Advanced image processing

Advanced Graphics

Rafał Mantiuk
Computer Laboratory, University of Cambridge

Linear filtering (revision)

- Output pixel value is a weighted sum of neighboring pixels

$$g(i,j) = \sum_{k,l} f(i-k, j-l)\,h(k,l)$$

Input pixel value

Kernel (filter)

Resulting pixel value

Sum over neighboring pixels, e.g. k=-1,0,1, j=-1,0,1 for 3x3 neighborhood

compact notation $g = f * h$

Convolution operation

Linear filter: example

$f(x,y)$:

45	60	98	127	132	133	137	133
46	65	98	123	126	128	131	133
47	65	96	115	119	123	135	137
47	63	91	107	113	122	138	134
50	59	80	97	110	123	133	134
49	53	68	83	97	113	128	133
50	50	58	70	84	102	116	126
50	50	52	58	69	86	101	120

*

$h(x,y)$:

0.1	0.1	0.1
0.1	0.2	0.1
0.1	0.1	0.1

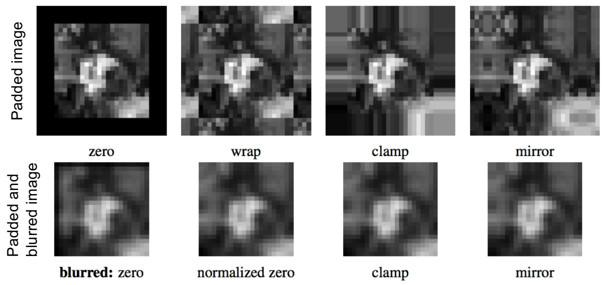
=

$g(x,y)$:

69	95	116	125	129	132
68	92	110	120	126	132
66	86	104	114	124	132
62	78	94	108	120	129
57	69	83	98	112	124
53	60	71	85	100	114

Why is the matrix g smaller than f ?

Padding an image



What is the computational cost of the convolution?

$$g(i,j) = \sum_{k,l} f(i-k, j-l)\,h(k,l)$$

- How many multiplications do we need to do to convolve 100x100 image with 9x9 kernel ?
  - The image is padded, but we do not compute the values for the padded pixels

Separable kernels

- Convolution operation can be made much faster if split into two separate steps:
  - 1) convolve all rows in the image with a 1D filter
  - 2) convolve columns in the result of 1) with another 1D filter
- But to do this, the kernel must be separable

$$\begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} \\ h_{2,1} & h_{2,2} & h_{2,3} \\ h_{3,1} & h_{3,2} & h_{3,3} \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \cdot \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix}$$

$$\vec{h} = \vec{u} \cdot \vec{v}$$

Examples of separable filters

- Box filter: $\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} = \begin{bmatrix} ? \\ ? \\ ? \end{bmatrix} \cdot \begin{bmatrix} ? & ? & ? \end{bmatrix}$
- Gaussian filter:

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

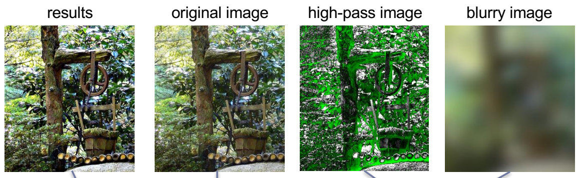
- What is $\frac{1}{2\pi\sigma^2}$ needed for?

$$G(x,y) = u(x) \cdot v(y)$$

- What are the corresponding 1D components of this separable filter (u(x) and v(y))?

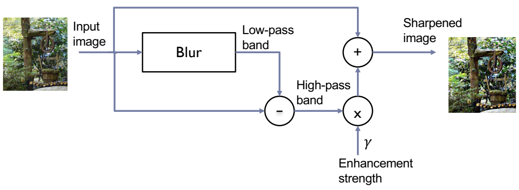
Unsharp masking

- How to use blurring to sharpen an image ?



$$g_{\text{sharp}} = f + \gamma(f - h_{\text{blur}} * f)$$

Unsharp masking as a block diagram



Why "linear" filters ?

- Linear functions have two properties:
  - Additivity: f(x) + f(y) = f(x+y)
  - Homogenity: f(a•x) = a•f(x) (where "f" is a linear function)
- Why is it important?
  - Linear operations can be performed in an arbitrary order
  - blur(a•F + b) = a•blur(F) + b
  - Linearity of the Gaussian filter could be used to improve the performance of your image processing operation
  - This is also how the separable filters work:

Matrix multiplication

Convolution

$$(u \cdot v) * f = u * (v * f)$$

The components of a separable kernel

An image

Edge stopping filters

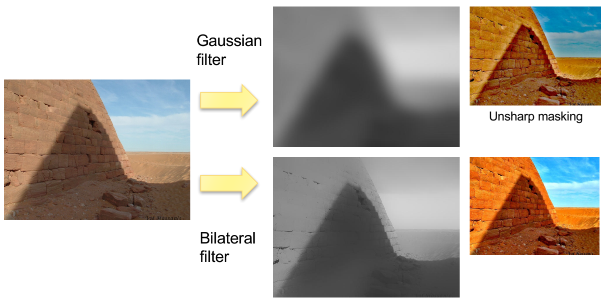
Original — Edge-aware smoothing — Detail enhancement

Stylization — Recoloring — Pencil drawing — Depth-of-field

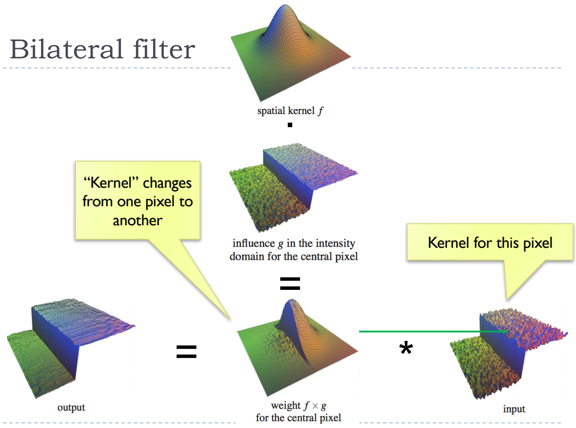
Examples from [Gastal & Oliveira 2011]

Nonlinear filters: Bilateral filter

- Goal: Smooth out the image without blurring edges



Bilateral filter



Bilateral filter

Input image

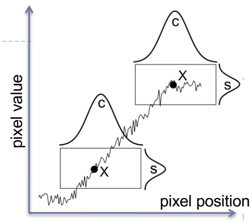
$$y(\boldsymbol{p}) = \frac{\sum_{\boldsymbol{q}\in\Omega} x(\boldsymbol{q})w(\boldsymbol{p},\boldsymbol{q})}{\sum_{\boldsymbol{q}\in\Omega} w(\boldsymbol{p},\boldsymbol{q})}$$

Pixel coordinates $\boldsymbol{p} = (i, j)$

Neighborhood of the pixel **p**

$$w(\boldsymbol{p},\boldsymbol{q}) = \underbrace{g_s(\boldsymbol{p}-\boldsymbol{q})}_{\text{distance in the spatial position (x,y)}}\underbrace{g_r(x(\boldsymbol{p})-x(\boldsymbol{q}))}_{\text{distance (difference) in pixel values}}$$

$$g_s(\boldsymbol{d}) = \exp\left(\frac{-\|\boldsymbol{d}\|_2}{2\sigma_s^2}\right) \qquad g_r(d) = \exp\left(\frac{-d^2}{2\sigma_s^2}\right)$$

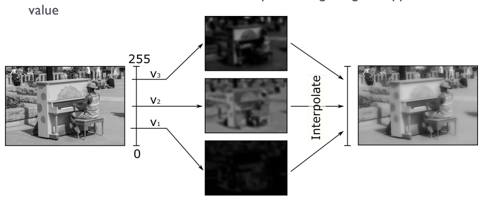


How to make the bilateral filter fast?

- A number of approximations have been proposed
  - Combination of linear filters [Durand & Dorsey 2002, Yang et al. 2009]
  - Bilateral grid [Chen et al. 2007]
  - Permutohedral lattice [Adams et al. 2010]
  - Domain transform [Gastal & Oliveira 2011]

Fast bilateral: combination of linear filters [Durand & Dorsey 2002]

- General idea: approximate an expensive non-linear filter with a linear combination of fast linear filters
- Filter N times assuming that the pixel value in the range term is equal v_1, v_2, …, v_N
  - This is a good approximation for pixels close to the fixed value (v_1, v_2, …, v_N)
  - And, as we have N results, we can interpolate to get a good approximation for any value



Fast bilateral: combination of linear filters [Durand & Dorsey 2002]

- The original non-separable bilateral filter

$$y(\boldsymbol{p}) = \frac{\sum_{\boldsymbol{q}\in\Omega} x(\boldsymbol{q})g_s(\boldsymbol{p}-\boldsymbol{q})g_r(x(\boldsymbol{p})-x(\boldsymbol{q}))}{\sum_{\boldsymbol{q}\in\Omega} g_s(\boldsymbol{p}-\boldsymbol{q})g_r(x(\boldsymbol{p})-x(\boldsymbol{q}))}$$

- Let us fix $x(\boldsymbol{p}) = v_1 = const.$ in $g_r()$

$$\tilde{y}(\boldsymbol{p}; v_1) = \frac{\sum_{\boldsymbol{q}\in\Omega} x(\boldsymbol{q})g_r(v_1-x(\boldsymbol{q}))g_s(\boldsymbol{p}-\boldsymbol{q})}{\sum_{\boldsymbol{q}\in\Omega} g_r(v_1-x(\boldsymbol{q}))g_s(\boldsymbol{p}-\boldsymbol{q})}$$

$$\tilde{y}(\boldsymbol{p}; v_1) = \frac{\sum_{\boldsymbol{q}\in\Omega} h(\boldsymbol{q})g_s(\boldsymbol{p}-\boldsymbol{q}))}{\sum_{\boldsymbol{q}\in\Omega} l(\boldsymbol{q})g_s(\boldsymbol{p}-\boldsymbol{q})}$$

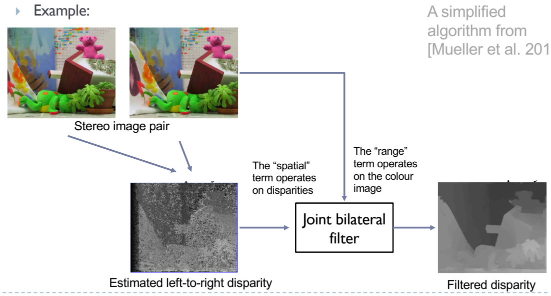
- Final image:

$$y(\boldsymbol{p}) \approx interpolate\ (v_k \to \tilde{y}(p; v_k), x(\boldsymbol{p}))$$

Joint-bilateral filter

- The "range" term does not need to operate in the same domain as the filter output
  - Example:

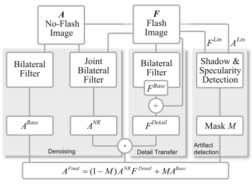
A simplified algorithm from [Mueller et al. 2010]



Joint bilateral filter: Flash / no-flash

Flash No-flash

- Preserve colour and illumination from the no-flash image
- Use flash image to remove noise and add details
- [Petshnigg et al. 2004]



Example of edge preserving filtering

- Domain Transform for Edge-Aware Image and Video Processing
- Video:
  - https://youtu.be/UlIxh1IQrTY?t=4m10s
  - From: http://inf.ufrgs.br/~eslgastal/DomainTransform/

Optimization-based methods

Poisson image editing [Perez et al. 2003]

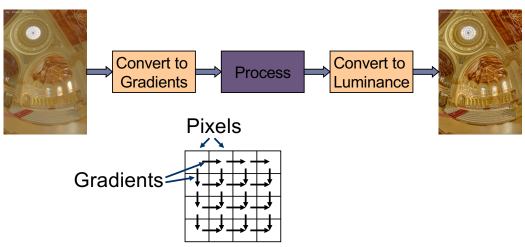
Gradient Domain compositing

- Compositing [Wang et al. 2004]

images from [Drori at al. 2004]

Gradient domain methods

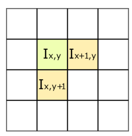
- Operate on pixel gradients instead of pixel values



Forward Transformation

- Forward Transformation
  - Compute gradients as differences between a pixel and its two neighboors

$$\nabla I_{x,y} = \begin{bmatrix} I_{x+1,y} - I_{x,y} \\ I_{x,y+1} - I_{x,y} \end{bmatrix}$$



  - Result: 2D gradient map (2 x more values then the number of pixels)

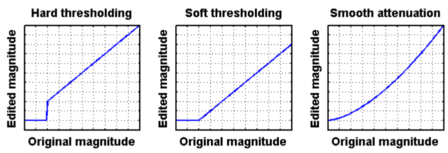
Processing gradient field

- Usually gradient magnitudes are modified while gradient direction (angle) remains the same

Gradient editing function

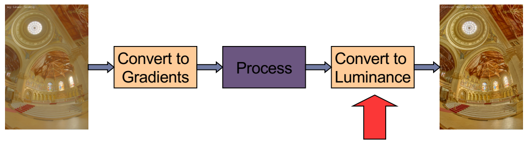
$$G_{x,y} = \nabla I_{x,y} \cdot f\left(||\nabla I_{x,y}||\right)$$

- Examples of gradient editing functions:



Inverse transform: the difficult part

- There is no strightforward transformation from gradients to luminance



- Instead, a minimization problems is solved:

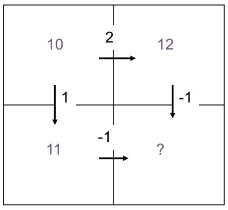
$$\arg\min_{I} \sum_{x,y} \left[\left(I_{x+1,y} - I_{x,y} - G^{(x)}_{x,y}\right)^2 + \left(I_{x,y+1} - I_{x,y} - G^{(y)}_{x,y}\right)^2 \right]$$

Image Pixels

Desired gradients

Inverse transformation

- Convert modified gradients to pixel values
  - Not trivial!
  - Most gradient fields are inconsistent - do not produce valid images
  - If no accurate solution is available, take the best possible solution
  - Analogy: system of springs



Gradient field reconstruction: derivation

- The minimization problem is given by:

$$\arg\min_{I} \sum_{x,y} \left[\left(I_{x+1,y} - I_{x,y} - G^{(x)}_{x,y}\right)^2 + \left(I_{x,y+1} - I_{x,y} - G^{(y)}_{x,y}\right)^2 \right]$$

- After equating derivatives over pixel values to 0 we get:
  - Dervation done in the lecture

$$I_{x-1,y} + I_{x+1,y} + I_{x,y-1} + I_{x,y+1} - 4I_{x,y} = G^{(x)}_{x,y} - G^{(x)}_{x-1,y} + G^{(y)}_{x,y} - G^{(y)}_{x,y-1}$$

  - In matrix notation:

Laplace operator (NxN matrix)

$$\nabla^2 I = \mathrm{div} G$$

Divergence of a vector field (Nx1 vector)

$$\nabla^2 I_{x,y} = I_{x-1,y} + I_{x+1,y} + I_{x,y-1} + I_{x,y+1} - 4I_{x,y}$$

Image as a column vector

$$\begin{bmatrix} I_{1,1} \\ I_{2,1} \\ \dots \\ I_{N,M} \end{bmatrix}$$

$$\mathrm{div} G_{x,y} = G^{(x)}_{x,y} - G^{(x)}_{x-1,y} + G^{(y)}_{x,y} - G^{(y)}_{x,y-1}$$

Laplace operator for 3x3 image

$$\nabla^2 = \begin{bmatrix} -2 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -3 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -3 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -3 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -3 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -2 \end{bmatrix}$$

Solving sparse linear systems

- Just use "\" operator in Matlab / Octave:
  - x = A \ b;
- Great "cookbook":
  - TEUKOLSKY, S.A., FLANNERY, B.P., PRESS, W.H., AND VETTERLING, W.T. 1992. *Numerical recipes in C*. Cambridge University Press, Cambridge.
- Some general methods
  - Cosine-transform – fast but cannot work with weights (next slides) and may suffer from floating point precision errors
  - Multi-grid – fast, difficult to implement, not very flexible
  - Conjugate gradient / bi-conjugate gradient – general, memory efficient, iterative but fast converging

Pinching artefacts

- A common problem of gradient-based methods is that they may result in “pinching” artefacts (left image)
- Such artefacts can be avoided by introducing weights to the optimization problem


31

Weighted gradients

- The new objective function is:

$$\arg\min_I \sum_{x,y} \left[w^{(x)}_{x,y} \left(I_{x+1,y} - I_{x,y} - G^{(x)}_{x,y}\right)^2 + w^{(y)}_{x,y} \left(I_{x,y+1} - I_{x,y} - G^{(y)}_{x,y}\right)^2 \right]$$

- so that higher weights are assigned to low gradient magnitudes (in the original image).

$$w^{(x)}_{x,y} = w^{(y)}_{x,y} = \frac{1}{||\nabla I^{(o)}_{x,y}|| + \epsilon}$$

- The linear system can be derived again
  - but this is a lot of work and is error-prone

32

Least-squares – matrix notation

- Given an error function (in the matrix notation):

$$F = (Ax - b)'(Ax - b)$$

- It’s derivative is given by:

$$\frac{\partial F}{\partial x} = 2\, A' A x - 2\, A' b$$

  - See for example 15.4 in Numerical Recipes

33

Weighted gradients - matrix notation (1)

- The objective function:

$$\arg\min_I \sum_{x,y} \left[w^{(x)}_{x,y} \left(I_{x+1,y} - I_{x,y} - G^{(x)}_{x,y}\right)^2 + w^{(y)}_{x,y} \left(I_{x,y+1} - I_{x,y} - G^{(y)}_{x,y}\right)^2 \right]$$

- In the matrix notation:

$$\arg\min_I \left(W \nabla_x I - W G^{(x)}\right)' \left(W \nabla_x I - W G^{(x)}\right) + \left(W \nabla_y I - W G^{(y)}\right)' \left(W \nabla_y I - W G^{(y)}\right).$$

- Gradient operators (for 3x3 pixel image):

$$\nabla_x = \begin{bmatrix} -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \nabla_y = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

34

Weighted gradients - matrix notation (2)

- The objective function again:

$$\arg\min_I \left(W \nabla_x I - W G^{(x)}\right)' \left(W \nabla_x I - W G^{(x)}\right) + \left(W \nabla_y I - W G^{(y)}\right)' \left(W \nabla_y I - W G^{(y)}\right).$$

- Derivates with respect to I:

$$\frac{\partial E}{\partial I} = 2\,(W \nabla_x)' W G^{(x)} + 2\,(W \nabla_y)' W G^{(y)} - 2\left[(W \nabla_x)'(W \nabla_x) + (W \nabla_y)'(W \nabla_y)\right] I$$

$$\boxed{F = (Ax - b)'(Ax - b) \qquad \frac{\partial F}{\partial x} = 2\,A' b - 2\,A' A x}$$

- The equation above can be solved using a sparse matrix solver

35

WLS filter: Edge stopping filter by optimization

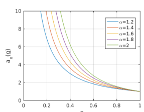
- Weighted-least-squares optimization

Make reconstructed image u possibly close to input g

Smooth out the image by making partial derivatives close to 0

$$\underset{\boldsymbol{u}}{\operatorname{argmin}} \sum_p \left((u_p - g_p)^2 + \lambda \left(a_{x,p}(g) \left(\frac{\partial u}{\partial x}\right)_p^2 + a_{y,p}(g) \left(\frac{\partial u}{\partial y}\right)_p^2 \right) \right)$$

Spatially varying smoothing – less smoothing near the edges

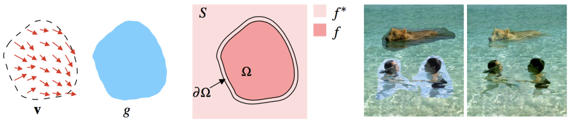


$$a_{x,p}(g) = \frac{1}{\left|\frac{\partial u}{\partial x}(g)\right|^{\alpha} + \epsilon}$$

- [Farbman et al., SIGGRAPH 2008]

36

Poisson image editing



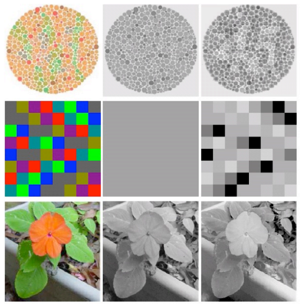
$$\min_f \iint_\Omega |\nabla f - \mathbf{v}|^2 \quad \text{subject to:} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

- Reconstruct unknown values *f* given a source guidance gradient field *v* and the boundary conditions $f|_{\partial\Omega} = f^*|_{\partial\Omega}$
- [Perez et al. 2003]

37

Color 2 Gray

- Transform color images to gray scale
- Preserve color saliency
  - When gradient in luminance close to 0
  - Replace it with gradient in chrominance
  - Reconstruct an image from gradients
- [Gooch et al. 2005]



February 28, 2017

Gradient Domain: applications

- More applications:
  - Lightness perception (Retinex) [Horn 1974]
  - Matting [Sun et al. 2004]
  - Color to gray mapping [Gooch et al. 2005]
  - Video Editing [Perez at al. 2003, Agarwala et al. 2004]
  - Photoshop's Healing Brush [Georgiev 2005]

February 28, 2017

References

- F. Durand and J. Dorsey, "Fast bilateral filtering for the display of high-dynamic-range images," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 257–266, Jul. 2002.
- E. S. L. Gastal and M. M. Oliveira, "Domain transform for edge-aware image and video processing," *ACM Trans. Graph.*, vol. 30, no. 4, p. 1, Jul. 2011.
- Patrick Pérez, Michel Gangnet, and Andrew Blake. 2003. Poisson image editing. *ACM Trans. Graph.* 22, 3 (July 2003), 313-318. DOI: http://dx.doi.org/10.1145/882262.882269
- Zeev Farbman, Raanan Fattal, Dani Lischinski, and Richard Szeliski. 2008. Edge-preserving decompositions for multi-scale tone and detail manipulation. *ACM Trans. Graph.* 27, 3, Article 67 (August 2008), 10 pages. DOI: https://doi.org/10.1145/1360612.1360666

40