Computational Photography

Matthias Zwicker, Siavash Bigdeli
University of Bern
Fall 2014

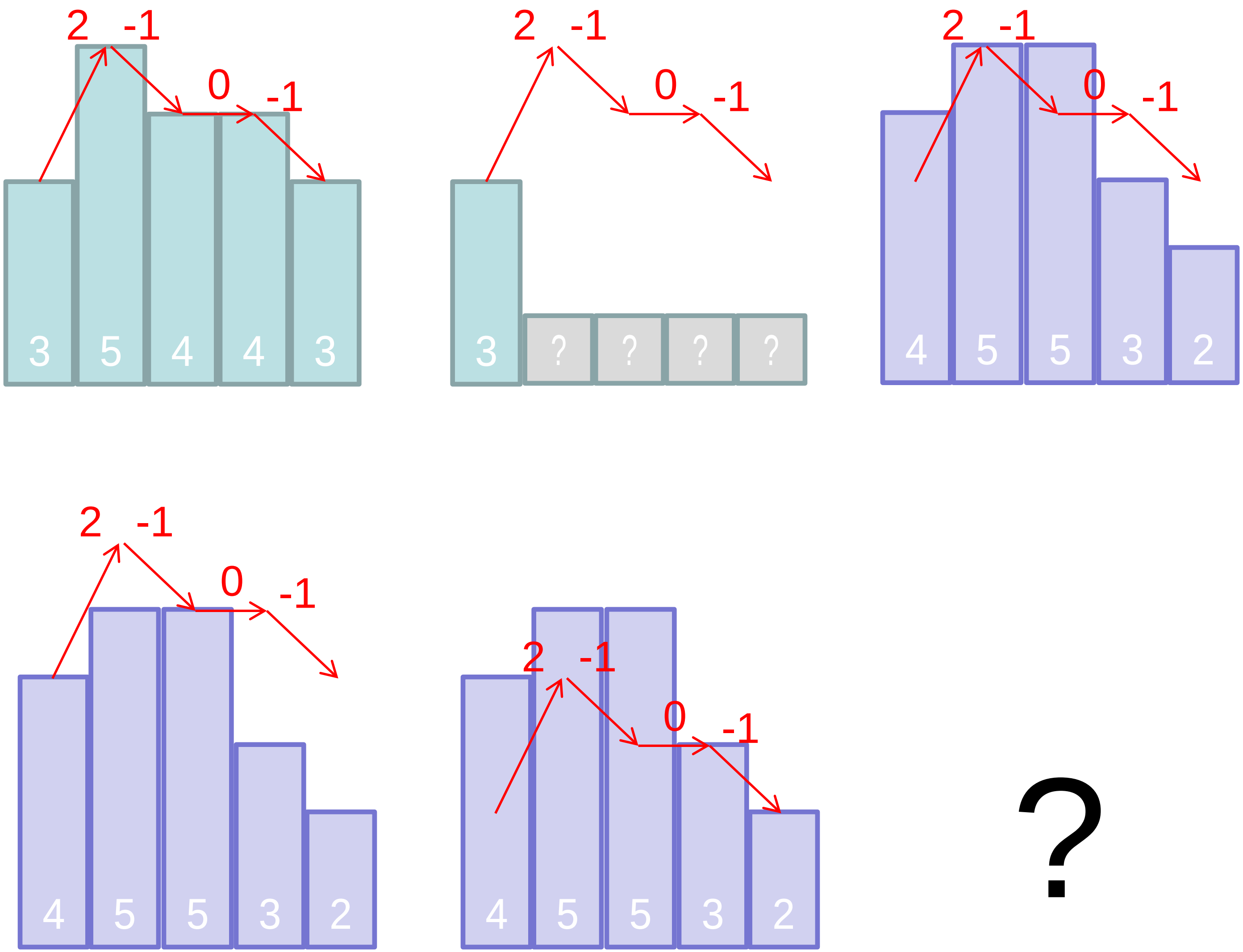
Project 4

- Poisson image editing
- Image segmentation using graph cut optimization

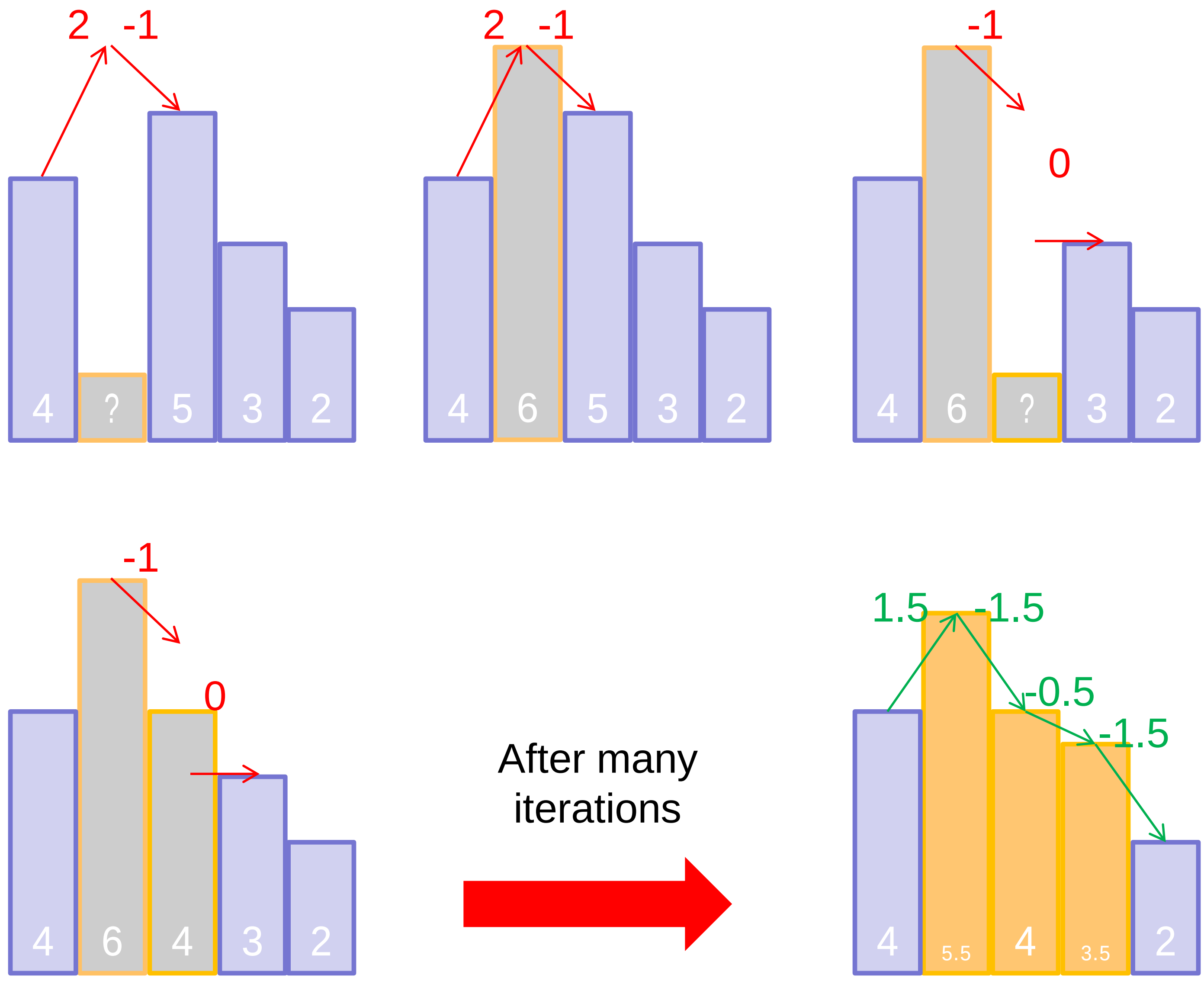
Poisson image editing

- Implement a Poisson solver
- Demonstrate using different scenarios
  - Seamless cloning
  - Combination of images
  - Highlight removal (gamma compression)

Poisson Image Editing



Poisson Image Editing



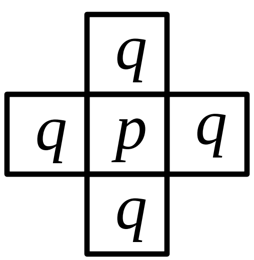
Reminder

- Goal: find image f such that

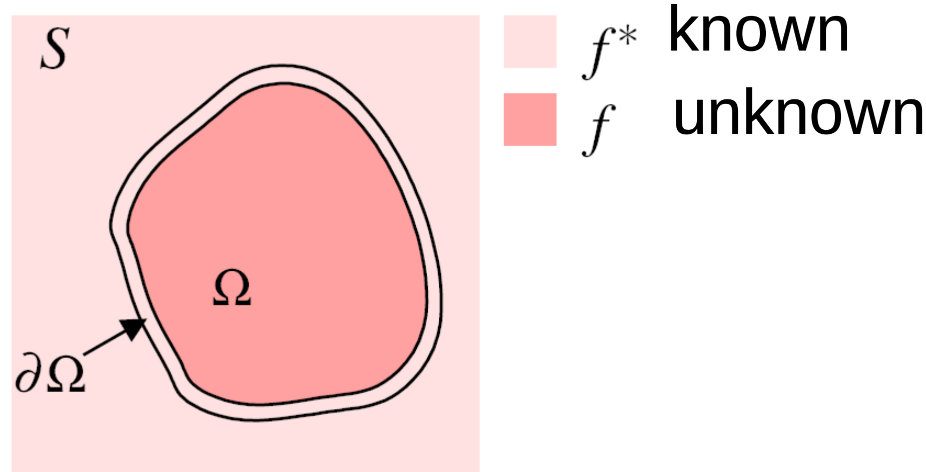
$$\min_f \iint_\Omega |\nabla f - \mathbf{v}|^2 \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

- Linear least squares problem

$$\min_{f|_\Omega} \sum_{\langle p,q\rangle \cap \Omega \neq \emptyset} (\underbrace{f_p - f_q}_{x \text{ or } y \text{ component of discrete gradient}} - \underbrace{v_{pq}}_{x \text{ or } y \text{ component of guidance fields}})^2, \text{ with } f_p = f_p^*, \text{for all } p \in \partial\Omega$$



Pixel neighborhood

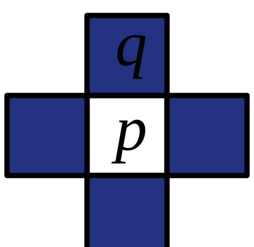


Gauss-Seidel solver

- Loop many times
  - Loop over all unknown pixels p
    - At each pixel, solve for pixel value f_p using current values for neighboring pixels f_q

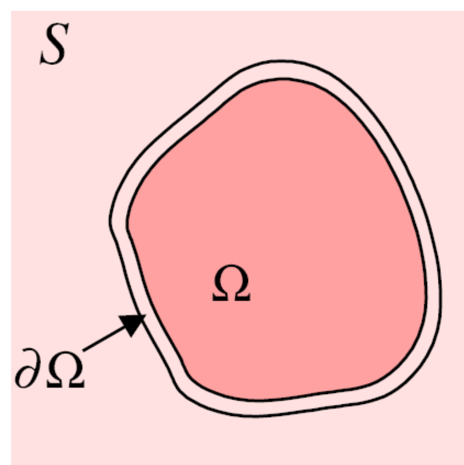
$$|N_p| f_p - \sum_{q \in N_p \cap \Omega} f_q = \sum_{q \in N_p \cap \partial\Omega} f_q^* + \sum_{q \in N_p} v_{pq}$$

- Pixel neighborhood



$$\boxed{q} \in N_p$$

$$|N_p| = 4$$



f^* known

f unknown

Gauss-Seidel solver

- Loop many times
  - Loop over all unknown pixels p
    - At each pixel, solve for pixel value f_p using current values for neighboring pixels f_q

$$|N_p| \boxed{f_p} - \sum_{q \in N_p \cap \Omega} \boxed{f_q} = \sum_{q \in N_p \cap \partial\Omega} f_q^* + \sum_{q \in N_p} v_{pq}$$

- Thousand(s) of iterations necessary for convergence on large images
  -> very slow
- Test with small images!

Speed up Gauss-Seidel solver

- Work only on affected pixels

```
% Find affected pixels
[I,J] = find(mask(:,:,1)==0);
for it=1:nIter
    % Work only on found pixels
    for k=1:length(I)
        i = I(k);
        j = J(k);
        out(i,j) = ...
    end
end
```

Speed up Gauss-Seidel solver

- Downsampling

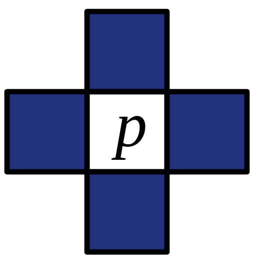
```
% Half of the steps “down sampled”
nIter = ceil(nIter/2);
% downsampling for the first half
outDS = imresize(out, 0.5);
maskDS = imresize(mask, 0.5);
[vxDS vyDS] = imageGradient(imresize(source, 0.5));
% Do Gauss-Seidel itarations
for k=1:nIter
    ...
end

% upsampling for the second half.
out = imresize(outDS, 2);
% Do Gauss-Seidel iterations
for k=1:nIter
    ...
end
```

Implementation detail

- v_{pq} includes sign according to order of pq!
- Desired gradient field stored in vx, vy

$$\sum_{q \in N_p} v_{pq} = vy(i-1,j)-vy(i,j)+vx(i,j-1)-vx(i,j))$$



- Test implementation by reconstructing image wihtout modifying gradient

Gradient

- Matlab provides a function to calculate gradients: `Gradient(A)`. **Dont use it!**
- Use instead `imfilter(A, [-1 1], 'same')`
- Or `A(:,2:end)-A(:,1:end-1)`

Gradient

- `Gradient(A)` vs. `A(:,2:end)-A(:,1:end-1)`

```
A =

    96    15    97    95    48    80    14    42    91    79

>> A(2:10)-A(1:9)

ans =

   -81    82    -2   -47    32   -66    28    49   -12
```

$$f'(p) = f(p+1) - f(p)$$

Gradient

- `Gradient(A)` vs. `A(:,2:end)-A(:,1:end-1)`

```
A =

    96    15    97    95    48    80    14    42    91    79

>> A(2:10)-A(1:9)

ans =

   -81    82    -2   -47    32   -66    28    49   -12

>> gradient(A)

ans =

  -81.0   0.5  40.0 -24.5  -7.5 -17.0 -19.0  38.5  18.5 -12.0
```

$$f'(p) = \frac{1}{2}([f(p) - f(p-1)] + [f(p+1) - f(p)]) = \frac{1}{2}(f(p+1) - f(p-1))$$

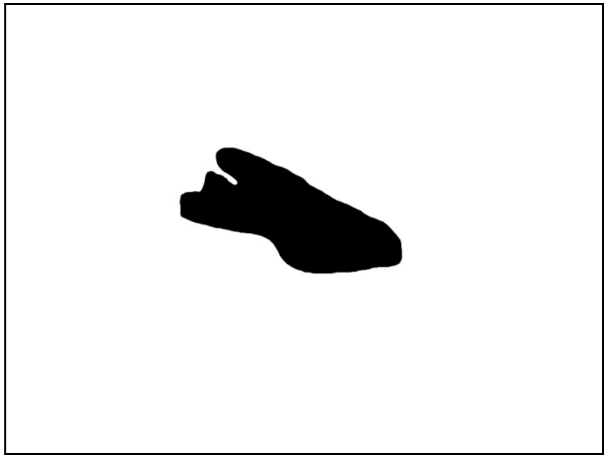
Gradient(A)

A(:,2:end)-A(:,1:end-1)

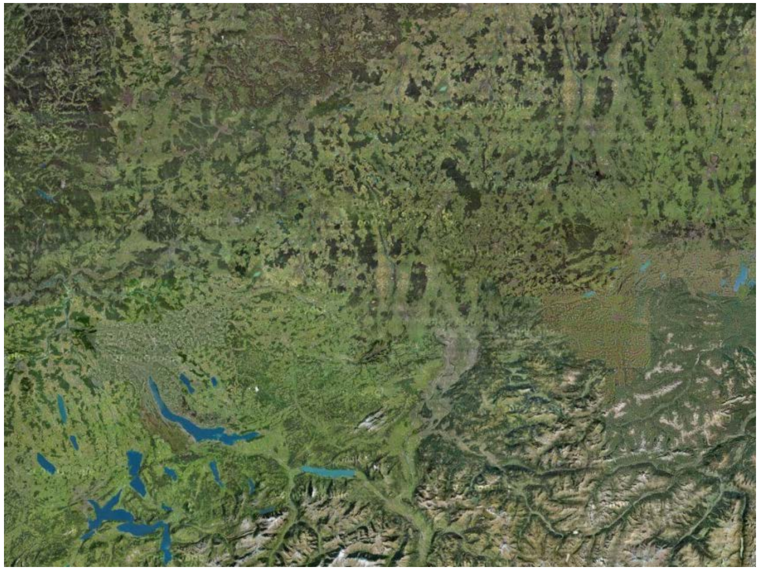
Seamless cloning

Target

Source



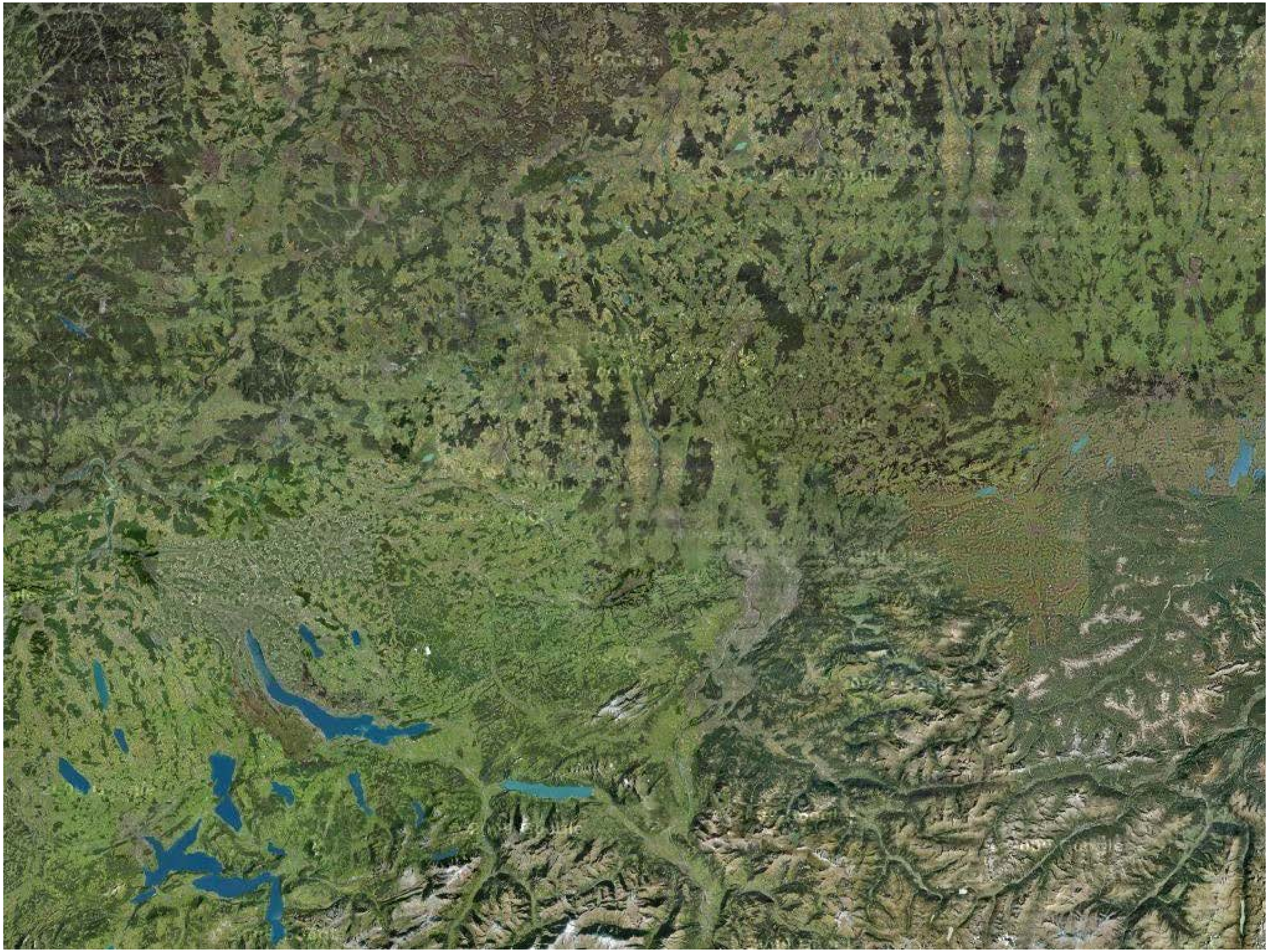
Mask



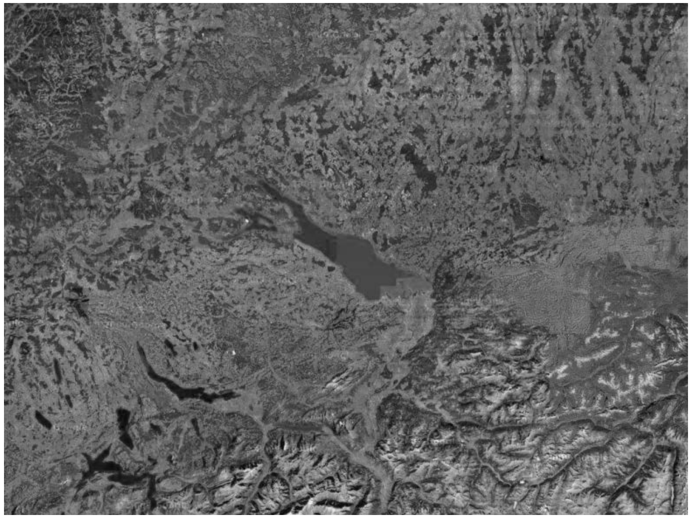
Output

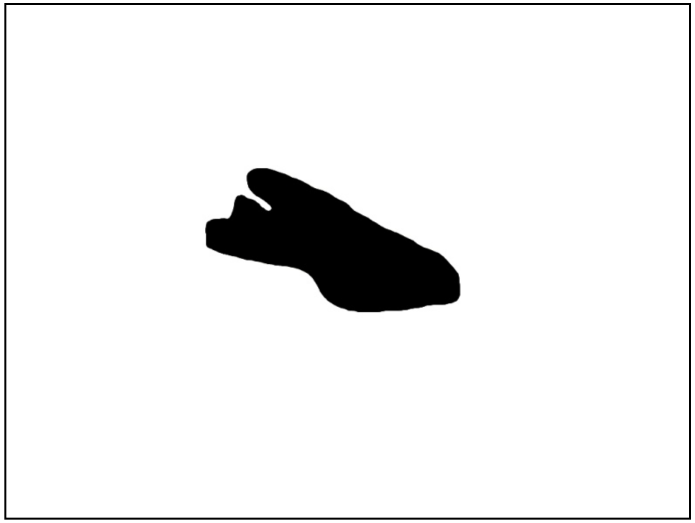
Seamless cloning

Seamless cloning

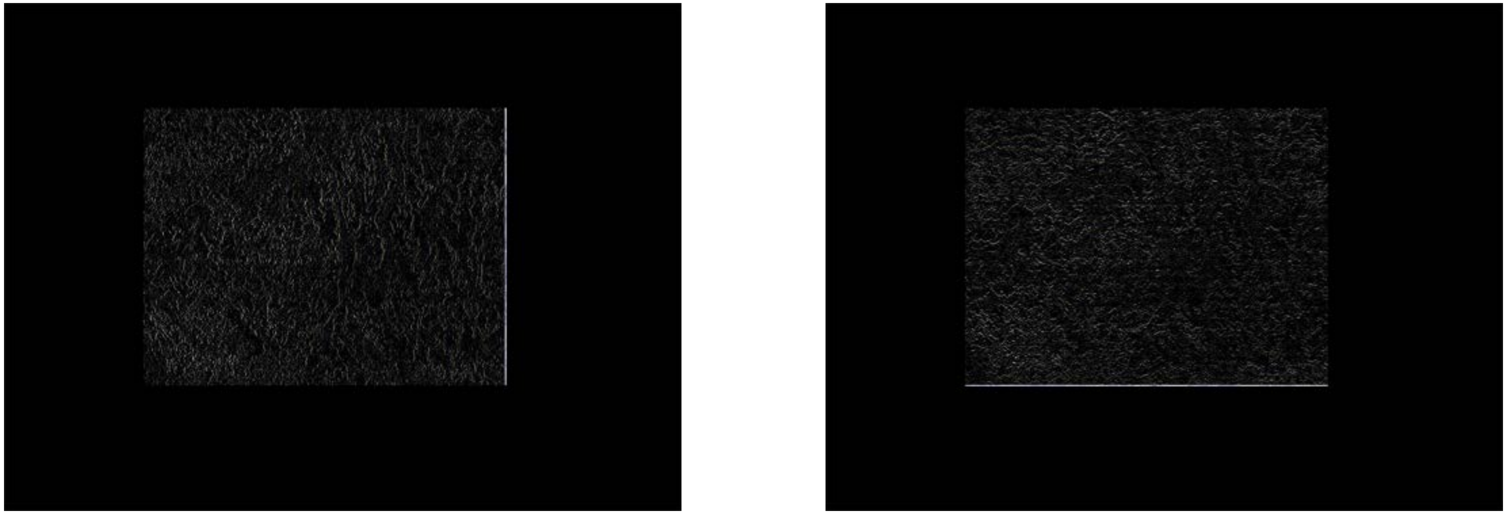


Input to Poisson solver





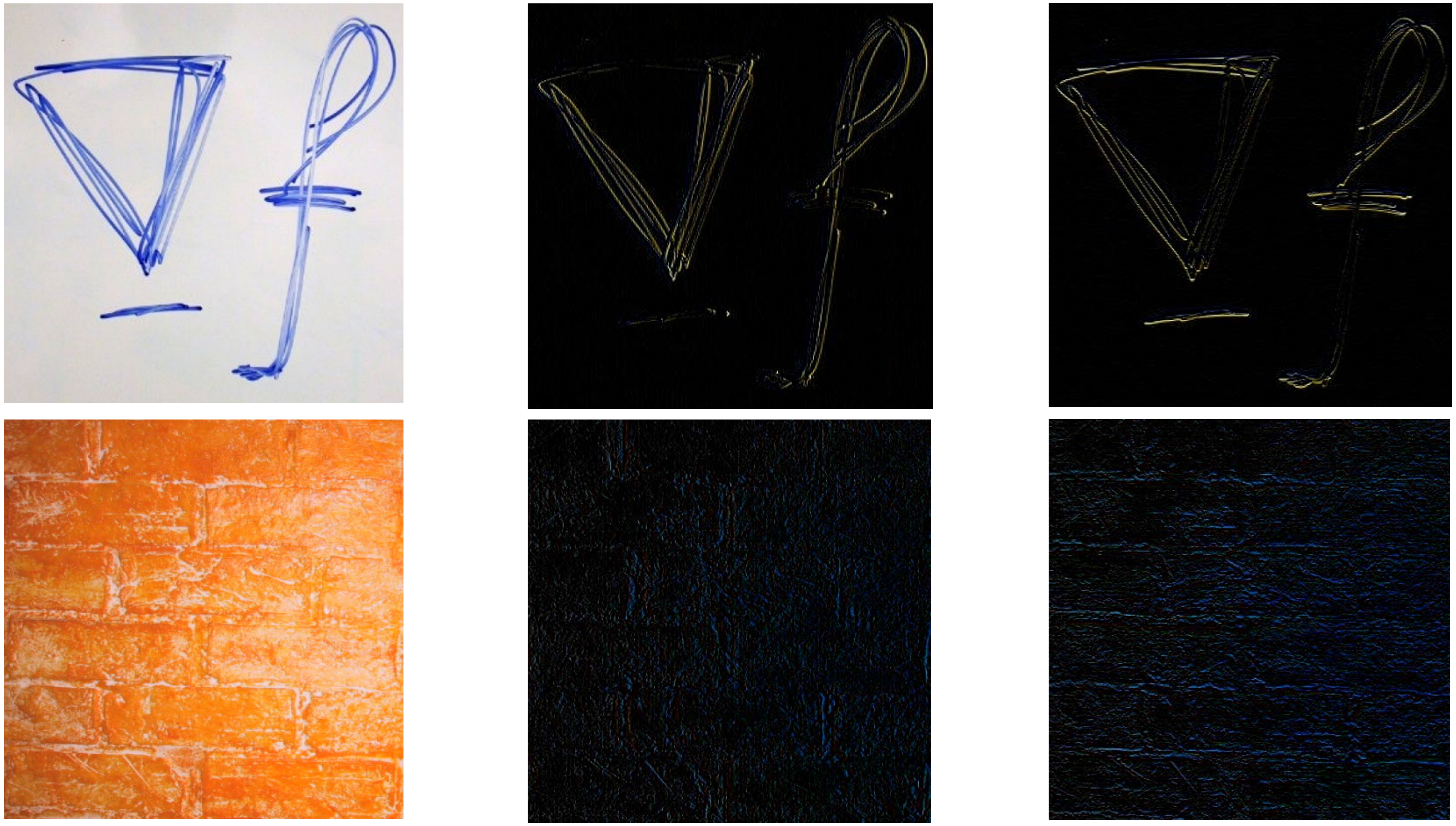
Target (boundary cond.) Mask



x and y components of desired gradient taken from source

Gradient mixing

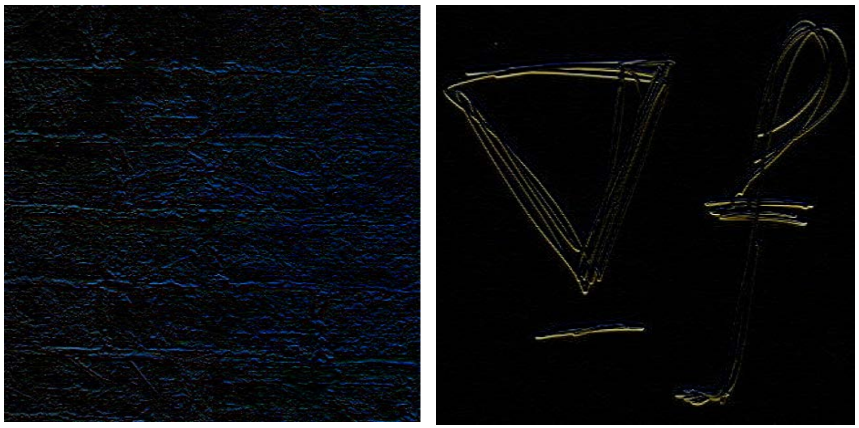
- Desired gradient is constructed mixing gradients of two images



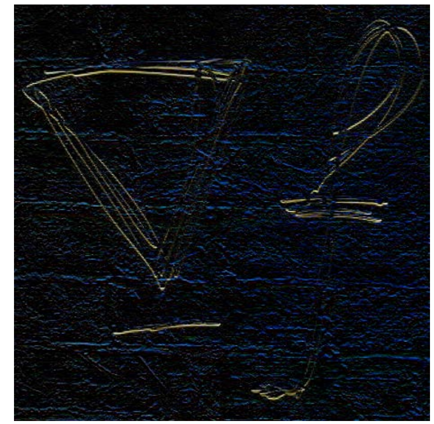
Input images | Gradient x comp. | Gradient y comp.

Gradient mixing using maximum

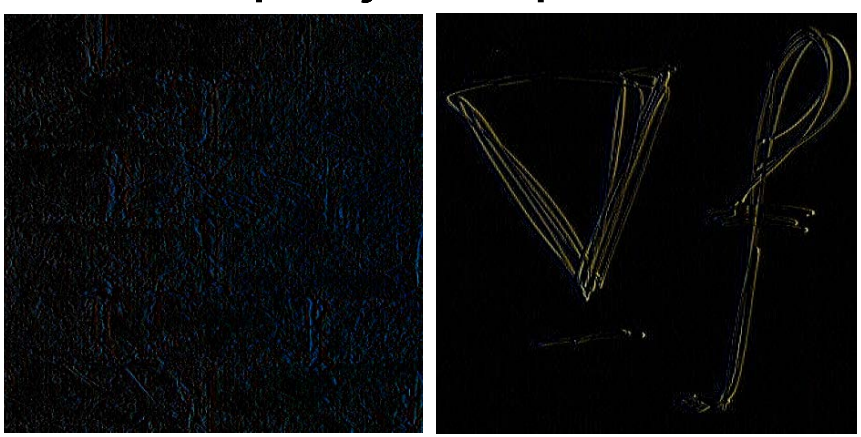
- At each pixel, desired x,y gradient components contain larger of two values from input images



Input y comp.



Desired y comp.



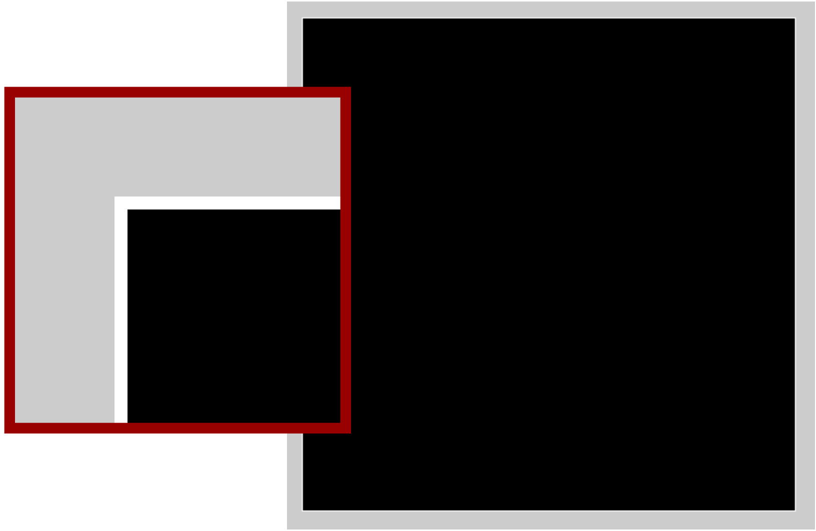
Input x comp.

Desired x comp.

Input to poisson solver

Target

Desired y comp.



Mask with 1-pixel boundary

Desired x comp.

Result

Input images

Result

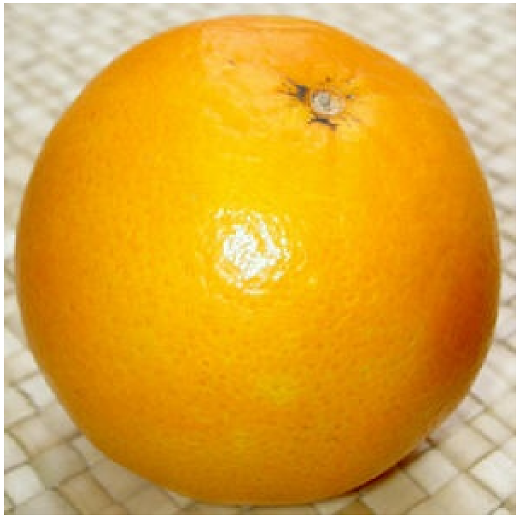
Gradient compression

- Desired gradient by gamma compression
- Use gradient ∇f^* of logarithm of input image
  - Weber‘s law: brightness perception is logarithmic
    http://en.wikipedia.org/wiki/Weber%E2%80%93Fechner_law
- Desired gradient $\mathbf{v}$

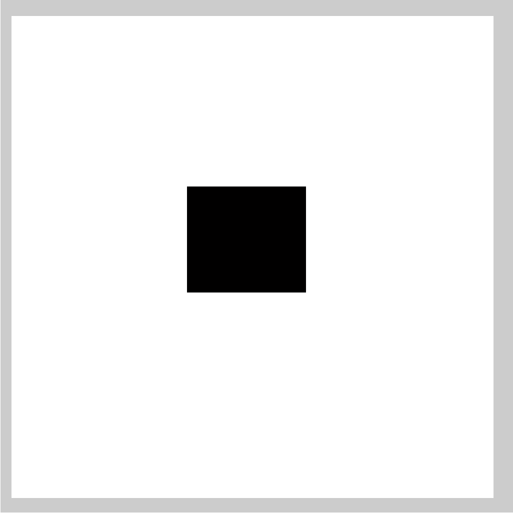
$$\mathbf{v} = \alpha^{\beta} |\nabla f^*|^{-\beta} \nabla f^*$$

- α: small constant (~0.005) times average gradient magnitude
- β: ~0.4

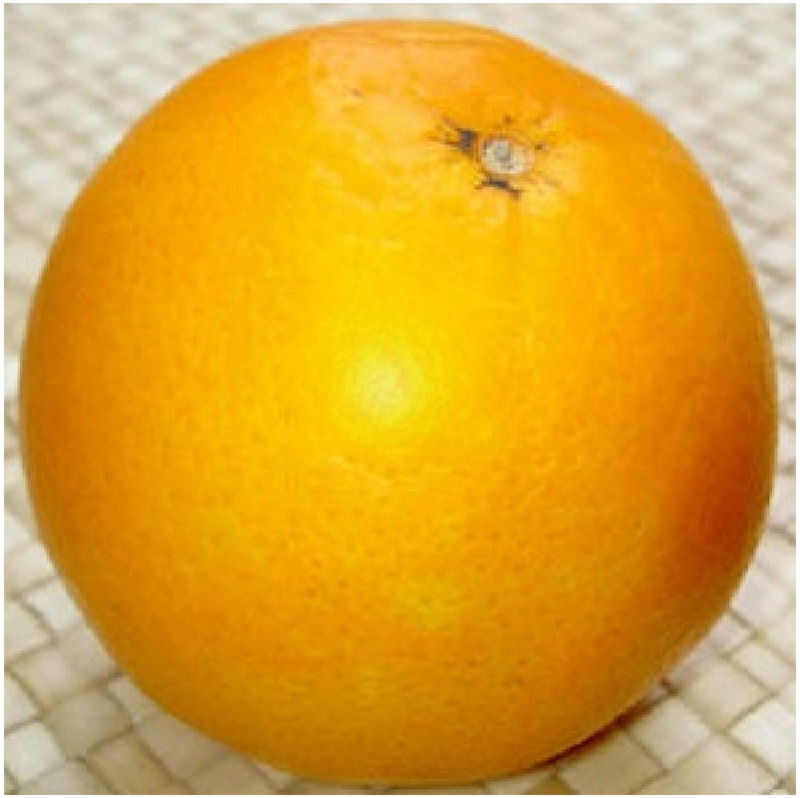
Result



Target



Mask



Output

Project 4

- Poisson image editing
- **Image segmentation using graph cut optimization**

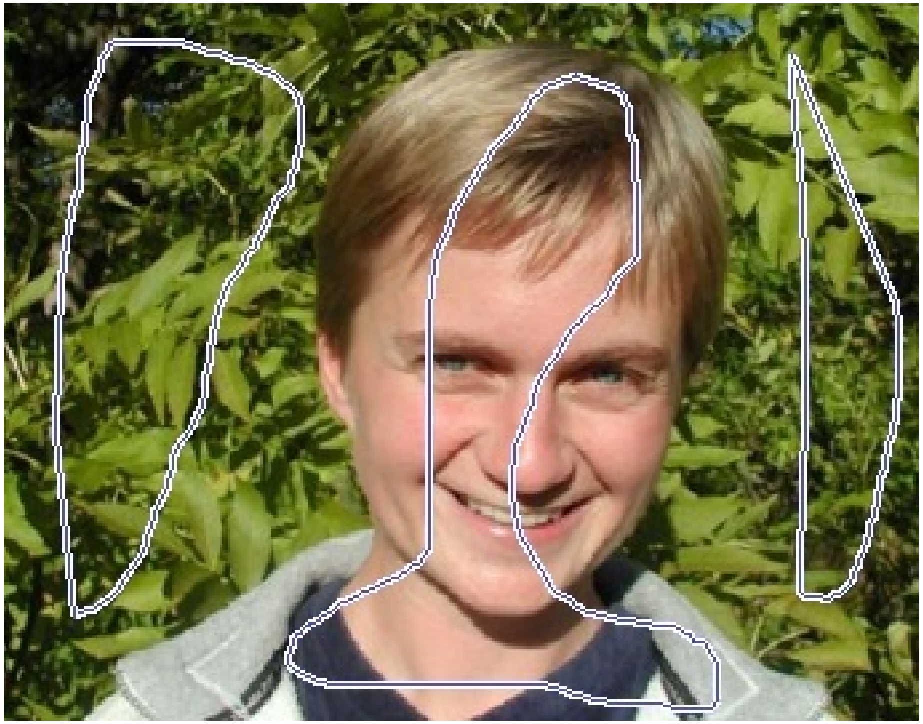
Interactive image segmentation

- Four steps
  1. Interactive foreground and background labeling
  2. Construct color model
  3. Set up data and smoothness terms, graph connectivity
  4. Solve optimization using graph cuts

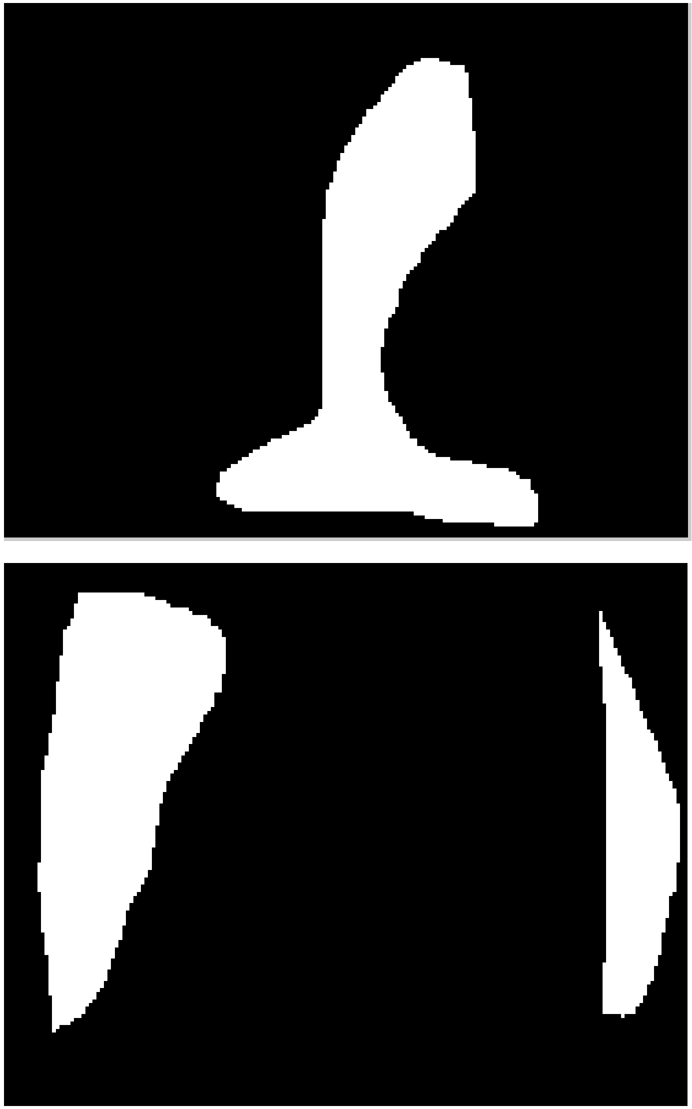
Interactive labeling

- Use `imfreehand` to draw foreground and background masks on images
- Get interactive keyboard input using `input` to select foreground or background
- Store in binary images
- Visualize the results

Interactive labeling



User provided masks



Binary representation

Color model

- Extract labeled foreground and background pixels
- Input image `img`, binary masks `fmask`, `bmask`

```
% get all pixel colors in a 3 x (#pixels) array
colors = reshape(shiftdim(img,2),3,size(img,1)*size(img,2));
% get all pixel coordinates in a 2 x (#pixels) array
x = zeros(size(img,1),size(img,2),2);
[x(:,:,1) x(:,:,2)] = meshgrid(1:size(img,2),1:size(img,1));
x = reshape(shiftdim(x,2),2,size(img,1)*size(img,2));
% get the foreground and background masks in a 1 x (#pixels) array
fmask = reshape(fmask,1,size(img,1)*size(img,2));
bmask = reshape(bmask,1,size(img,1)*size(img,2));
% get the indices of the foreground and background pixels
[tt ttt findices] = intersect(x', (x.*[fmask; fmask])', 'rows');
[tt ttt bindices] = intersect(x', (x.*[bmask; bmask])', 'rows');
% extract only labeled foreground and background pixels
fcolors = colors(:,findices);
bcolors = colors(:,bindices);
```

Color model

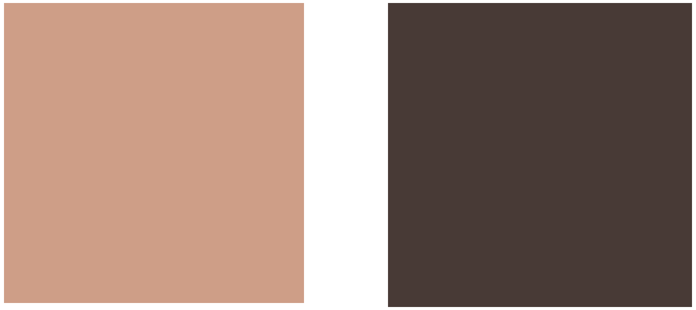
- Use `gmdistribution` to obtain Gaussian mixture model
  - Two components should be sufficient
- Obtain the estimated probability distribution using `pdf`
  - Type `help gmdistribution/pdf`

Color model

Background probabilities

Foreground probabilities

Means of bgr. components



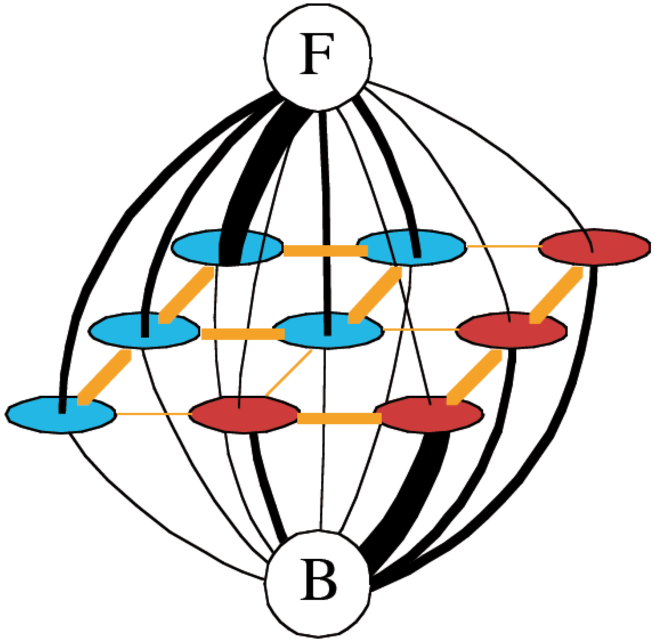
Means of fgr. components

Graph cut

- Obtain graph cut code from
  http://vision.ucla.edu/~brian/gcmex.html
- Solve graph cut using
  `GCMex(class, unary, pairwise, labelcost, 0)`
- Study example for how to use it
- Assume: image with W×H = N pixels

Parameters

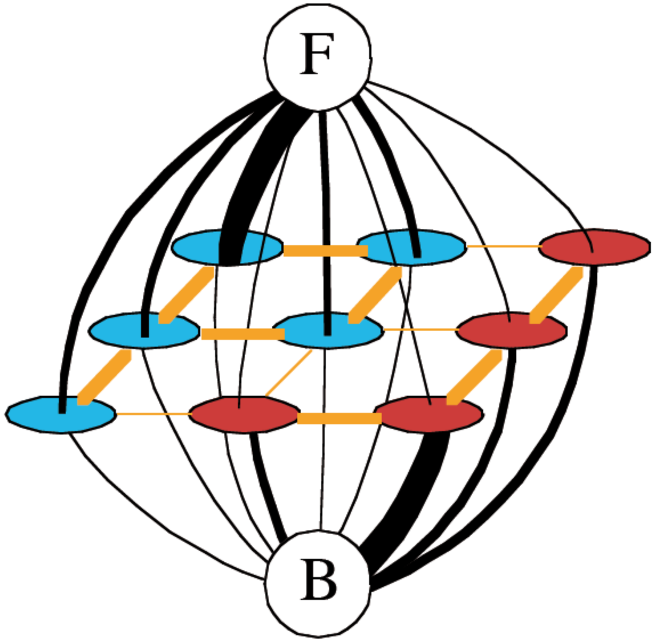
- `class`
  - Initial labeling (red/blue in the graph)
  - 1×N matrix (N = W×H)
  - All zeros



- `unary`
  - Correlation between pixel and labels (data term, black lines in the graph)
  - With 2 labels, 2×N matrix
  - Fill in with data gained from color model and user provided masks

Parameters

- `pairwise`

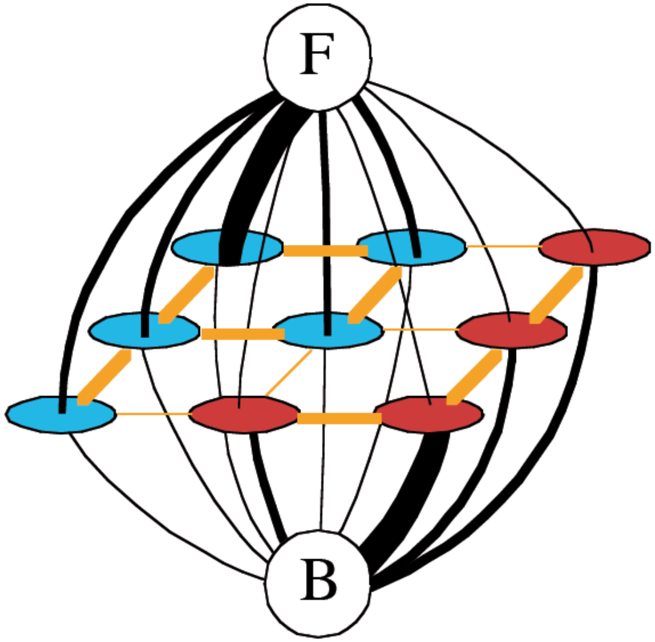


  - Smoothness term (part of orange lines)
  - Stored in sparse N×N matrix
  - Max. 8 non-zero entries per row
    - Because each pixel has at most 8 neighbors
  - Construct sparse matrix using `sparse`
    - Directly writing into matrix is extremely slow
    - Construct index arrays first, then pass to `sparse(rowindices, colindices, values, N, N);`

Pairwise

- Each matrix entry is $e^{-\beta\|c_p - c_q\|^2}$
- Neighboring pixels p, q, pixel colors c_p, c_q
- Visualization by summing over columns

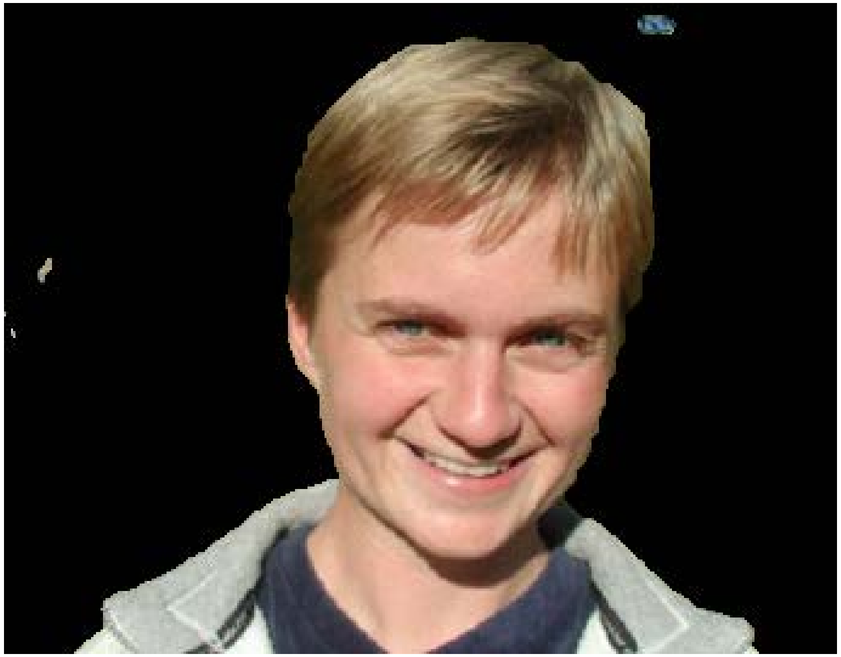
Parameters



- `labelcost`
  - 2×2 matrix for binary labels
  - Values depend of labels of interacting pixels
  - Values are multiplied with matrix entries in `pairwise` to get smoothness term
  - Use `[0 1; 1 0]`

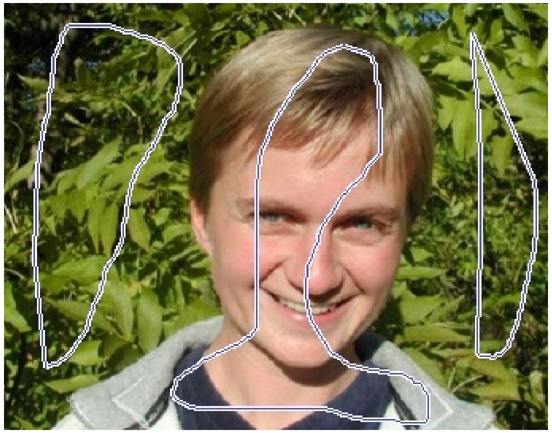
Result

Background



Foreground

Binary labels

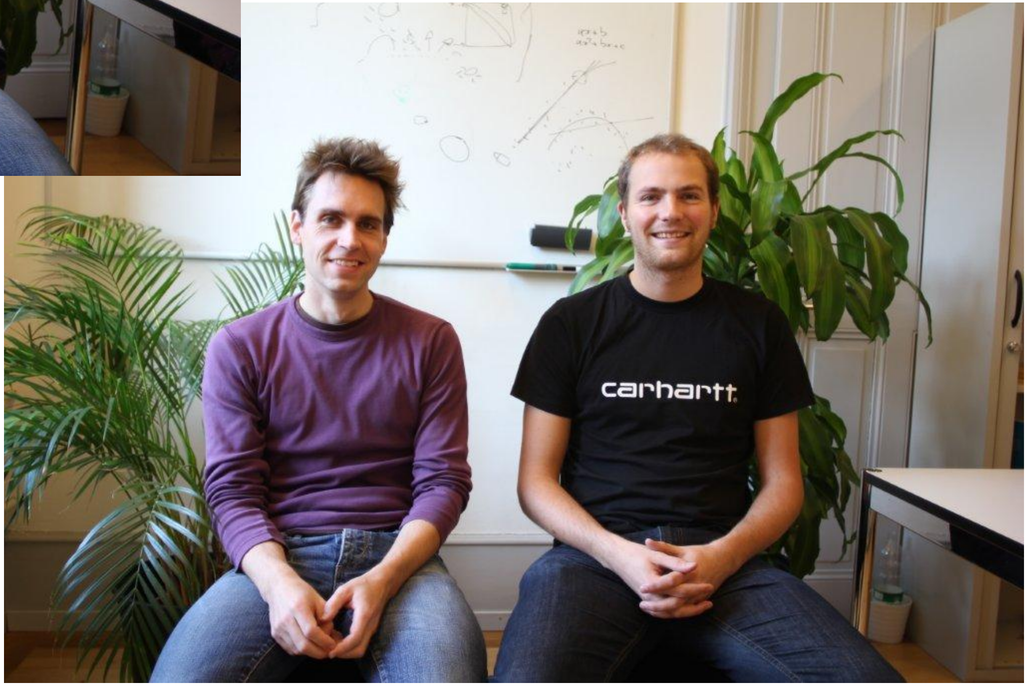


Input masks

Bonus

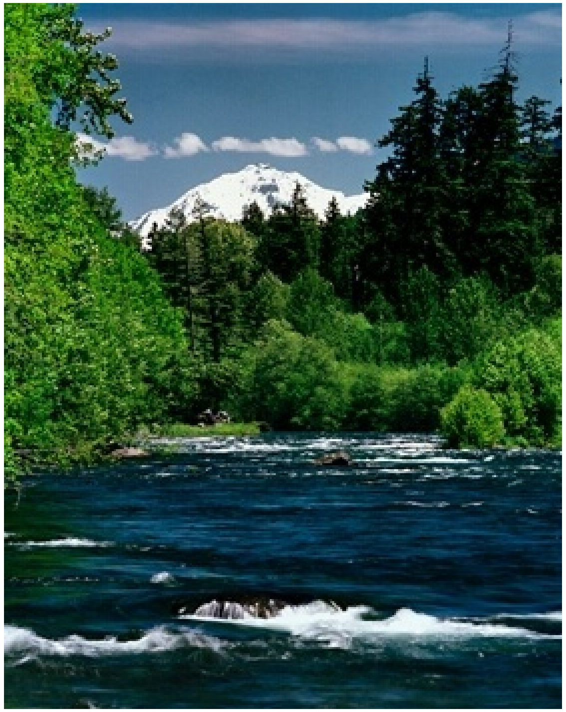
- Interactive digital photomontage
- Only 2 photos
- Combination of the pervious assignements
- New data and smoothness terms for the graph cut
- See project webpage for the original paper and a C++ implementation
  http://grail.cs.washington.edu/projects/photomontage/

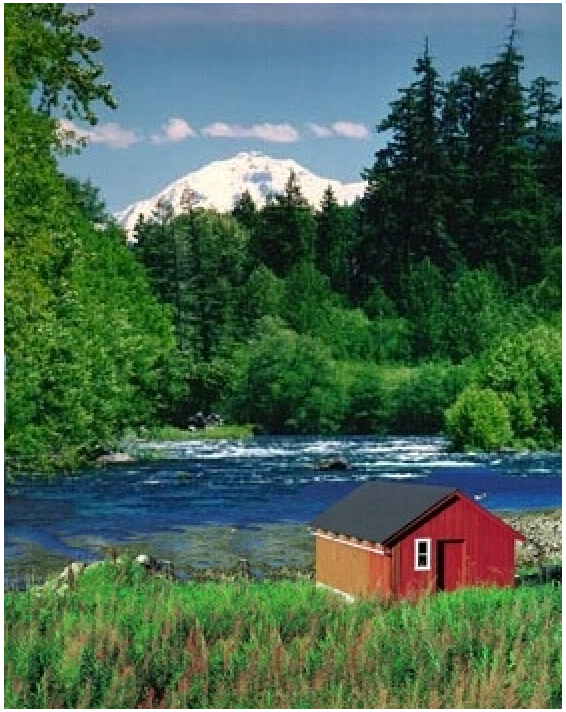
Bonus



Bonus

Bonus





Bonus