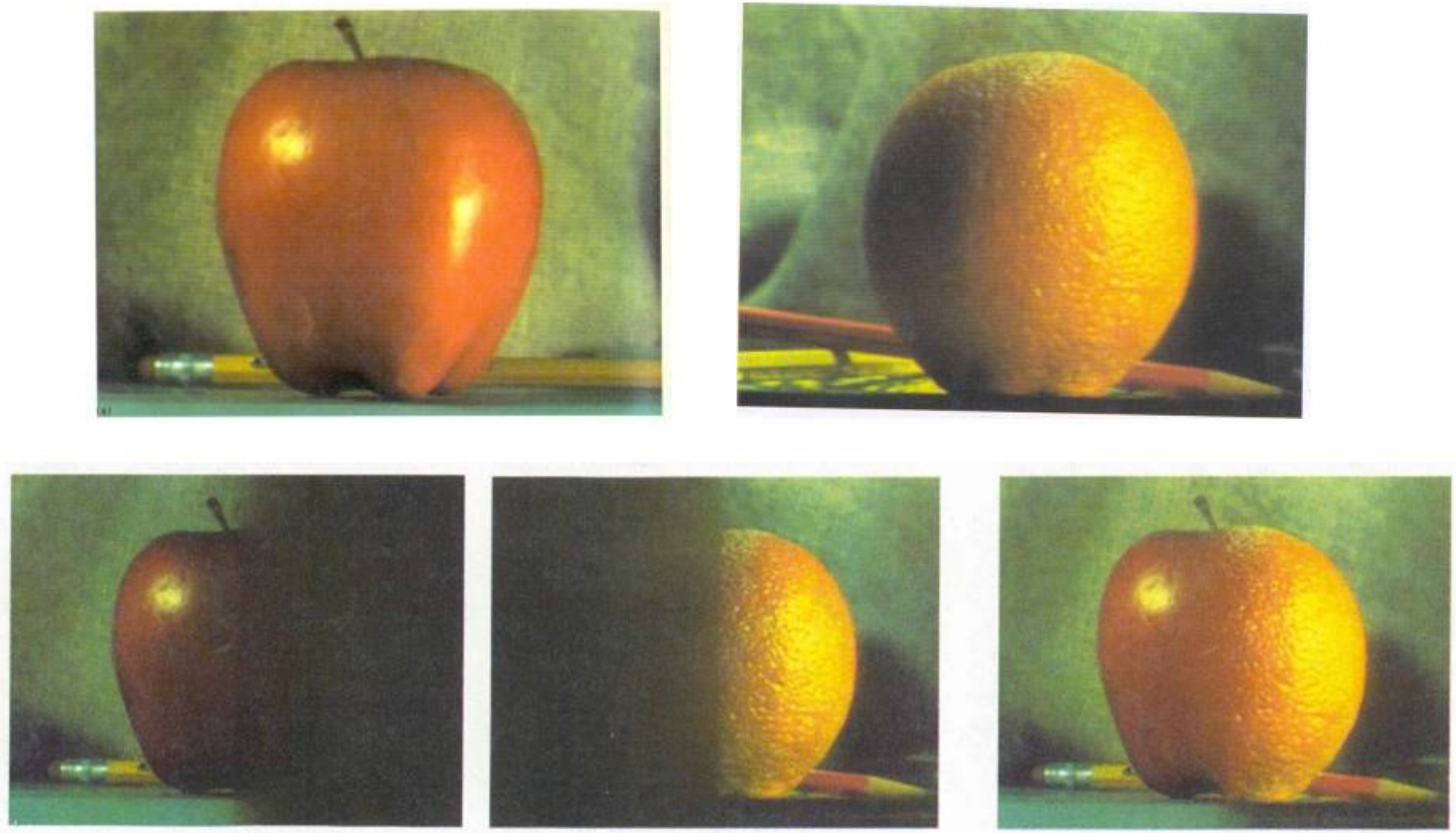
Gaussian & Laplacian Pyramid - A 1D Example Blending Algorithm

Slides by Yawen Ma

Based on Sam Hasinoff’s tutorial notes

Pyramid Blending



(d) (h) (l)

The 5-tap 1D

- Symmetric
  - $\hat{w} = [\mathrm{a\ b\ c\ b\ a}]$
- Same weight for sum odd and sum even location
  - $a + c + a = b + b = 0.5$
  - So $\hat{w} = [0.25 - \frac{\mathrm{c}}{2}\ , 0.25, \mathrm{c}, 0.25, 0.25 - \frac{\mathrm{c}}{2}]$

In this example we use c=0.4

So $\hat{w} = [0.05, 0.25, 0.40, 0.25, 0.05] = \frac{1}{20}[1\ 5\ 8\ 5\ 1]$

Consider a 9=2^3+1 pixel example image

G_{k-1} = [6 8 1 5 1 9 5 7 9]

- let's find G_k = REDUCE(G_{k-1})
  - It will have ~half as many pixels, more specifically 5=2^2+1 pixels
  - we only need to compute the odd pixels ..... since we'll throw the rest away .. "x" below means shouldn't/cannot compute

G_{k-1} = [6 8 1 5 1 9 5 7 9]

G_k = [? x ? x ? x ? x ?]

(this is just convolution with $\widehat{w}$ but we only need to compute this convolution for the odd pixels)

Consider a 9=2^3+1 pixel example image

G_{k-1} = [6 8 1 5 1 9 5 7 9]

- let's find G_k = REDUCE(G_{k-1})

G_{k-1} = [6 8 1 5 1 9 5 7 9]

G_k = [? x ? x ? x ? x ?]

eg. $G_k(3) = [1\ 5\ 1\ 9\ 5] * \frac{1}{20}[1\ 5\ 8\ 5\ 1]$

$= \frac{1}{20}(1 + 25 + 8 + 45 + 5)$

= 84/20

= 4.2

$G_k(2)$?

Consider a 9=2^3+1 pixel example image

G_{k-1} = [6 8 1 5 1 9 5 7 9]

- let's find G_k = REDUCE(G_{k-1})

G_{k-1} = [6 8 1 5 1 9 5 7 9]

G_k = [? x ? x ? x ? x ?]

eg. $G_k(1)$ = [x x 6 8 1] * $\frac{1}{20}$[1 5 8 5 1]

= [6 8 1] * $\frac{1}{14}$ [8 5 1] [reweighted]

= $\frac{1}{14}$ (48 + 40 + 1)

= 89/14

= 6.4

$G_k(5)$?

Consider a 9=2^3+1 pixel example image G_{k-1} = [6 8 1 5 1 9 5 7 9]

- let's find G_k = REDUCE(G_{k-1})

G_{k-1} = [6 8 1 5 1 9 5 7 9]

G_k = [? x ? x ? x ? x ?]

after computing everything we get

G_k = [6.4, 4.0, 4.2, 6.5, 8.0]

EXPAND([6.4, 4.0, 4.2, 6.5, 8.0])

- this is up-sampling a smaller image, ~doubling its resolution ..
- conceptually the reverse of the REDUCE function
- it will have the same number of pixels as G_{k-1} , 9=3^2+1 pixels

- we line things up and apply the same kernel as before, in the opposite direction, being careful to ”reweight” the kernel so it sums to 1, given that ~half the pixels from the source G_k will be missing.

G_k = [6.4 x 4.0 x 4.2 x 6.5 x 8.0]

EXPAND(G_k) = [? ? ? ? ? ? ? ? ?]

EXPAND([6.4, 4.0, 4.2, 6.5, 8.0])

G_k = [6.4 x 4.0 x 4.2 x 6.5 x 8.0]

EXPAND(G_k) = [? ? ? ? ? ? ? ? ?]

eg. EXPAND(G_k)(5) = [4.0 x 4.2 x 6.5] * $\frac{1}{20}$ [1 5 8 5 1]

= [4.0 4.2 6.5] * $\frac{1}{10}$ [1 8 1] [reweighted]

= 4.4

eg. EXPAND(G_k)(4) = [x 4.0 x 4.2 x] * $\frac{1}{20}$[1 5 8 5 1]

= [4.0 4.2] * $\frac{1}{10}$[5 5] [reweighted]

= 4.1

EXPAND([6.4, 4.0, 4.2, 6.5, 8.0])

G_k = [6.4 x 4.0 x 4.2 x 6.5 x 8.0]

EXPAND(G_k) = [? ? ? ? ? ? ? ? ?]

eg. EXPAND(G_k)(1) = [x x 6.4 x 4.0] * $\frac{1}{20}$ [1 5 8 5 1]

= [6.4 4.0] * $\frac{1}{9}$[8 1] [reweighted]

= 6.1

EXPAND([6.4, 4.0, 4.2, 6.5, 8.0])

G_k = [6.4 x 4.0 x 4.2 x 6.5 x 8.0]

EXPAND(G_k) = [6.1, 5.2, 4.3, 4.1, 4.4, 5.4, 6.4, 7.3, 7.8]

We started with

G_{k-1} = [6 8 1 5 1 9 5 7 9]

2D Case

- the 2D kernel function is separable and symmetric,
  - ie. $w(m,n) = \widehat{w}(m) * \widehat{w}(n)$
- so we can convolve the 1D $\widehat{w}$ first across rows, and then across columns (or vice versa) ... this will give the same result as convolving with the 2D kernel w directly

e.g. REDUCE for a 9x9 image ....

- REDUCE across rows to get a 9x5 image
- then REDUCE across columns to get a 5x5 image

e.g. EXPAND for a 5x5 image ....

- EXPAND across rows to get a 5x9 image
- then EXPAND across columns to get a 9x9 image

Compute the pyramids

To compute the Gaussian pyramid:

- G_0 = I [original image]
- G_k = REDUCE(G_{k-1}) [successively blur/reduce resolution]
- G_N [stop when the image is 1x1, or earlier]

To compute the Laplacian pyramid:

- L_N = G_N [base case, use the smallest Gaussian pyramid image]
- L_k = G_k - EXPAND(G_{k+1}) [detail/difference between successive Gaussian pyramid levels]

Blending algorithm

Input: images A, B & (binary) mask M that specifies the blend (0=A,1=B)

(pad everything to make them powers of 2 (+ 1), (2^N+1)x(2^N+1))

- -> A & B‘s Laplacian pyramids AL_0 , AL_1 ,…, AL_N & BL_0 , BL_1 ,…, BL_N
- -> M‘s Gaussian pyramid MG_0 , MG_1 ,…, MG_N
- compute a Laplacian pyramid for the result, S, using linear interpolation, for every pyramid level k,

SL_k (r,c) = (1- MG_k (r,c))* AL_k (r,c) + MG_k(r,c)* BL_k (r,c)

(we‘re simply doing linear interpolation over every pixel, with a blend mask given at different levels of detail)

Reconstitute the full-resolution image for S, by computing

- SG_N = SL_N
- SG_k = SL_k + EXPAND(SG_{k+1})

S = SG_0