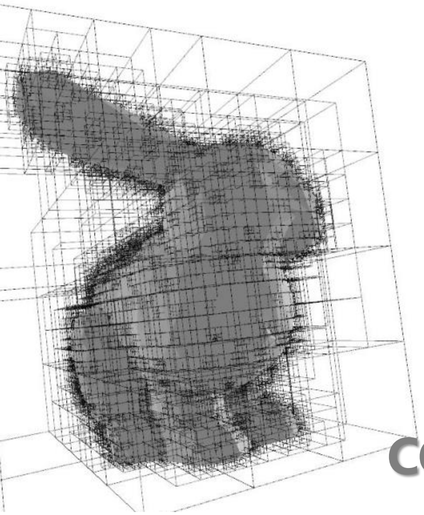
Image-Based Computer Graphics



Advanced image editing and processing tools

Image Composition

CGRA 352

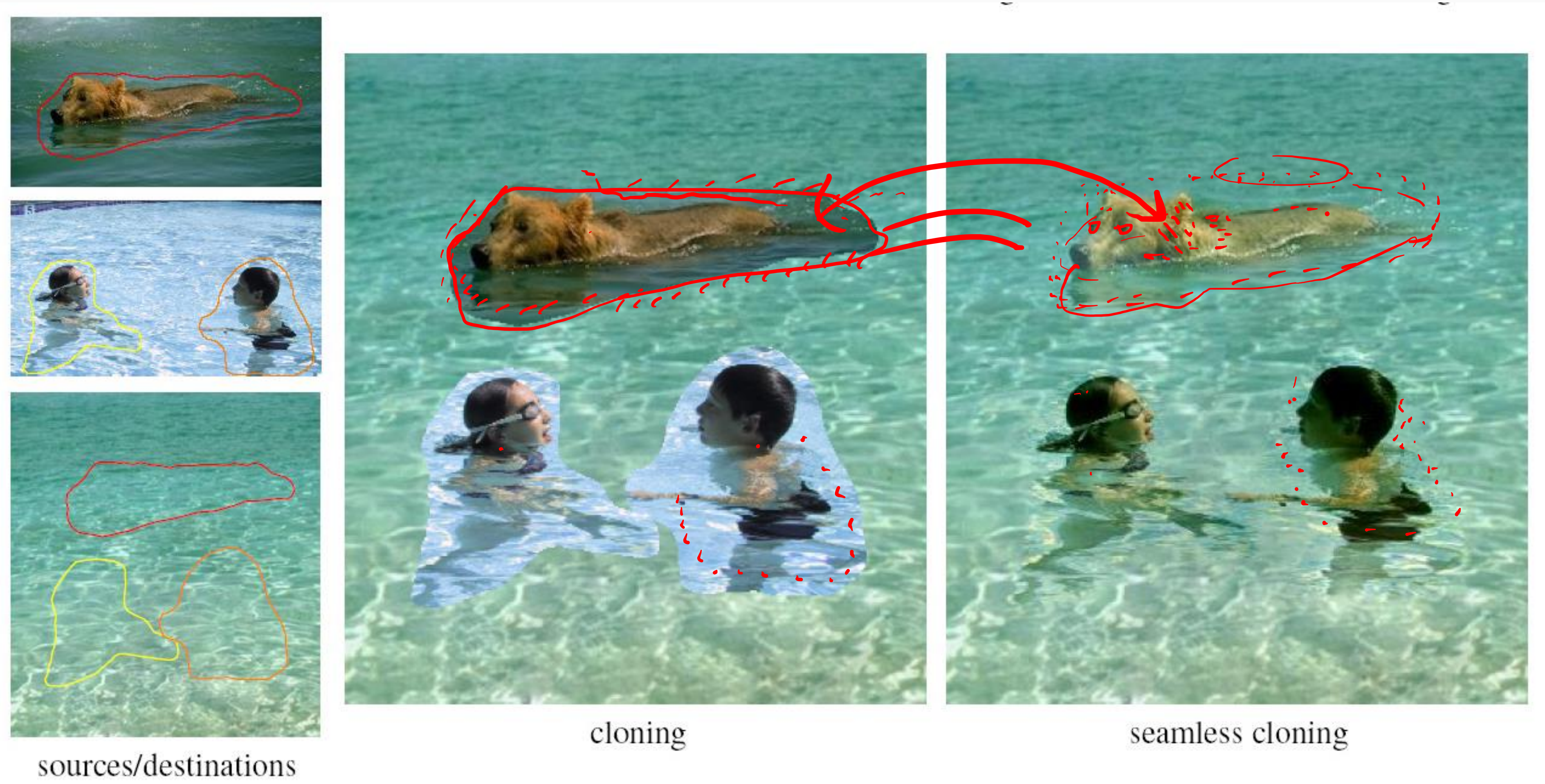
How to insert new objects?

- **Key idea:** Retain the gradient information as best as possible

sources | destinations | cloning | seamless cloning

Poisson editing for seamless cloning

- *What happened to the color after cloning?*

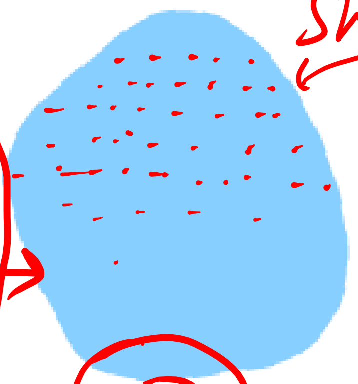


[Perez 2003]

How to insert new objects?

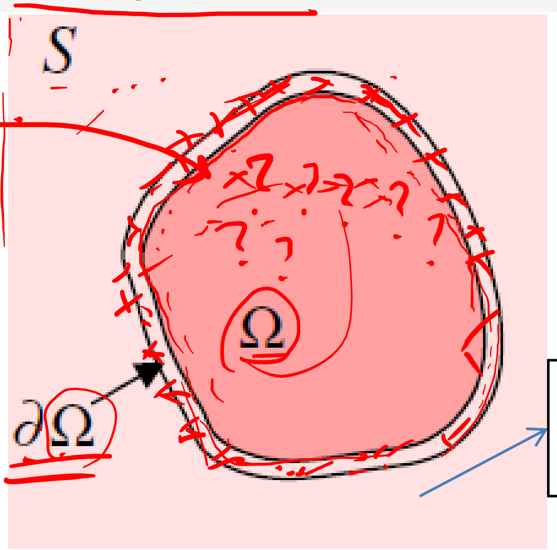
mapping
Img → 2D function
$f(x,y) \to V$ $(\vec{v})$

$(x,y) \to (R,G,B)$



Shape

S: Image domain



f^*

$f \to ?$

Closed subset of S

Source function g
(The original region to insert)

f*:Destination function
f: Unknown function

Δf.

Should f look like g?

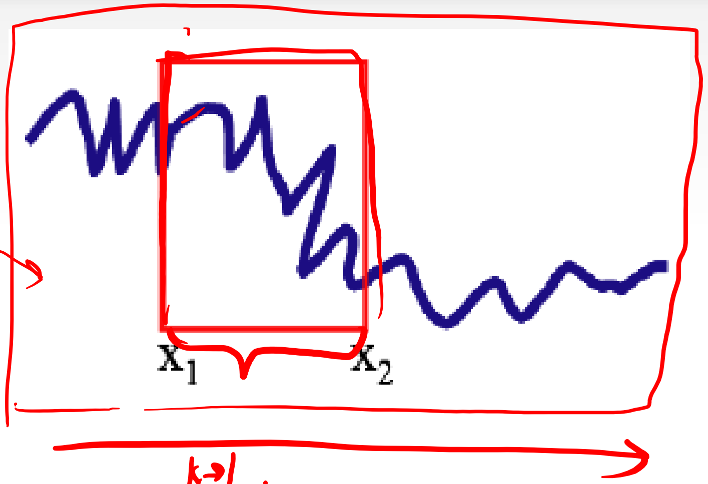
Should f look like f?* ✗

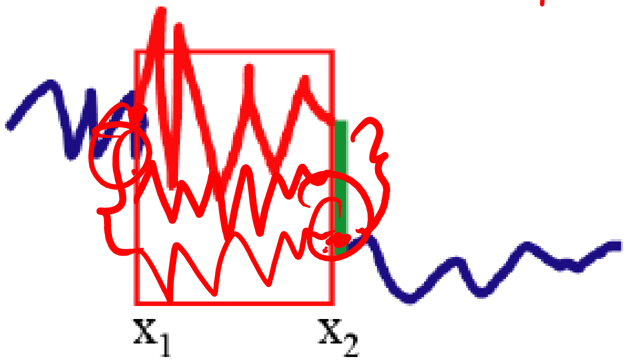
$f|_{\partial\Omega} = f^*|_{\partial\Omega}$

1D example

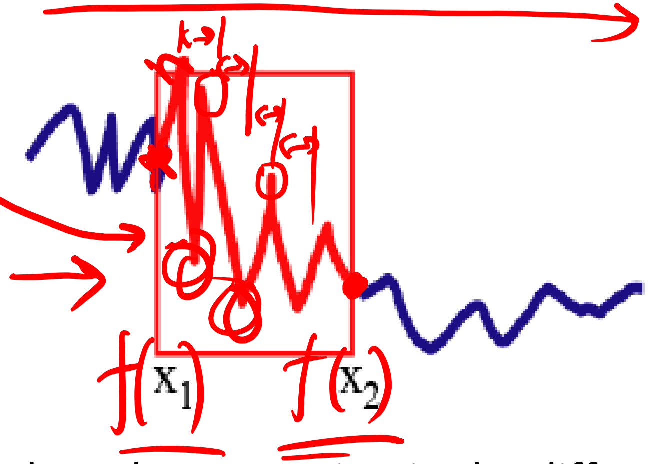
Seamlessly paste

onto



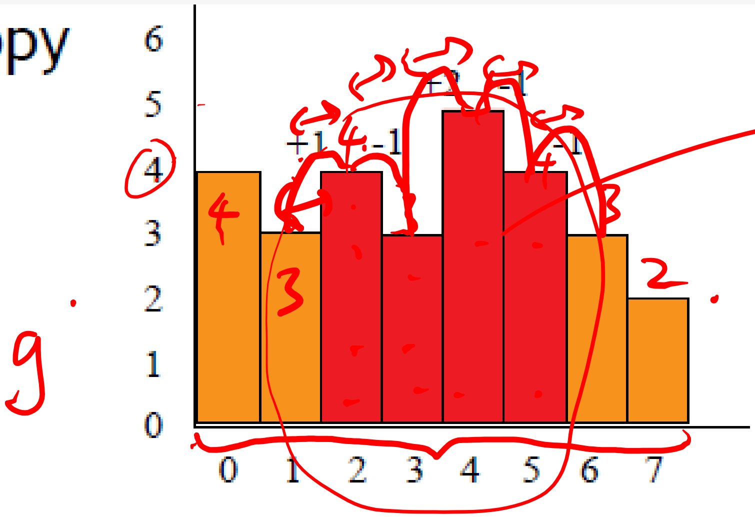


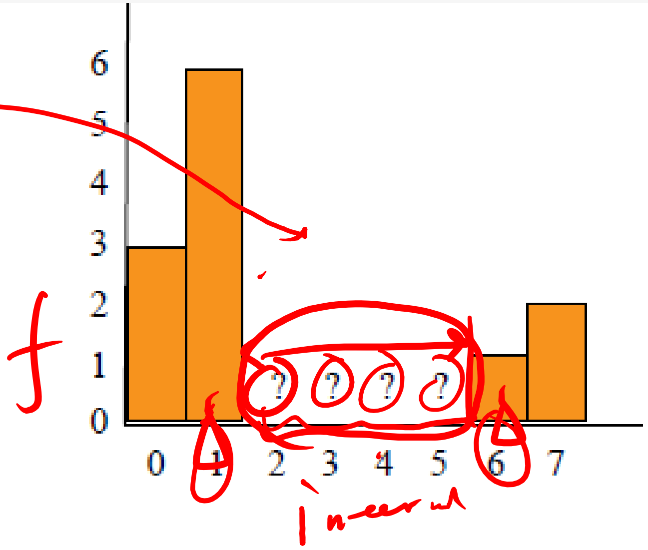
Directly put it here, obviously not a good idea



Adjust the values to maintain the differences between neighboring pixels while satisfying boundary conditions.

How we can do that

- Copy 

to 

$f_2\,?\quad f_3\,?\quad f_4\,?\quad f_5\,?$

4 unknowns

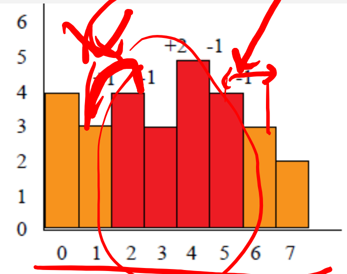
$f \to$ unknown

f^*.

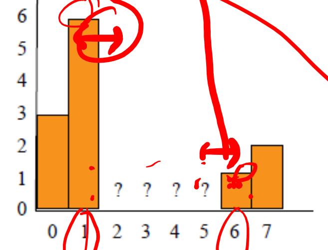
G: | 4 | 3 | 4 | 3 | 5 | 4 | 3 | 2 | ... | $\to$ F | 3 | 6 | ? | ? | ? | ? | 1 | 2 |

$g_0 \; g_1 \; g_2 \; \ldots\ldots \; g_6 \; g_7$ $\qquad f_0 \; f_1 \; f_2 \; f_3 \; f_4 \; f_5 \; f_6 \; f_7$

- Copy



to



$\Delta G \qquad \Delta g_1 = +1 \quad \Delta g_2 = -1.$

$\Delta g_3 = +2. \quad \Delta g_4 = -1. \; \Delta g_5 = -1.$

$\Delta f_1 \approx \Delta g_1.$

$\Delta f_2 \approx \Delta g_2$

$\ldots\ldots \quad \Delta f_5 \approx \Delta g_5$

f

$f_1 = f_1^*$

$f_6 = f_6^*$

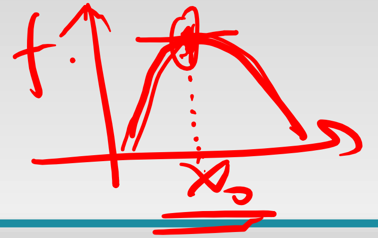
Obj: $\boxed{\min \; (\Delta f_1 - \Delta g_1)^2 + (\Delta f_2 - \Delta g_2)^2 + \ldots\ldots (\Delta f_5 - \Delta g_5)^2.}$

$\Delta f_1 = f_2 - f_1 \qquad \Delta g_1 = g_2 - g_1.$ ✓ constant.

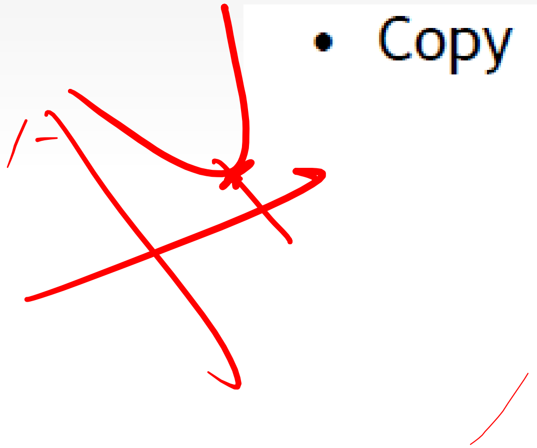
$\min . \left[(f_2 - f_1 - 1.)^2 + (f_3 - f_2 - (-1))^2 + \ldots\ldots \right]$

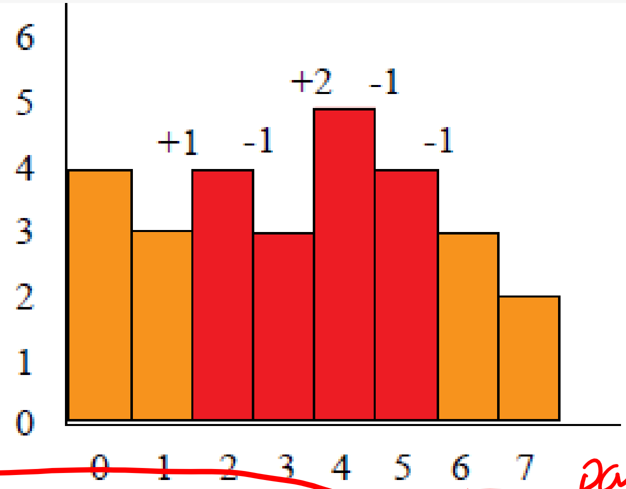
$\min \left[(f_2 - 6 - 1)^2 + (f_3 - f_2 + 1)^2 + \ldots\ldots \right] = F(f_2, f_3, f_4, f_5)$

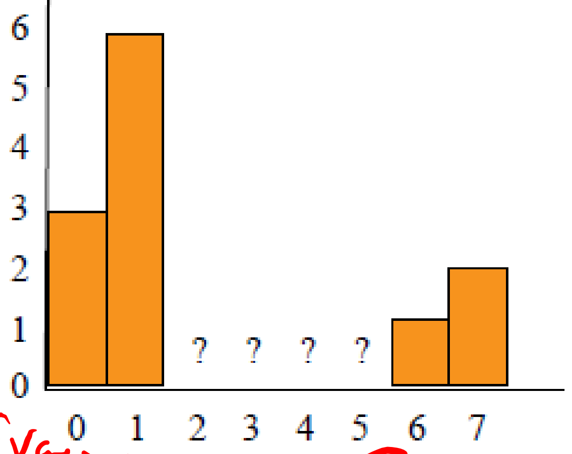
How we can do that



$$f'(x_0) = 0$$



- Copy 

to 

partial derivative $= 0$

$$\text{Min}\,(f_2^2+49-14f_2 + f_3^2+f_2^2+1-2f_3f_2 +2f_3-2f_2 + f_4^2+f_3^2+4-2f_3f_4 -4f_4+4f_3 + f_5^2+f_4^2+1-2f_5f_4 +2f_5-2f_4 + f_5^2+4-4f_5)$$

Denote it Q

$$\frac{dQ}{df_2} = 2f_2 + 2f_2 - 2f_3 - 16 = 0$$

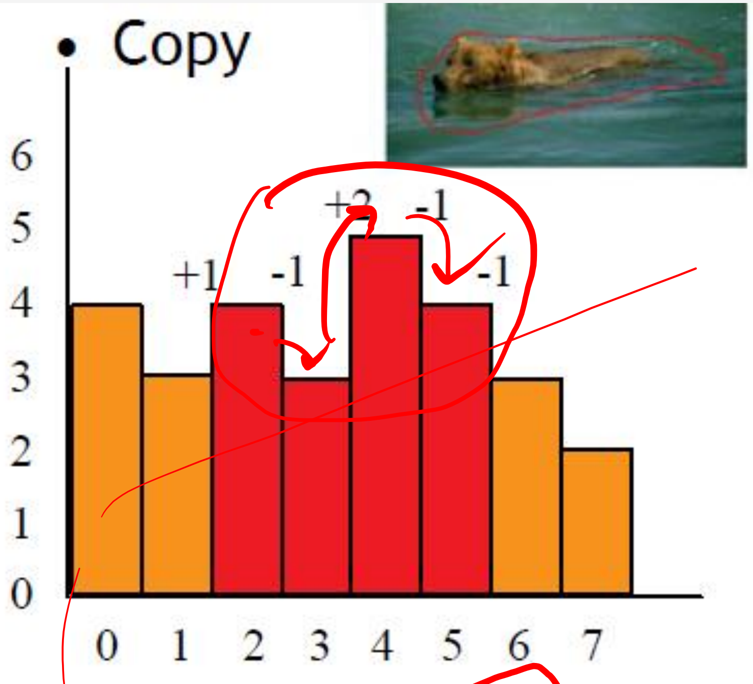
$$\frac{dQ}{df_3} = 2f_3 - 2f_2 + 2 + 2f_3 - 2f_4 + 4 = 0$$

$$\frac{dQ}{df_4} = 2f_4 - 2f_3 - 4 + 2f_4 - 2f_5 - 2 = 0$$

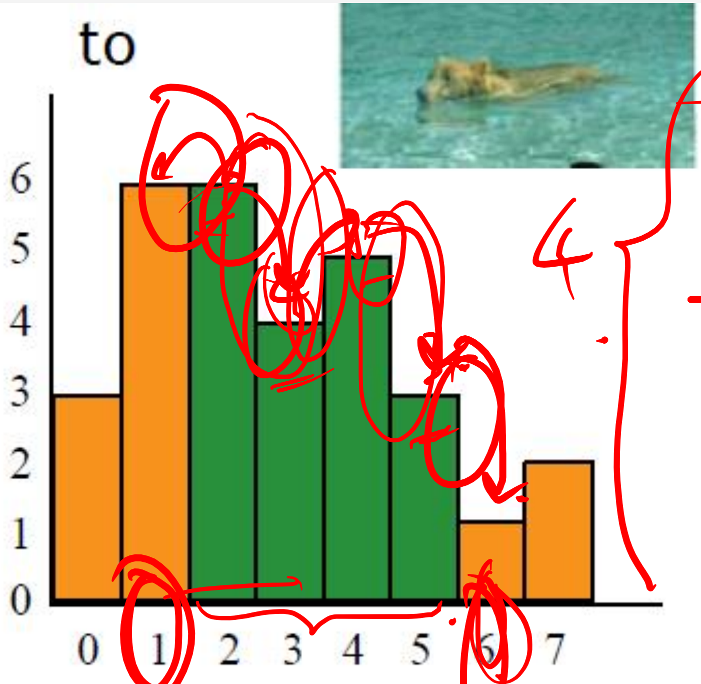
$$\frac{dQ}{df_5} = 2f_5 - 2f_4 + 2 + 2f_5 - 4 = 0$$

Minimize it by solving linear systems

- Copy



to



$$4f_2 - 2f_3 + 0 \cdot f_4 + 0 f_5 = 16$$

$$-2f_2 + 4f_3 - 2f_4 + 0 \cdot f_5 = -6$$

$$\begin{pmatrix} 4 & -2 & 0 & 0 \\ -2 & 4 & -2 & 0 \\ 0 & -2 & 4 & -2 \\ 0 & 0 & -2 & 4 \end{pmatrix} \begin{pmatrix} f_2 \\ f_3 \\ f_4 \\ f_5 \end{pmatrix} = \begin{pmatrix} 16 \\ -6 \\ 6 \\ 2 \end{pmatrix}$$

$$\boxed{\begin{pmatrix} f_2 \\ f_3 \\ f_4 \\ f_5 \end{pmatrix} = \begin{pmatrix} 6 \\ 4 \\ 5 \\ 3 \end{pmatrix}}$$

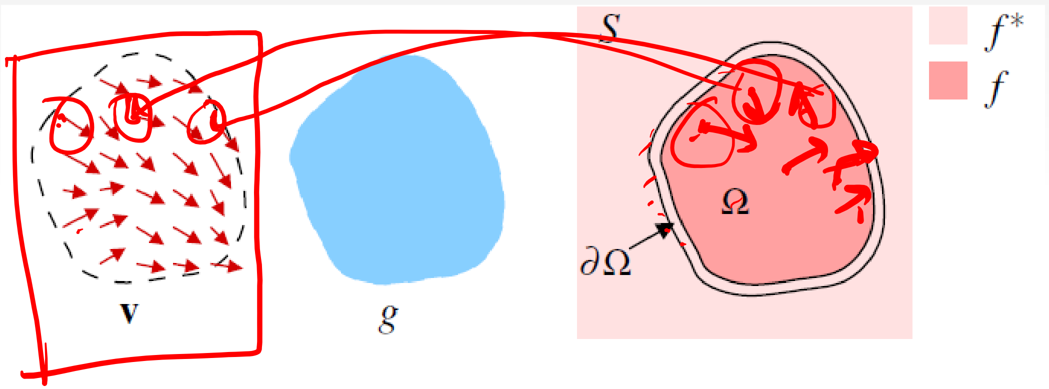
$$\begin{bmatrix} f_2 \\ f_3 \\ f_4 \\ f_5 \end{bmatrix} = \begin{bmatrix} 16 \\ \\ \\ \end{bmatrix}$$

4x4   1x4   1x4

$A x = B \rightarrow x$ ✓

2D poisson blending

- In 2D, the gradient of the source image region is a vector field v.
- We want to minimize the difference:



$\sum\sum (\nabla f - \vec{v})^2$

$$\min_f \iint_\Omega \left|\nabla f - \mathbf{v}\right|^2 \; with \; f|_{\partial\Omega} = f^*|_{\partial\Omega} \qquad \mathbf{v} = (u, v) = \nabla g$$

Image gradient

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

$\left(\frac{\partial g}{\partial x}, \frac{\partial g}{\partial y}\right)$

what does this term do?

Gradient of f looks like gradient of g

what does this term do?

f is equivalent to f* at the boundaries

V is known, since the source function g is known

2D poisson blending

$$\min_f \iint_\Omega \left|\nabla f - \mathbf{v}\right|^2 \; with \quad f\,|_{\partial\Omega} = f^*\,|_{\partial\Omega}$$

$$|\nabla f - \boldsymbol{v}|^2 = \left(\frac{\partial f}{\partial x} - u\right)^2 + \left(\frac{\partial f}{\partial y} - v\right)^2$$

$$\boxed{(u, v) = \nabla g}$$

2D poisson blending

$$\min_f \iint_\Omega |\nabla f - \mathbf{v}|^2 \ \textit{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

$$|\nabla f - \boldsymbol{v}|^2 = \left(\frac{\partial f}{\partial x} - u\right)^2 + \left(\frac{\partial f}{\partial y} - v\right)^2 \qquad \boxed{(u, v) = \nabla g}$$

- **The solution for minimization is the unique solution for this Poisson equation (with Dirichlet boundary conditions)**

$$\Delta f = \operatorname{div} \mathbf{v} \quad \text{over} \quad \Omega, \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Laplacian

$$div\ \mathbf{v} = \frac{\partial v}{\partial y} + \frac{\partial u}{\partial x} \Longrightarrow \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = \frac{\partial v}{\partial y} + \frac{\partial u}{\partial x}$$

Divergence

2D poisson blending

So what does this mean ...

Gradient $\mathbf{v} = (u, v) = \nabla g$

Laplacian $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$

$$\begin{aligned} \text{div } \mathbf{v} &= \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \\ &= \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2} \\ &= \Delta g \end{aligned}$$

Laplacian of f same as g

$$\Delta f = \text{div } \mathbf{v} \quad \text{over} \quad \Omega, \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

Dirichlet boundary condition

It specifies the values of a solution needs to take on the boundary of the domain

2D poisson blending

- Let's first recall gradient again

first-order finite difference

$$f'(x) = \lim_{h \to 0} \frac{f(x+0.5h) - f(x-0.5h)}{h}$$

derivative filter

1	0	-1

second-order finite difference

$$f''(x) \approx \frac{\delta_h^2[f](x)}{h^2} = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}.$$

Laplace filter

1	-2	1

$$It\ is\ I(x+1) + I(x-1) - 2I(x)$$

For 2D

$$\Delta f = \frac{\partial f^2}{\partial x^2} + \frac{\partial f^2}{\partial y^2} = I(x+1,y) + I(x-1,y) - 2I(x,y) + I(x,y+1) + I(x,y-1) - 2I(x,y)$$

$$= I(x+1,y) + I(x-1,y) + I(x,y+1) + I(x,y-1) - 4I(x,y)$$

0	1	0
1	-4	1
0	1	0

2D Laplace filter

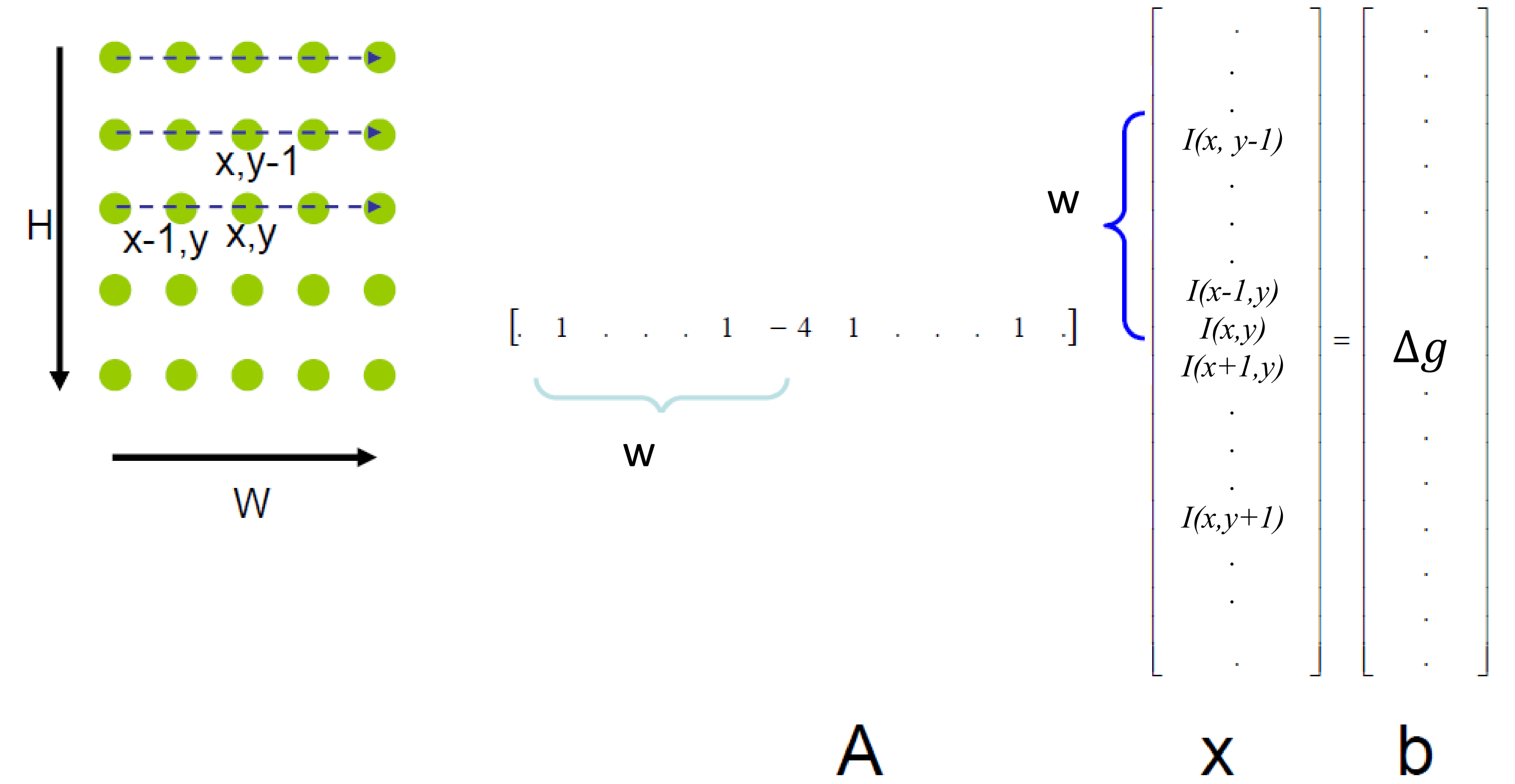
Linear System

So for each pixel p

$$\Delta f_p = \Delta g_p$$

$$\sum_{q \in N_p} f_q - 4 f_p = \sum_{q \in N_p} g_q - 4 g_p$$

What's known, what's unknown?



$$I(x+1,y) + I(x-1,y) + I(x,y+1) + I(x,y-1) - 4I(x,y) = \Delta g$$

We have know G, so the second derivative is know for every point.

Δg

Linear System

- **In that linear system, note that the pixels at the region borders are also known as the same with the target image.** So in vector “x”, not all the pixels need to be solved
- Matrix A will look like this:

$$
\begin{bmatrix}
 & & 1 & -4 & 1 & & & & 1 & & & & \\
 & & & 1 & -4 & 1 & & & & 1 & & & \\
1 & & & & 1 & -4 & 1 & & & & 1 & & \\
 & 1 & & & & 1 & -4 & 1 & & & & 1 & \\
 & & 1 & & & & 1 & -4 & 1 & & & & 1 \\
 & & & 1 & & & & 1 & -4 & 1 & & & \\
 & & & & 1 & & & & 1 & -4 & 1 & &
\end{bmatrix}
$$

Linear System

General form of linear least squares

(**Warning:** change of notation. x is a vector of parameters!)

$$E_{\mathrm{LLS}} = \sum_i |\boldsymbol{a}_i \boldsymbol{x} - \boldsymbol{b}_i|^2$$

$$= \|\mathbf{A}\boldsymbol{x} - \boldsymbol{b}\|^2 \qquad \text{(matrix form)}$$

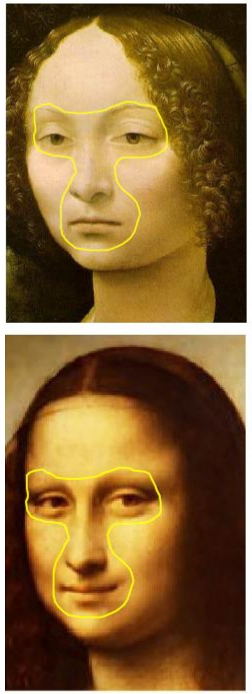
Minimize the error:

Expand

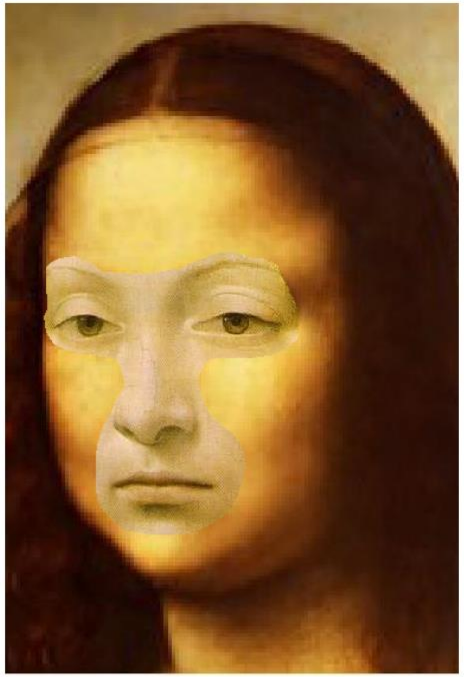
$$E_{\mathrm{LLS}} = \boldsymbol{x}^\top(\mathbf{A}^\top\mathbf{A})\boldsymbol{x} - 2\boldsymbol{x}^\top(\mathbf{A}^\top\boldsymbol{b}) + \|\boldsymbol{b}\|^2$$

Take derivative $\quad (\mathbf{A}^\top\mathbf{A})\boldsymbol{x} = \mathbf{A}^\top\boldsymbol{b} \quad$ (normal equation)

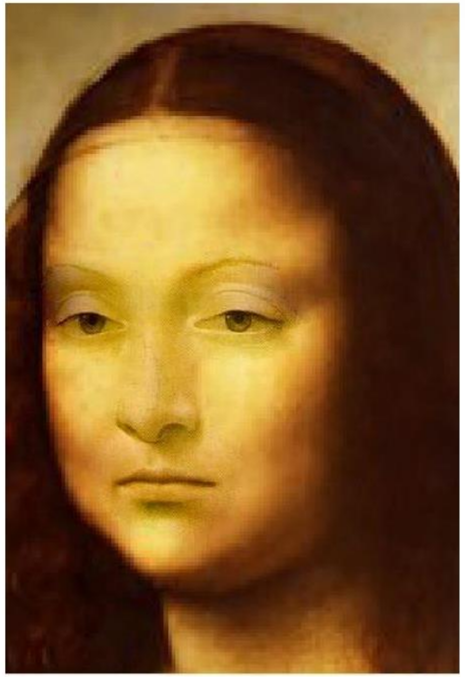
Solve for x $\quad \boldsymbol{x} = (\mathbf{A}^\top\mathbf{A})^{-1}\mathbf{A}^\top\boldsymbol{b}$



source/destination



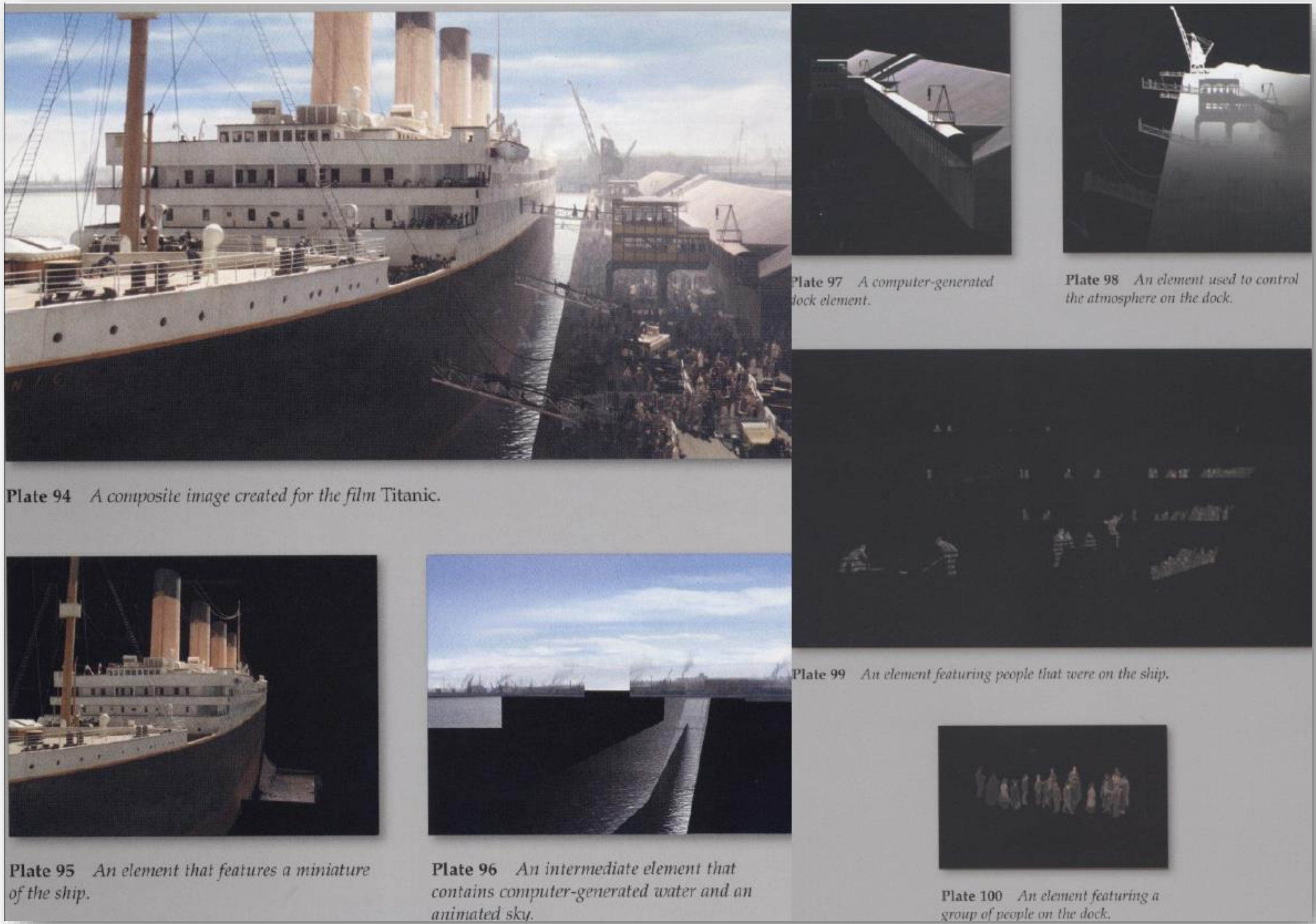
cloning



seamless cloning

OpenCV and Matlab both provide linear system solvers, if you can define your problem in this form, use them!

What if we do not want to do color blending?



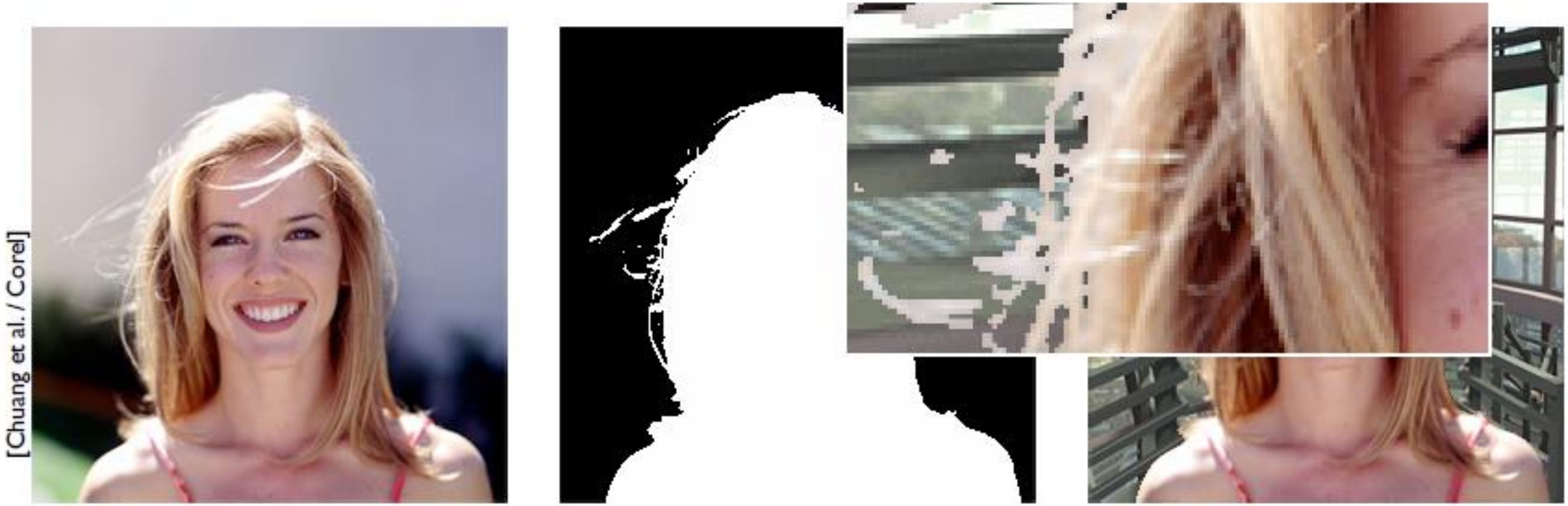
In many occasions, we do not want to change the color of the inserted objects.

From the Art & Science of Digital Compositing

What if we do not want to do color blending?

Magazine covers
We also need composite things with different background

If we only use binary mask…

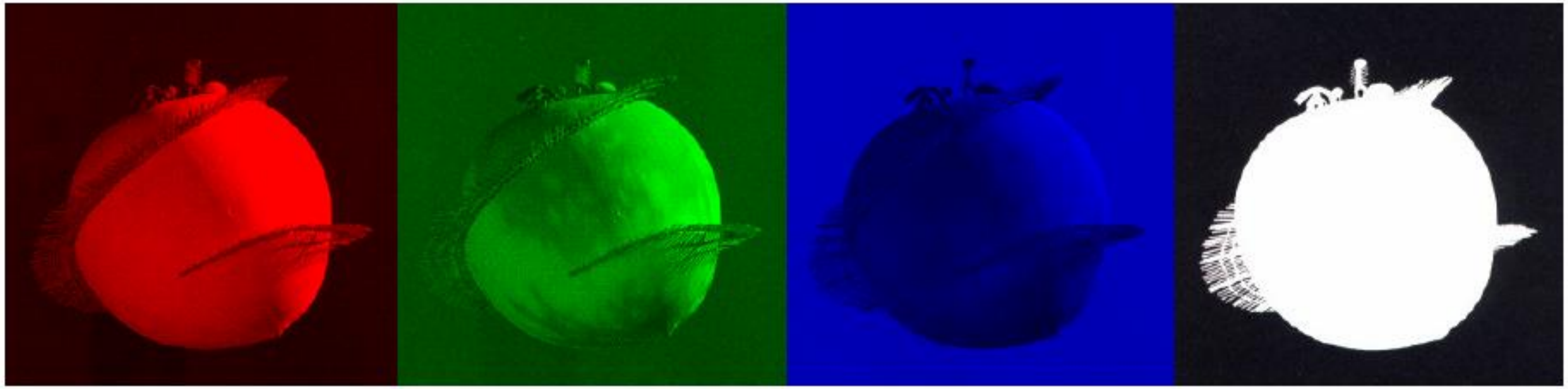


causes jaggy artifacts similar to point-sampled rasterization

Is this pixel part of the foreground? Only yes or no for a binary mask

Alpha matting

- **Key Idea:** pixels near boundary are not strictly foreground or background -- **adding an Alpha channel**

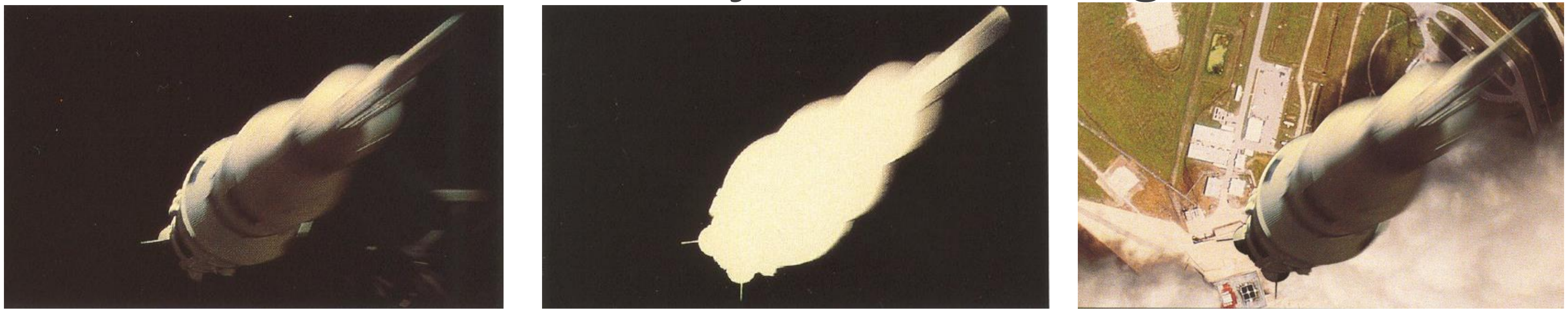


An extra alpha channel:
α=1 means opaque, α=0 means transparent

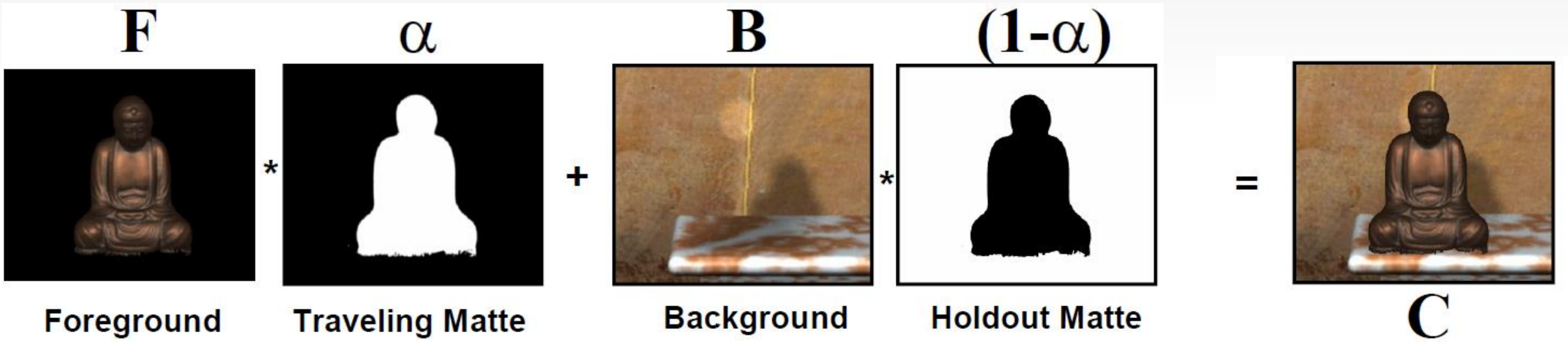
Why do we need fractional alpha?

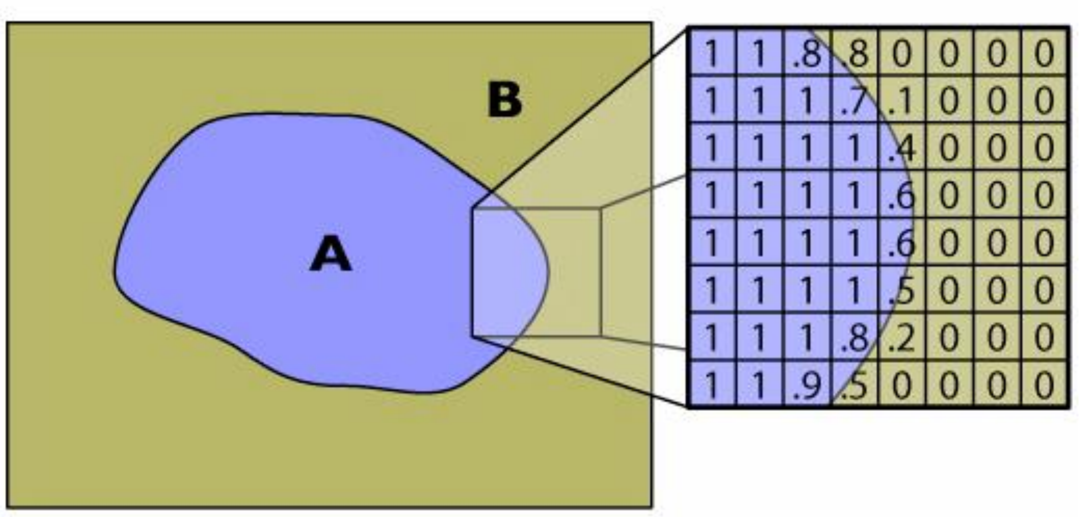
- Thin features (e.g. hair) cause mixed pixels

- Motion blurs "smears" objects into background



Alpha matting





The matting equation:

$$C = \alpha F + (1 - \alpha) B$$

α is a floating point number from 0 to 1

[Figure from Pat Hanrahan]

Why matting is hard

- $C = \alpha F + (1 - \alpha)B$ for three channels

Equations: **3**
3 channels of the observed color

$$\begin{aligned} C_r &= \alpha F_r + (1 - \alpha)B_r \\ C_g &= \alpha F_g + (1 - \alpha)B_g \\ C_b &= \alpha F_b + (1 - \alpha)B_b \end{aligned}$$

Unknowns: **7**
3 channels of the foreground/background color, and α

- We have to **get fewer unknowns (or more equations)**

Traditional blue/green screen matting

- Invented by Petro Vlahos (Technical Academy Award 1993)

Petro Vlahos
GORDON E. SAWYER AWARD
66TH ACADEMY AWARDS
1993

Initially for film, then video, then digital
- **Assume that the foreground has no blue/green**
- **Assume background is mainly blue/green**

Traditional blue/green screen matting

- **Idealized version:**
  - no blue in foreground. Only blue in background $F_b = 0 \; and \; B_r = B_g = 0$
  - Equations can be simplified:

$$
\begin{aligned}
C_r &= \alpha F_r + (1-\alpha)B_r \\
C_g &= \alpha F_g + (1-\alpha)B_g \\
C_b &= \alpha F_b + (1-\alpha)B_b
\end{aligned}
\Rightarrow
\begin{aligned}
C_r &= \alpha F_r \\
C_g &= \alpha F_g \\
C_b &= (1-\alpha)B_b
\end{aligned}
$$

3 equations in 3 unknowns

Problem Solved!

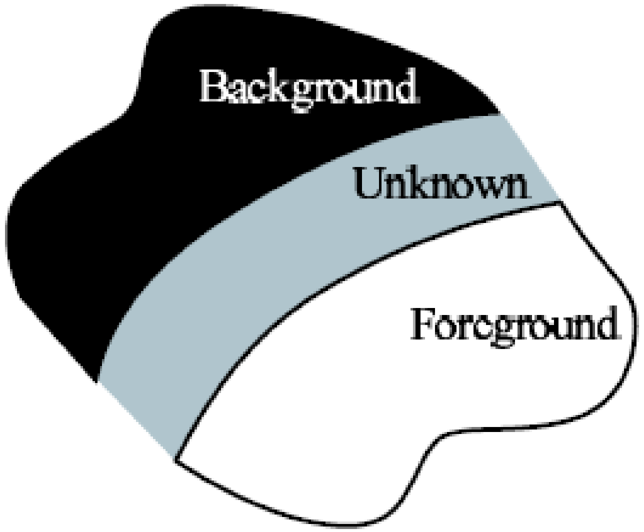
Issues

The background illuminates the foreground, blue/green at silhouettes

The background blue screen reflects blue on the wing surfaces

Natural Image Matting

- Someone has to specify which part is supposed to be extracted
- Normally take an initial binary map as input, then analyze the pixels along the boundaries.



From the initial boundaries, we derive a TRIMAP, where the alpha values should be solved.
(Otherwise alpha = 1 / 0 for foreground/background)

Natural Image Matting

- **Important assumption: F,B are approximately constant in a window (Local smooth assumption)**

Different weights for every pixel to combine foreground color  and background color 

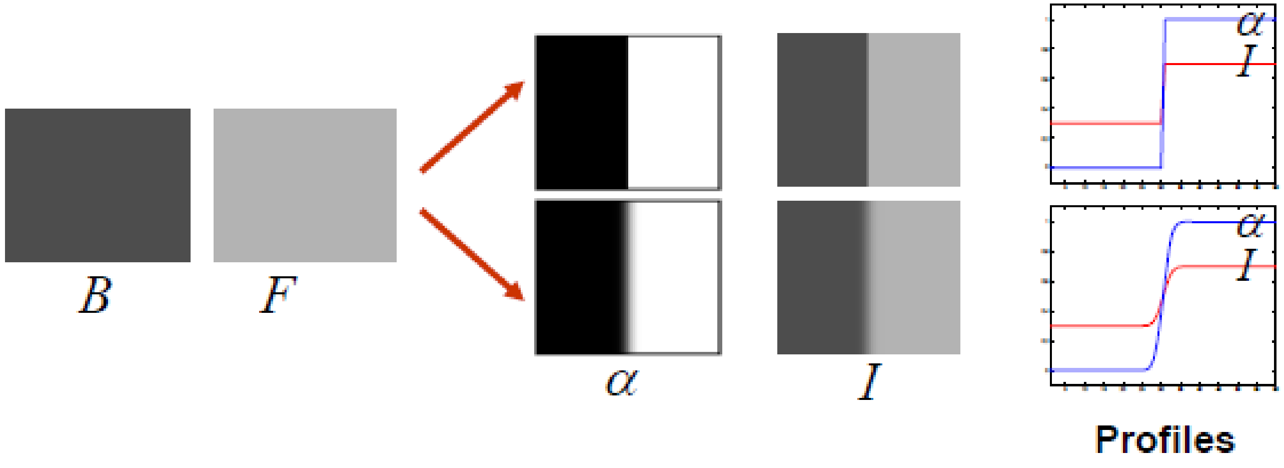
Natural Image Matting

- We can further assume that alpha is a linear transform of the input image within a local window

$$I_i \approx \alpha_i F + (1-\alpha_i)B \qquad i \in w$$

$$\Downarrow$$

$$\alpha_i \approx a I_i + b \qquad a = \frac{1}{F-B}, b = \frac{-B}{F-B}$$



Guided image filtering for fast matting

Guided image filtering:

- The key assumption of the guided filter is a local linear model between the guidance I and the filtering output q for input p. (The same as what we want to solve in the matting problem)

$$q_i = a_k I_i + b_k, \forall i \in \omega_k$$

some linear coefficients assumed to be constant in the window k

This local linear model ensures that q has an edge only if I has an edge

[He et al. TPAMI 2013}

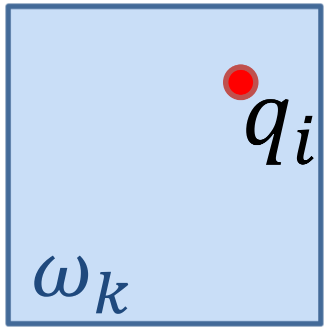
$$\alpha_i \approx aI_i + b \qquad a = \frac{1}{F-B}, b = \frac{-B}{F-B}$$

Guided image filtering

Guided image filtering:

Pixel in input image: p, output pixel: q, guidance map: I

- We seek a solution that minimizes the difference between **q** and **p** while maintaining that linear model



$$E(a_k, b_k) = \sum_{i \in \omega_k} \left((a_k I_i + b_k - p_i)^2 + \epsilon a_k^2 \right)$$

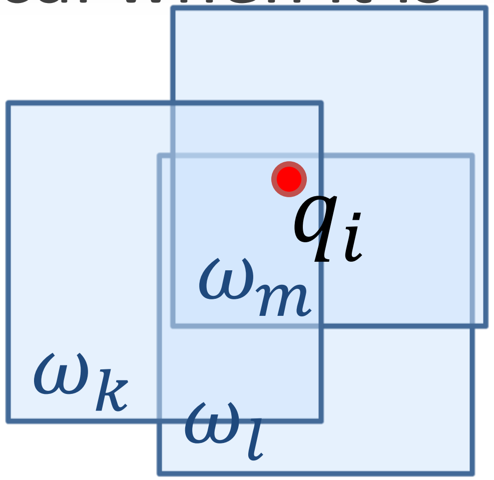
Using linear regression (see *Applied Regression Analysis. 2 edn*), the solution would be:

$$a_k = \frac{\frac{1}{|\omega|}\sum_{i \in \omega_k} I_i p_i - \mu_k \bar{p}_k}{\sigma_k^2 + \epsilon} \qquad b_k = \bar{p}_k - a_k \mu_k.$$

μ_k and σ_k^2 are the mean and variance of I in ω_k, $\bar{p}_k = \frac{1}{|\omega|}\sum_{i \in \omega_k} p_i$ is the mean of p in ω_k.

Guided image filtering

- However, a pixel i is involved in all the overlapping windows ω that covers i, so the value of output is not identical when it is computed in different windows.



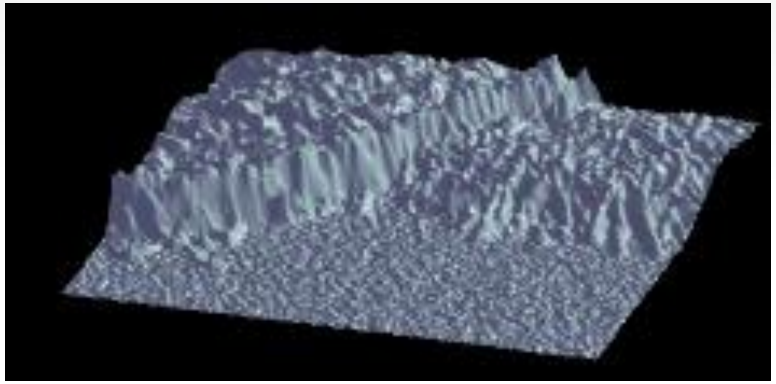
- We average all the possible values

$$q_i = \frac{1}{|\omega|} \sum_{k|i\in\omega_k} (a_k I_i + b_k)$$

- Noticing that $\sum_{k|i\in\omega_k} a_k = \sum_{k\in\omega_i} a_k$
- Rewrite the above equation as:

$$q_i = \bar{a}_i I_i + \bar{b}_i$$

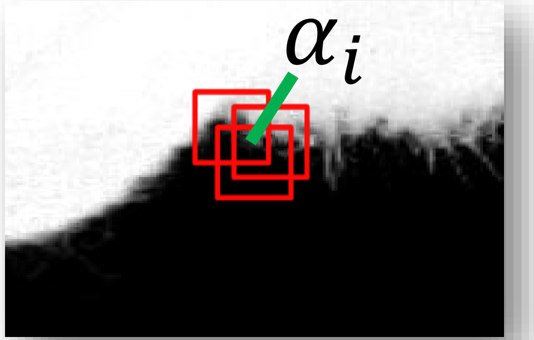
Filtering for matting

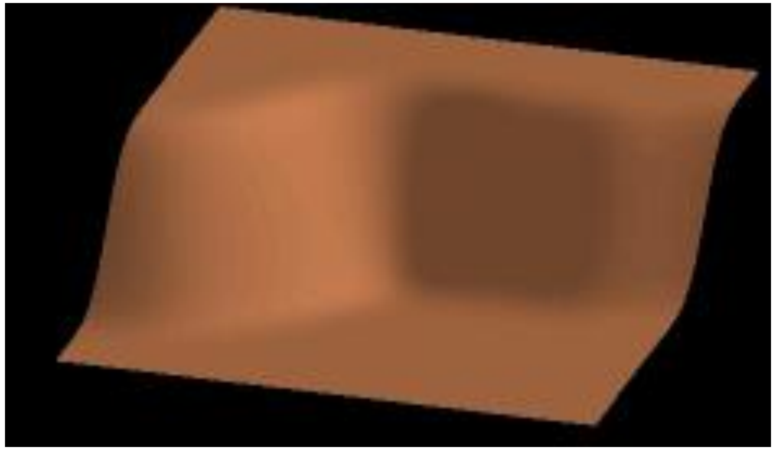


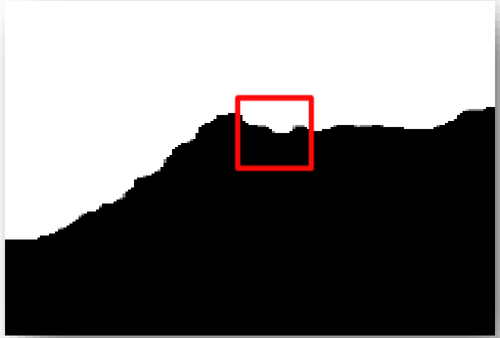
$Guide\ Image,\ \ I$

$$a_k = \frac{\frac{1}{|\omega|}\sum_{i\in\omega_k} I_i p_i - \mu_k \bar{p}_k}{\sigma_k^2 + \epsilon}$$

$$b_k = \bar{p}_k - a_k \mu_k.$$







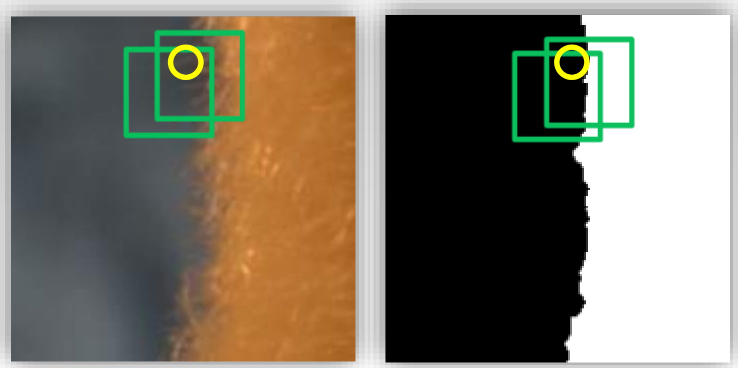
$Input\ \ Image,\ p$

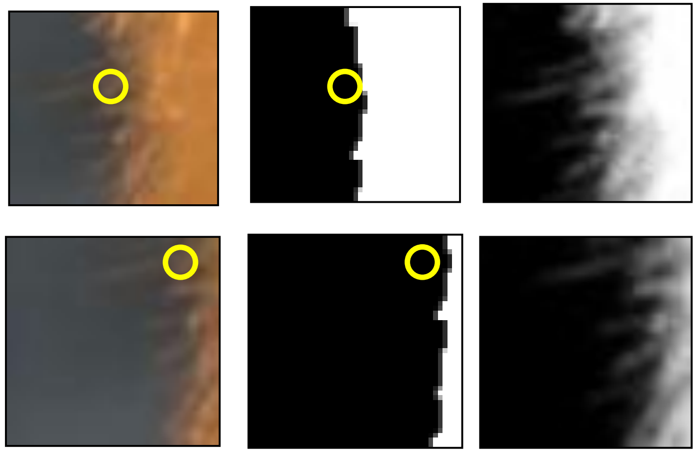
$(Initial\ \alpha, 0\ and\ 1)$

A window ω has a width of 2r

$$\boldsymbol{\alpha_i} = \frac{1}{|\omega|}\sum_{i\in\omega_k}(a_k I_i + b_k)$$

How to understand it?





Guide I   *Input, p* $\boldsymbol{\alpha}_i^k$

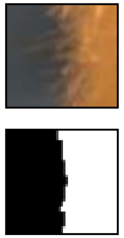
$$\alpha_i \approx aI_i + b \qquad a = \frac{1}{F-B}, b = \frac{-B}{F-B}$$

For each pixel in a window, we have $a_k = \dfrac{\frac{1}{|\omega|}\sum_{i\in\omega_k} I_i p_i - \mu_k \bar{p}_k}{\sigma_k^2 + \epsilon}$ $\quad b_k = \bar{p}_k - a_k\mu_k.$

$$\boldsymbol{\alpha}_i^k = a_k I_i + b_k = a_k I_i + (\overline{p_k} - a_k\mu_k) = a_k(I_i - \mu_k) + \overline{p_k}$$

$$a_k = \frac{\boxed{\frac{1}{|\omega|}\sum_{i\in\omega_k} I_i p_i - \mu_k \bar{p}_k}}{\sigma_k^2 + \epsilon}$$

Covariance shows the tendency in the linear relationship between the variables, if there are unwanted noisy parts, it will contribute less to the final result

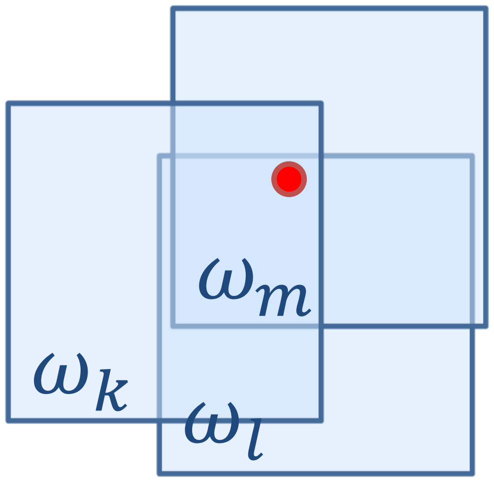


ϵ controls how smooth you want in the final results, normally you can take a small one like 0.1 to get finer details

So a_k controls how much it contributes to the transparency value
(For the pixel with an initial alpha value of 1, imagine that we should reduce some transparency from 1)

Filtering for matting

Input: filtering input image p, guidance image I, radius r, regularization ϵ
Output: filtering output α.



For every window in the image, compute all the values for a and b for all the covered pixels.

Then for every pixel, every computed a values and b values should be averaged

The abbreviations of correlation (corr), variance (var), and covariance (cov) indicate the intuitive meaning of these variables.

$$
\begin{array}{ll}
1: & \mathrm{mean}_I = f_{\mathrm{mean}}(I) \\
 & \mathrm{mean}_p = f_{\mathrm{mean}}(p) \\
 & \mathrm{corr}_I = f_{\mathrm{mean}}(I.*I) \\
 & \mathrm{corr}_{Ip} = f_{\mathrm{mean}}(I.*p) \\
2: & \mathrm{var}_I = \mathrm{corr}_I - \mathrm{mean}_I.*\mathrm{mean}_I \\
 & \mathrm{cov}_{Ip} = \mathrm{corr}_{Ip} - \mathrm{mean}_I.*\mathrm{mean}_p \\
3: & a = \mathrm{cov}_{Ip}./(\mathrm{var}_I + \epsilon) \\
 & b = \mathrm{mean}_p - a.*\mathrm{mean}_I \\
4: & \mathrm{mean}_a = f_{\mathrm{mean}}(a) \\
 & \mathrm{mean}_b = f_{\mathrm{mean}}(b) \\
5: & \alpha = \mathrm{mean}_a.*I + \mathrm{mean}_b
\end{array}
$$

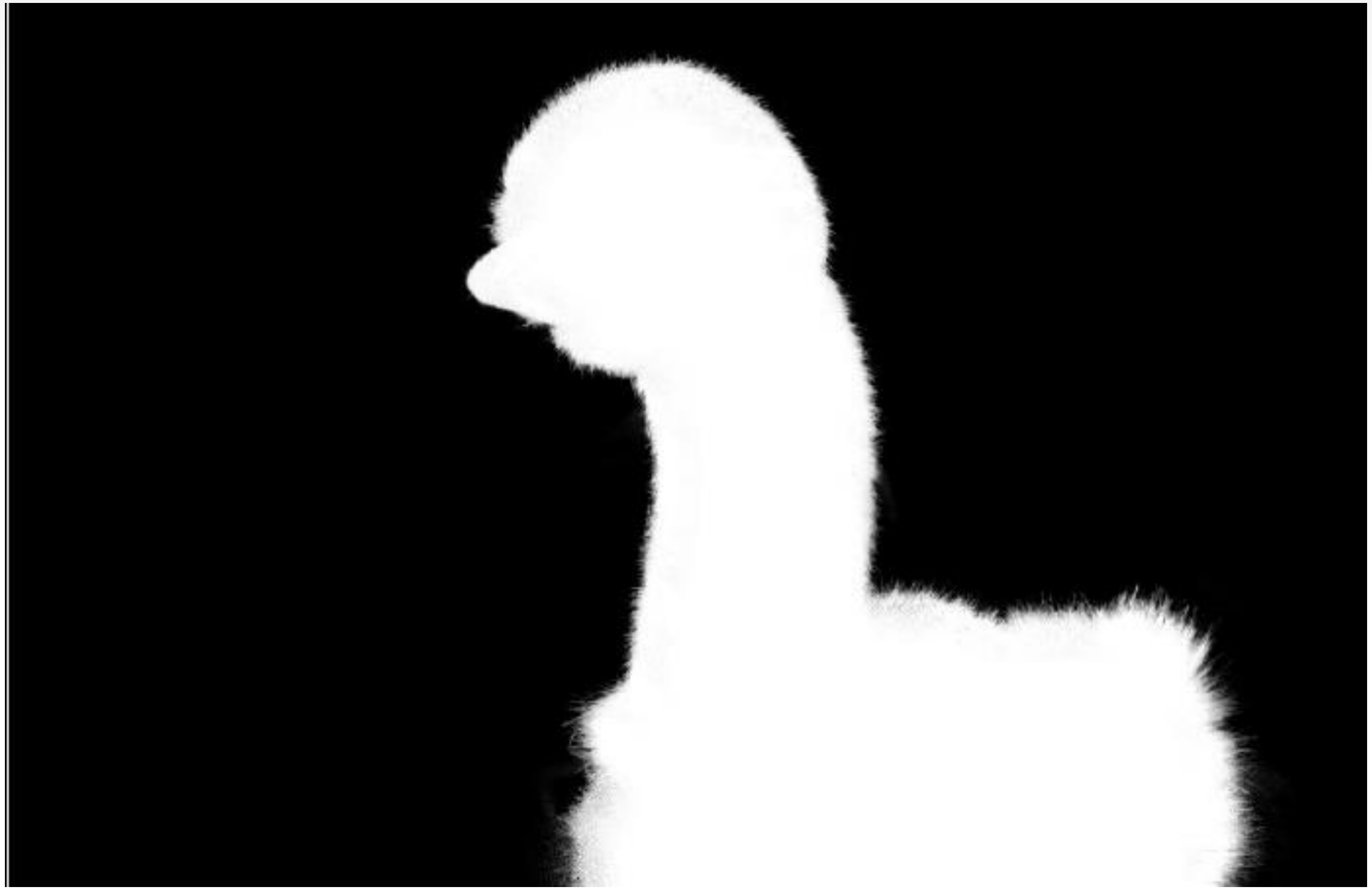
One result

Guide I

One result

Input mask, only 0 and 255 (representing 0 and 1)

One result



Filtered image gray level (representing alpha value from 0 to 1)

Matting results