
Image Composition and Editing
Kari Pulli
Senior Director
NVIDIA Research
NVIDIA

Digital Photomontage

Agarwala SIGGRAPH 2004

shoot until everybody has smiled at least once

Digital Photomontage

Agarwala SIGGRAPH 2004

shoot until everybody has smiled at least once

Segment

Assemble

Topics today

- Segmentation
  - Graph Cut
  - GrabCut
  - oversegmentation with Mean-Shift, followed by clustering
- Poisson blending
- Seam finding for Poisson blending
- Color transfer
  - helps blending
  - helps combining blurry – noisy image pairs
- TouchTone
  - interactive tonemapping

Microsoft Research Cambridge

SIGGRAPH2004

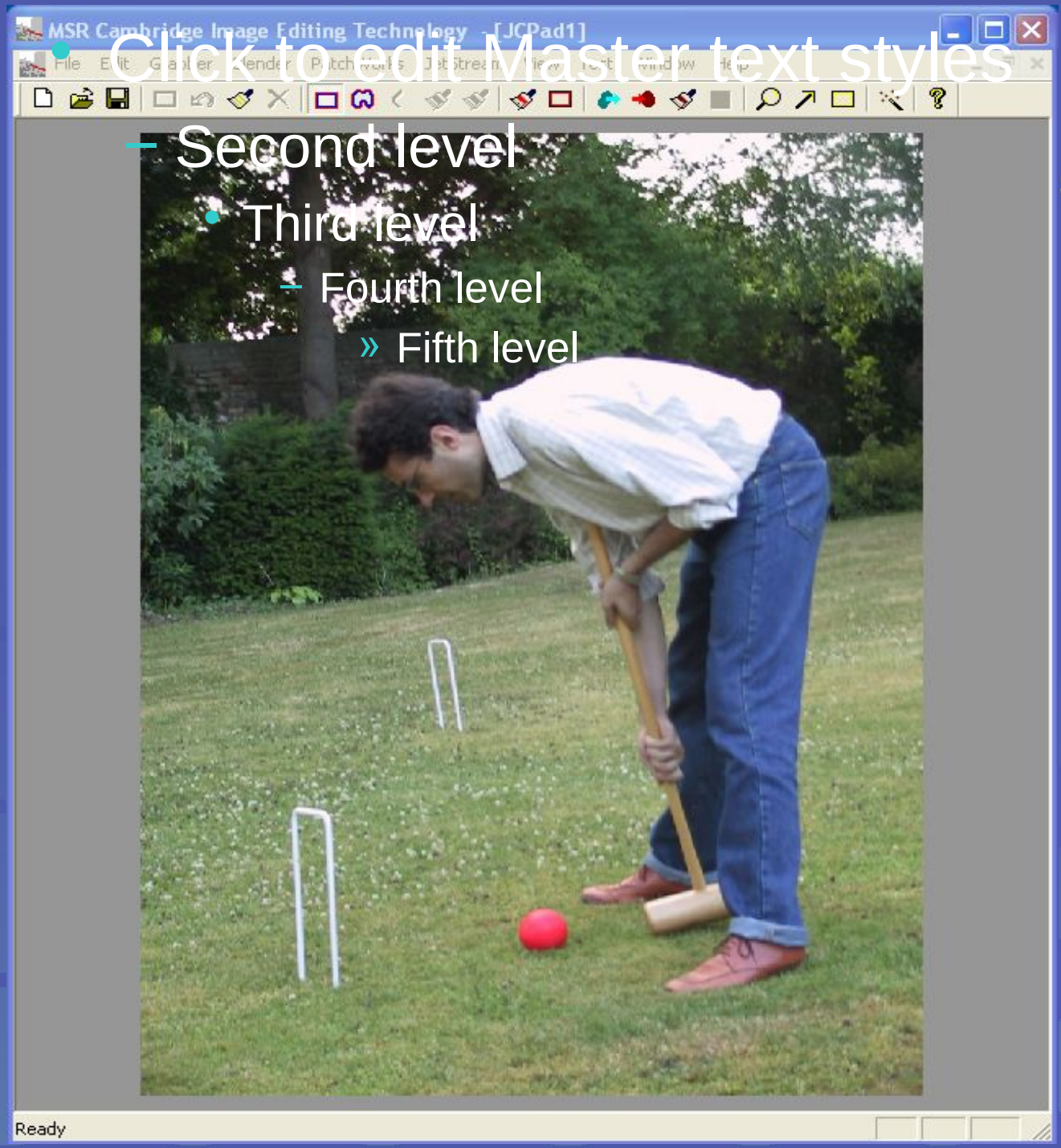
GrabCut

Interactive Foreground Extraction using Iterated Graph Cuts

Carsten Rother
Vladimir Kolmogorov
Andrew Blake

Microsoft Research Cambridge-UK

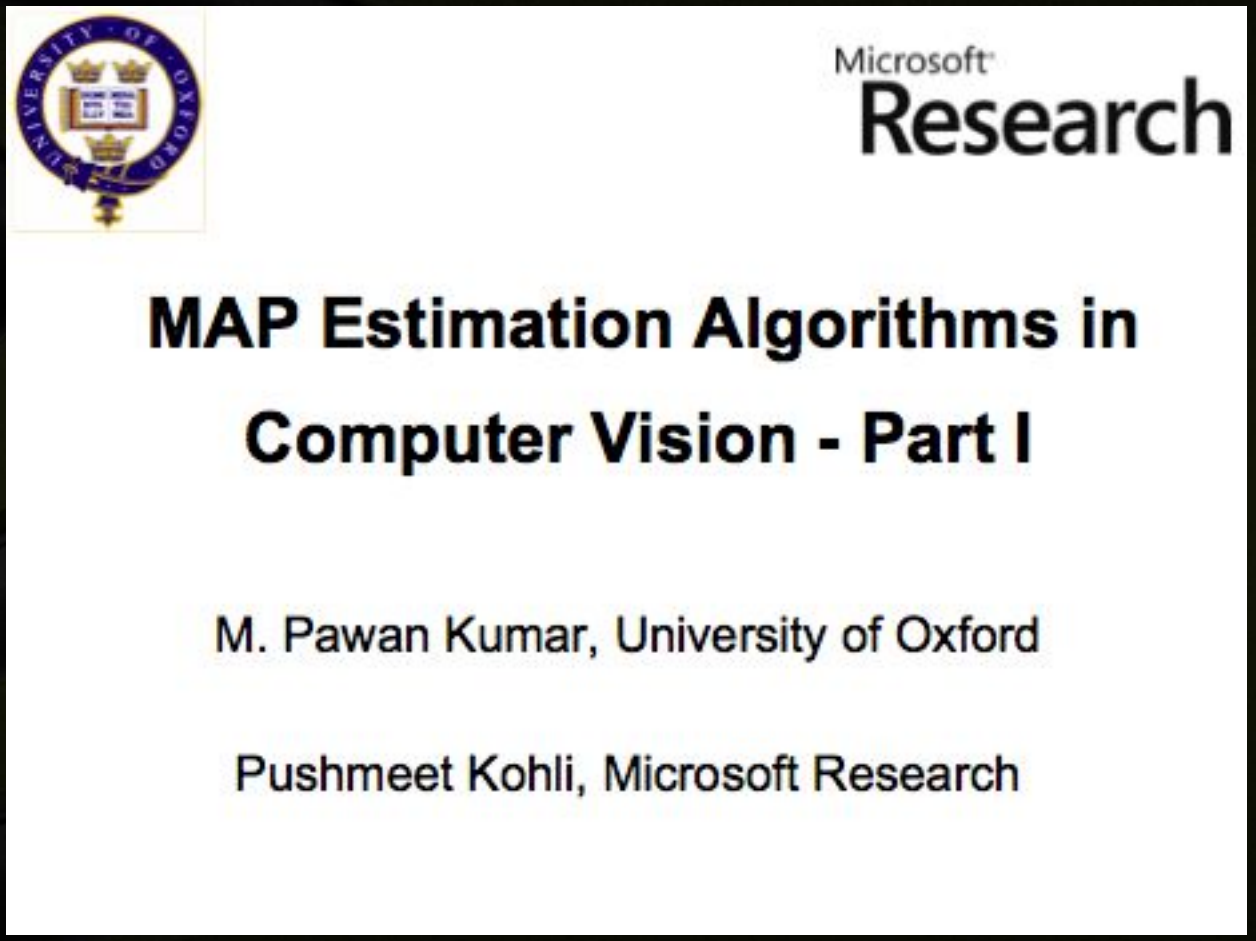
Photomontage



SIGGRAPH2004

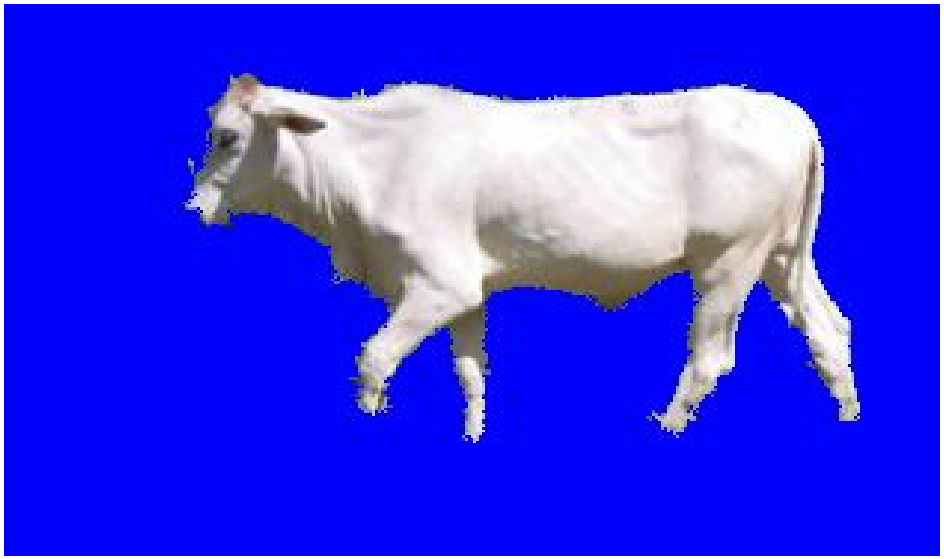
The following is a small part of

- http://research.microsoft.com/~pkohli/papers/Tutorial_Part1.ppt
- http://research.microsoft.com/~pkohli/papers/Tutorial_Part2.ppt



A Vision Application

Binary Image Segmentation



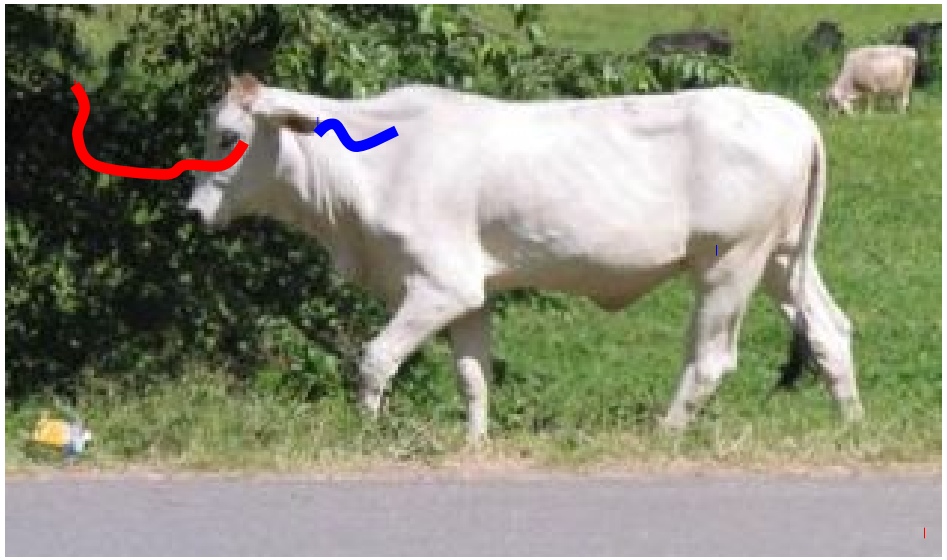
How ?

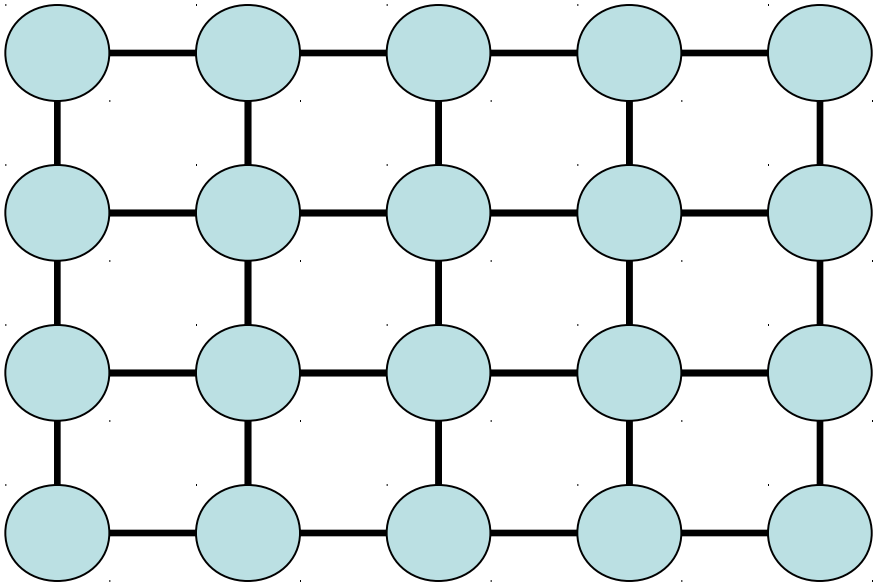
Cost function Models *our* knowledge about natural images

Optimize cost function to obtain the segmentation

A Vision Application

Binary Image Segmentation





Object - white, Background - green/grey

Graph G = (V,E)

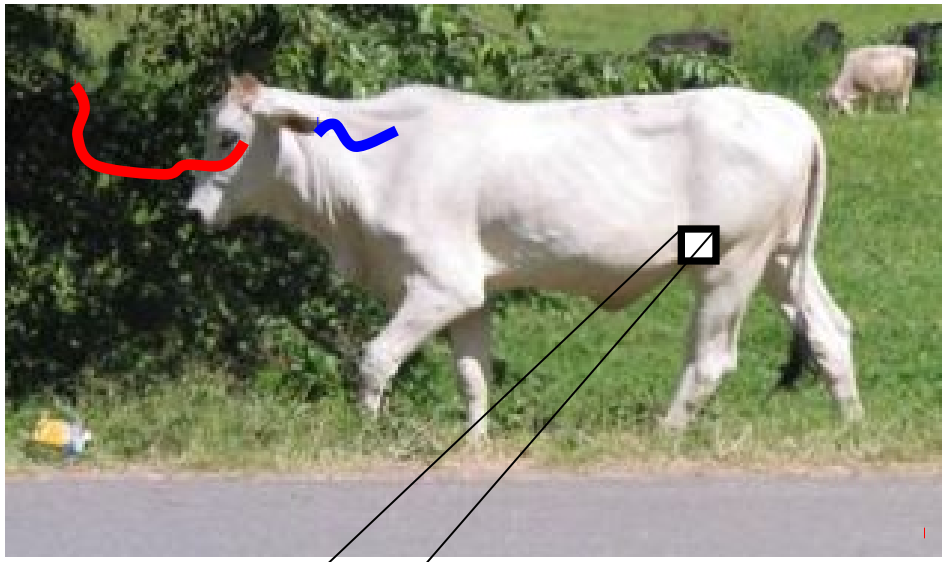
Each vertex corresponds to a pixel

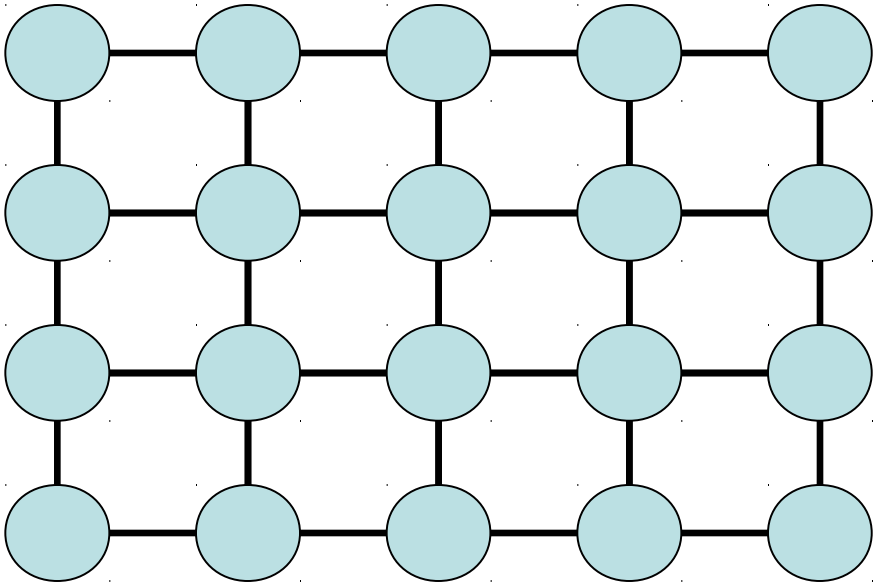
Edges define a 4-neighbourhood *grid* graph

Assign a label to each vertex from L = {obj,bkg}

A Vision Application

Binary Image Segmentation





Object - white, Background - green/grey

Graph G = (V,E)

Cost of a labelling f : V □ L

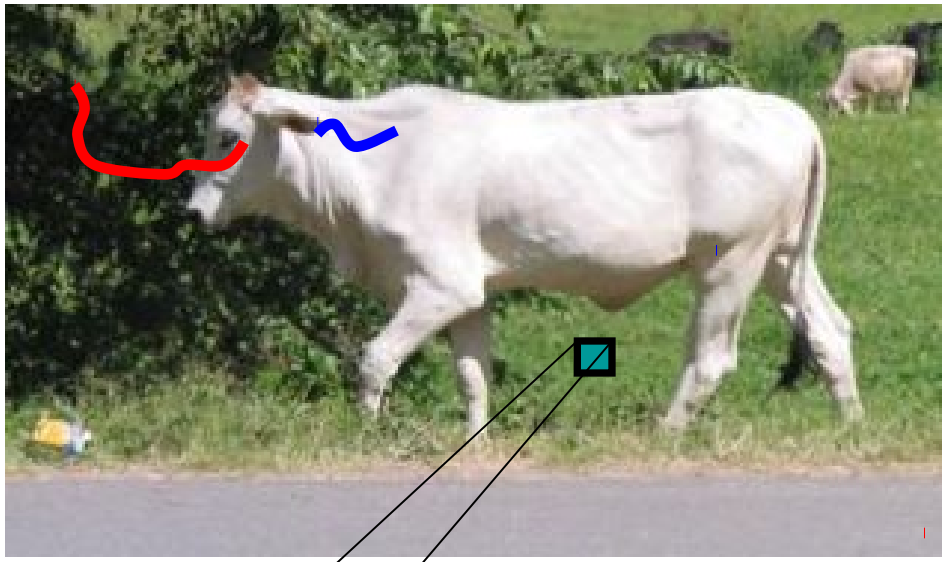
Per Vertex Cost

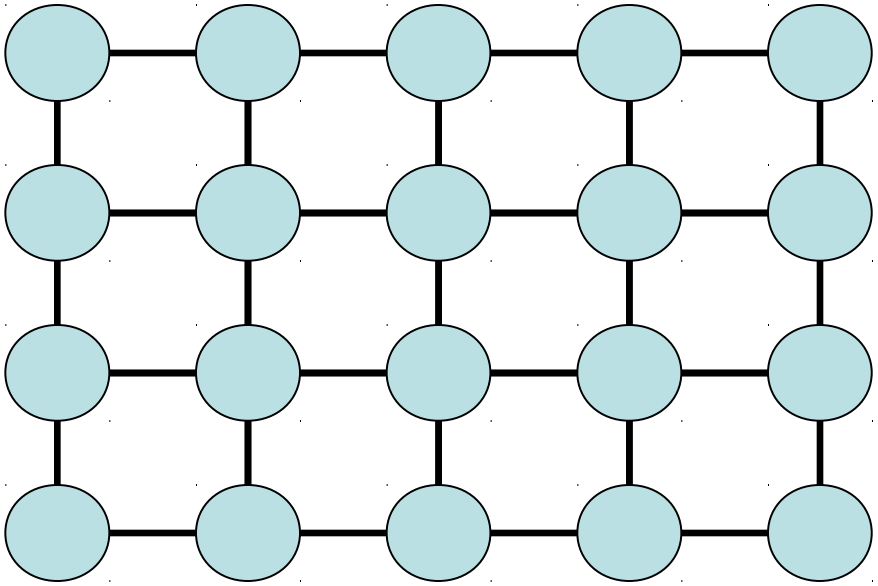
Cost of label ‘obj’ low

Cost of label ‘bkg’ high

A Vision Application

Binary Image Segmentation





Object - white, Background - green/grey

Graph G = (V,E)

Cost of a labelling f : V L

Per Vertex Cost

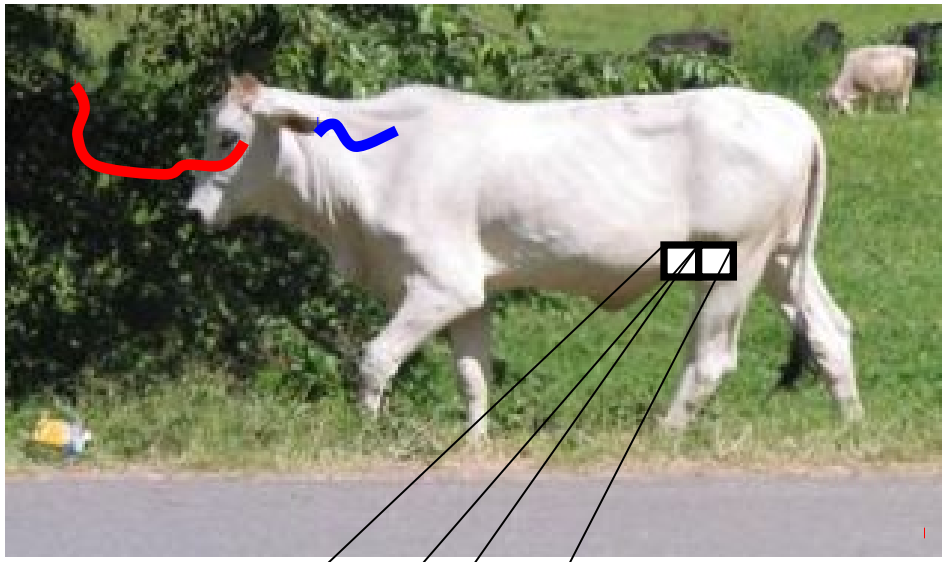
Cost of label ‘obj’ high

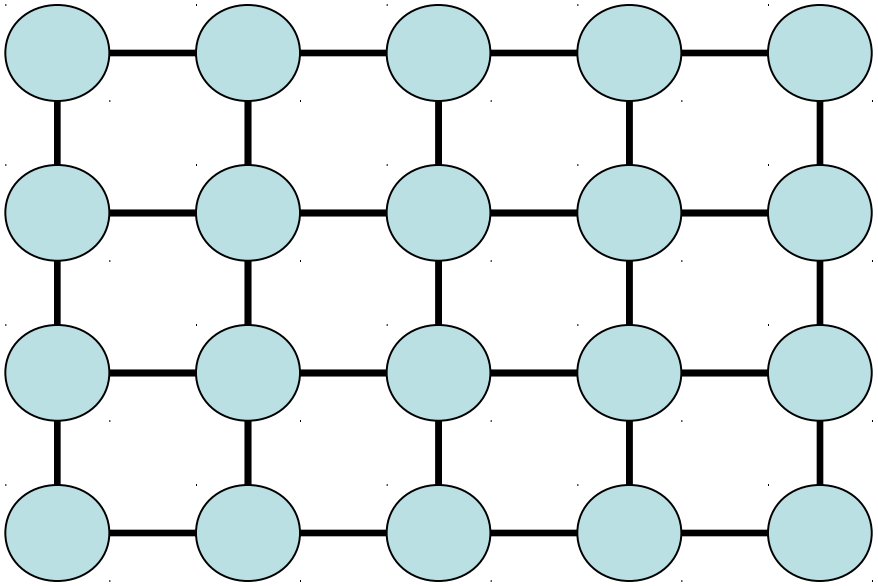
Cost of label ‘bkg’ low

UNARY COST

A Vision Application

Binary Image Segmentation



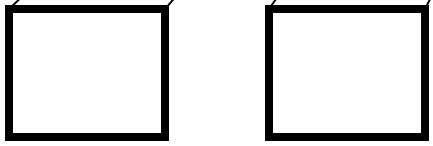


Object - white, Background - green/grey

Graph G = (V,E)

Cost of a labelling f : V L

Per Edge Cost

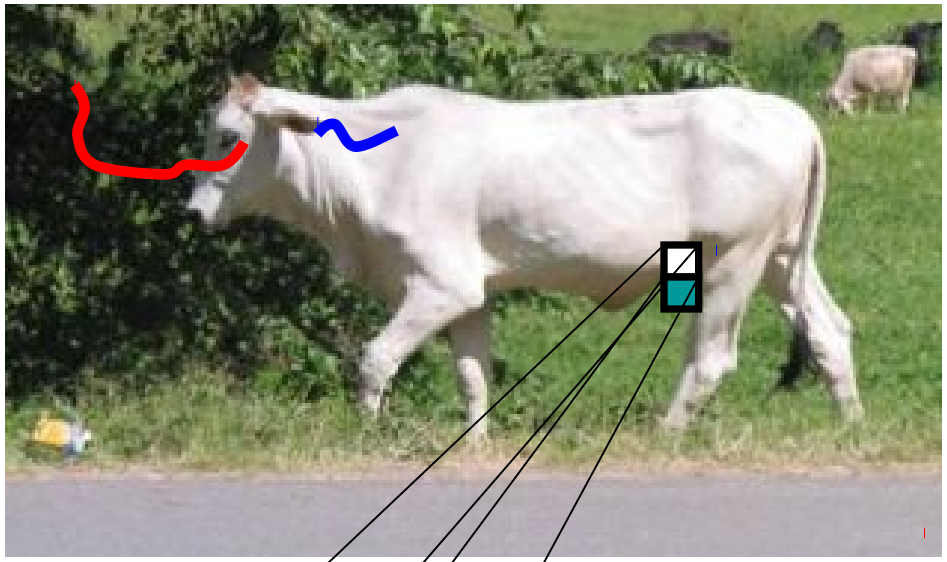


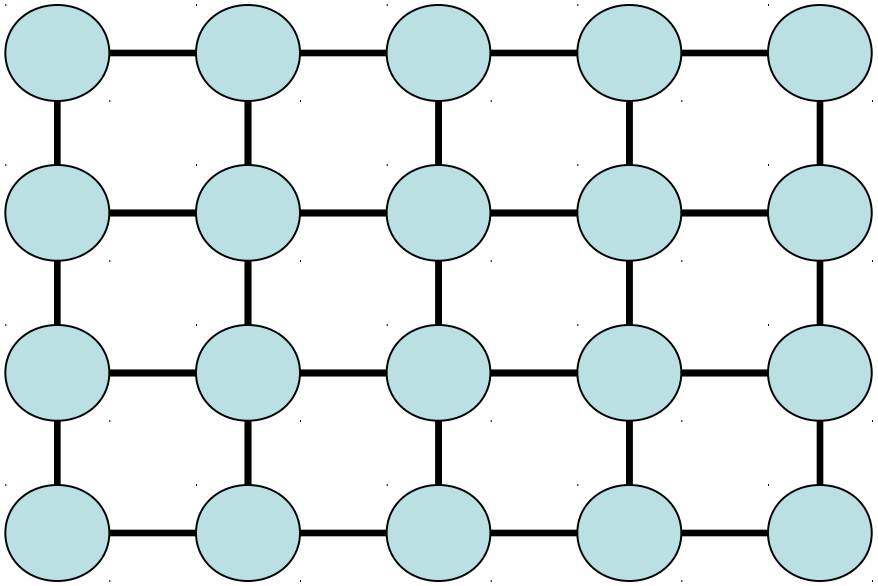
Cost of same label low

Cost of different labels high

A Vision Application

Binary Image Segmentation





Object - white, Background - green/grey

Graph G = (V,E)

Cost of a labelling f : V L

Per Edge Cost

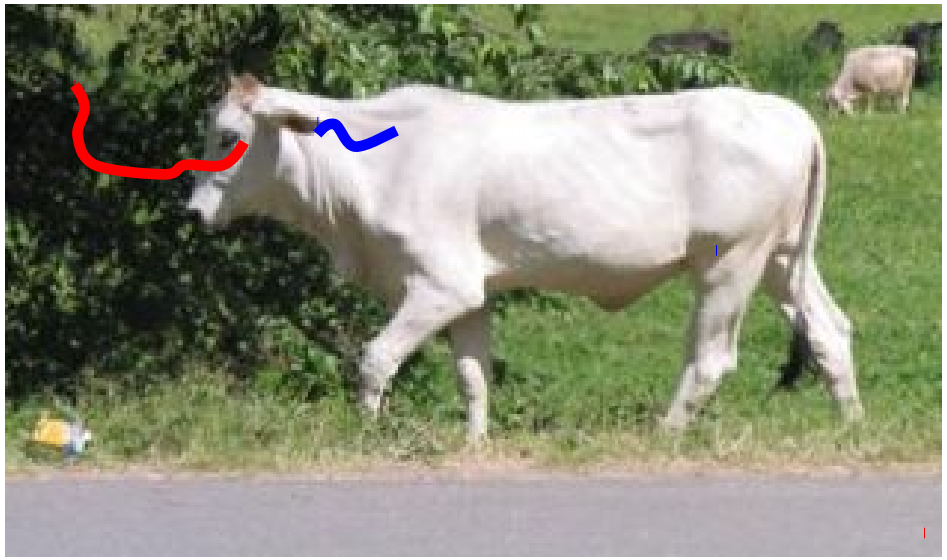
Cost of same label high

Cost of different labels low

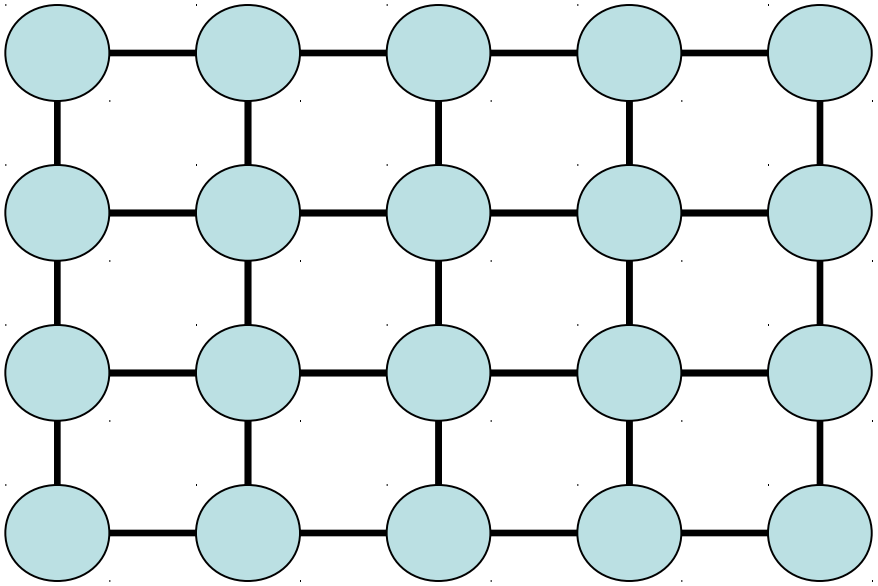
PAIRWISE COST

A Vision Application

Binary Image Segmentation



Object - white, Background - green/grey

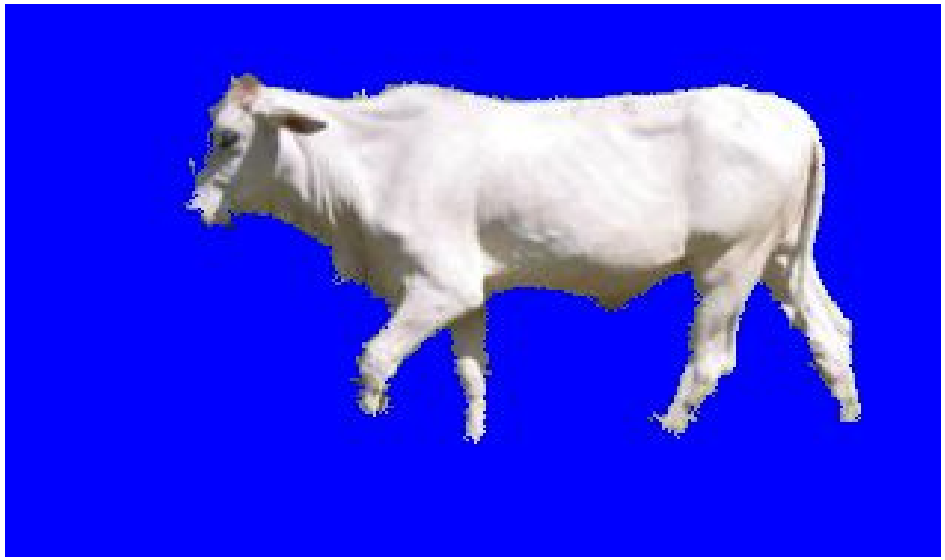


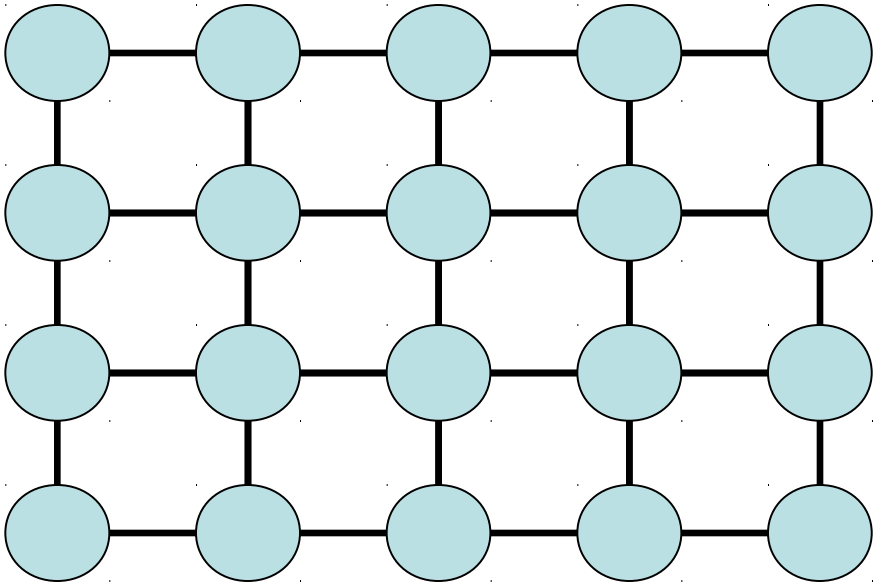
Graph G = (V,E)

Problem: Find the labelling with minimum cost f*

A Vision Application

Binary Image Segmentation

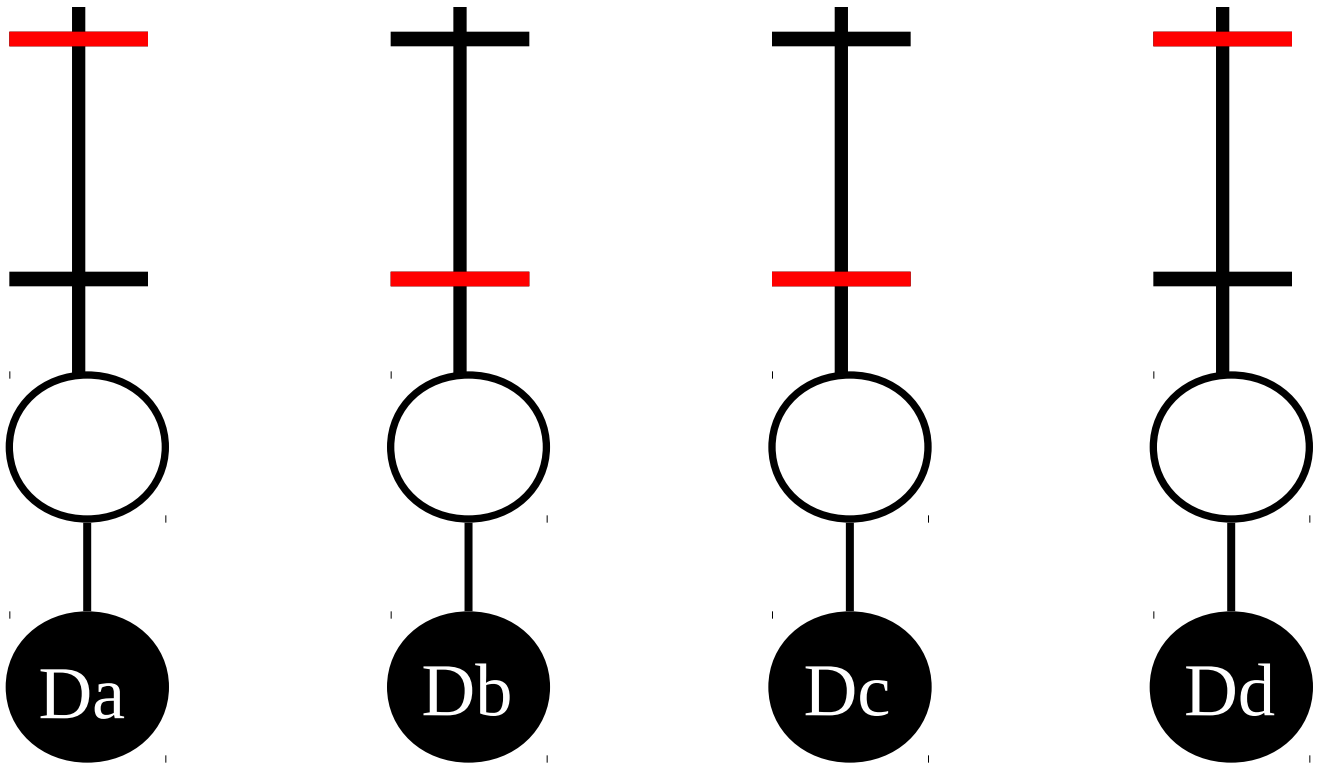




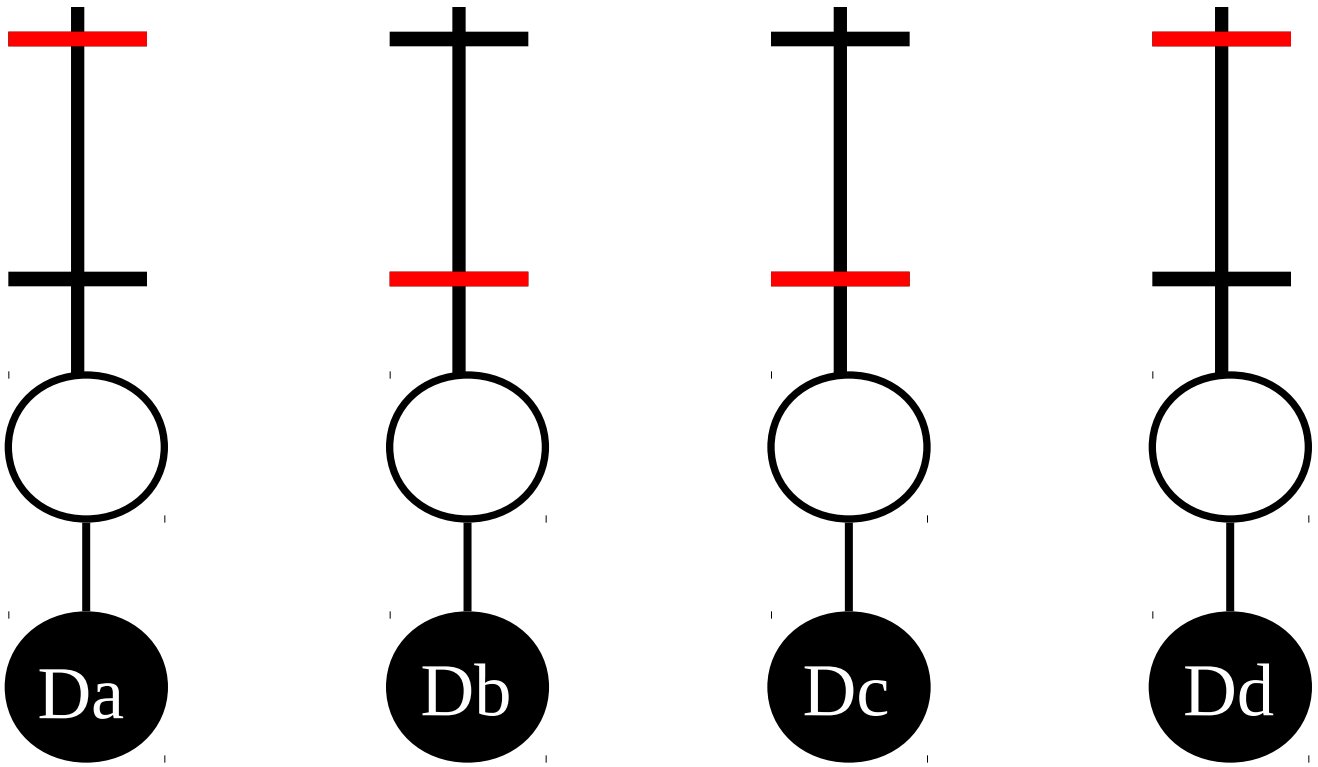
Graph G = (V,E)

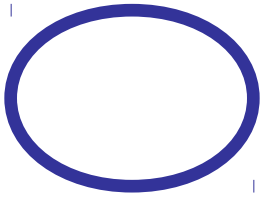
Problem: Find the labelling with minimum cost f*

Energy Function



Energy Function

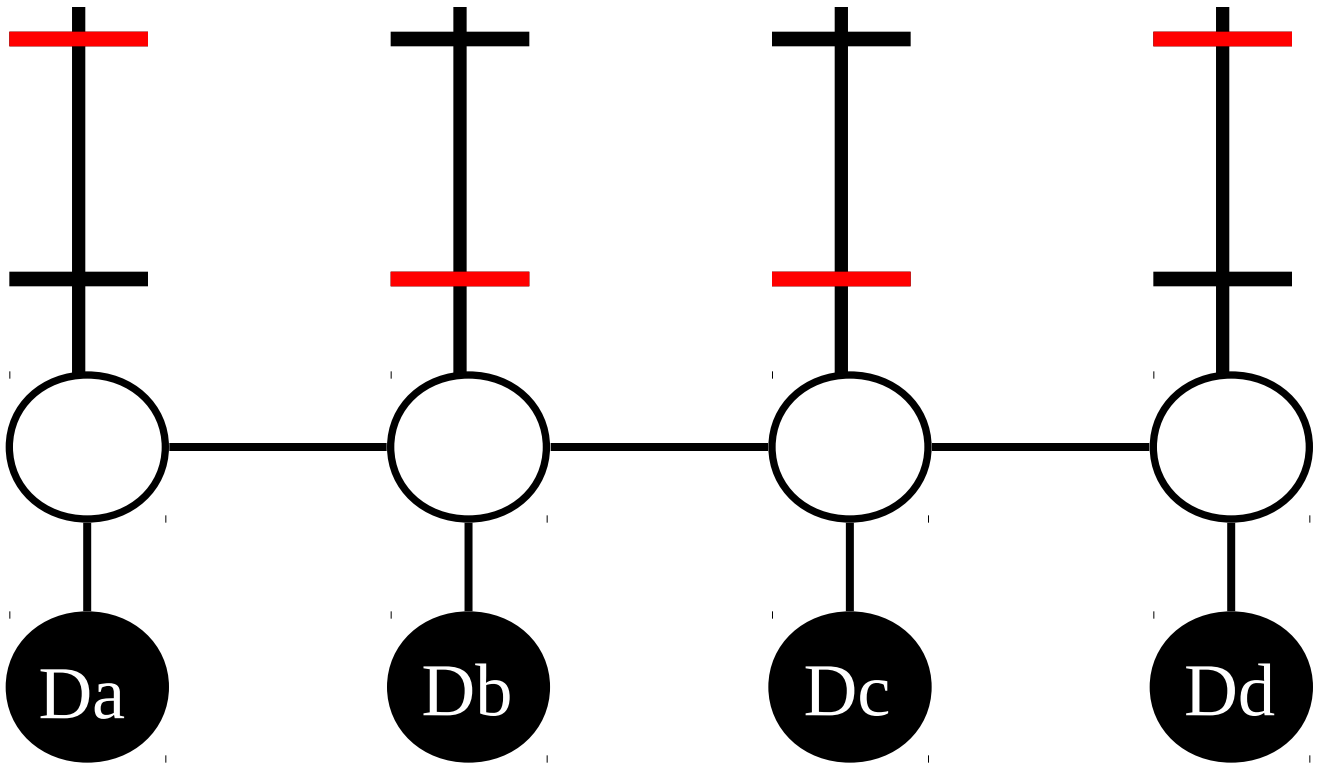




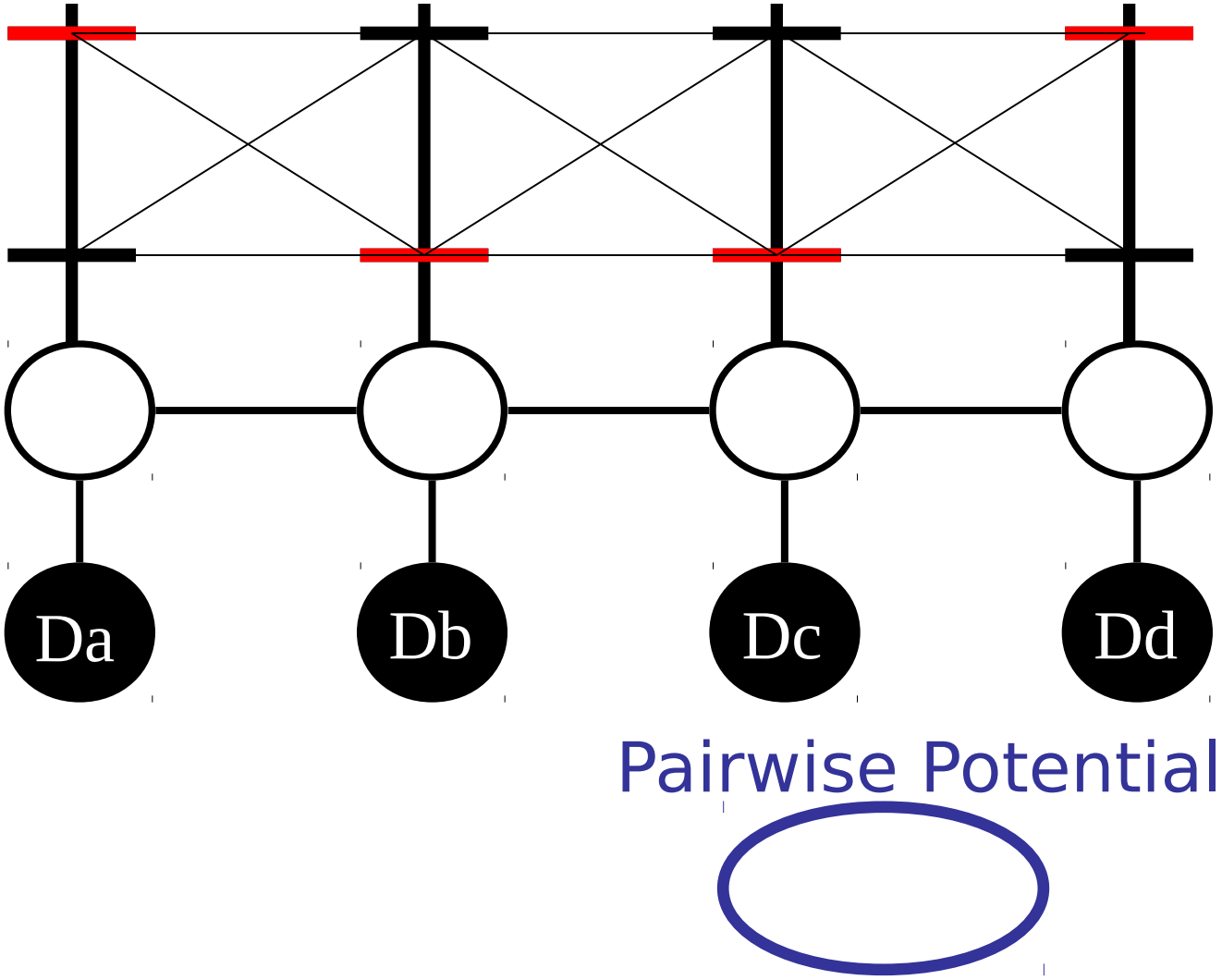
Unary Potential

Easy to minimize

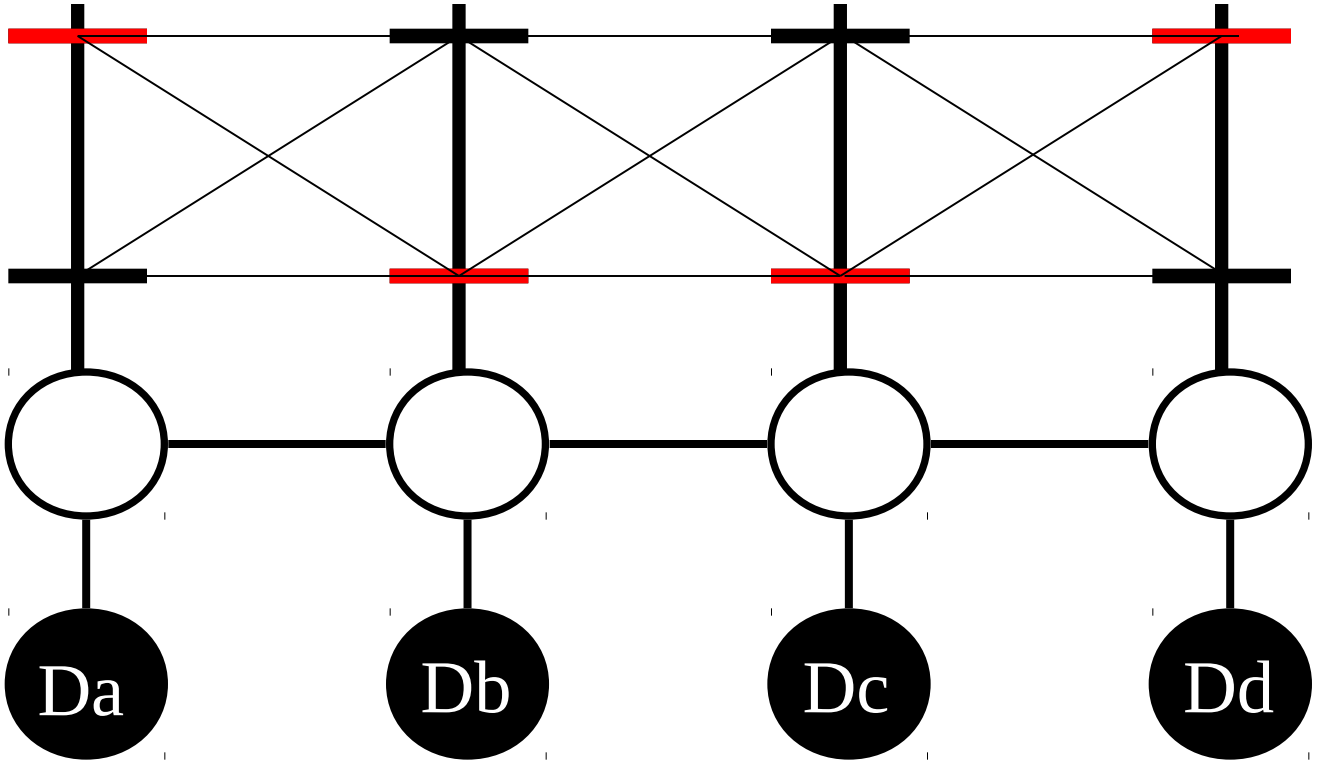
Energy Function



Energy Function

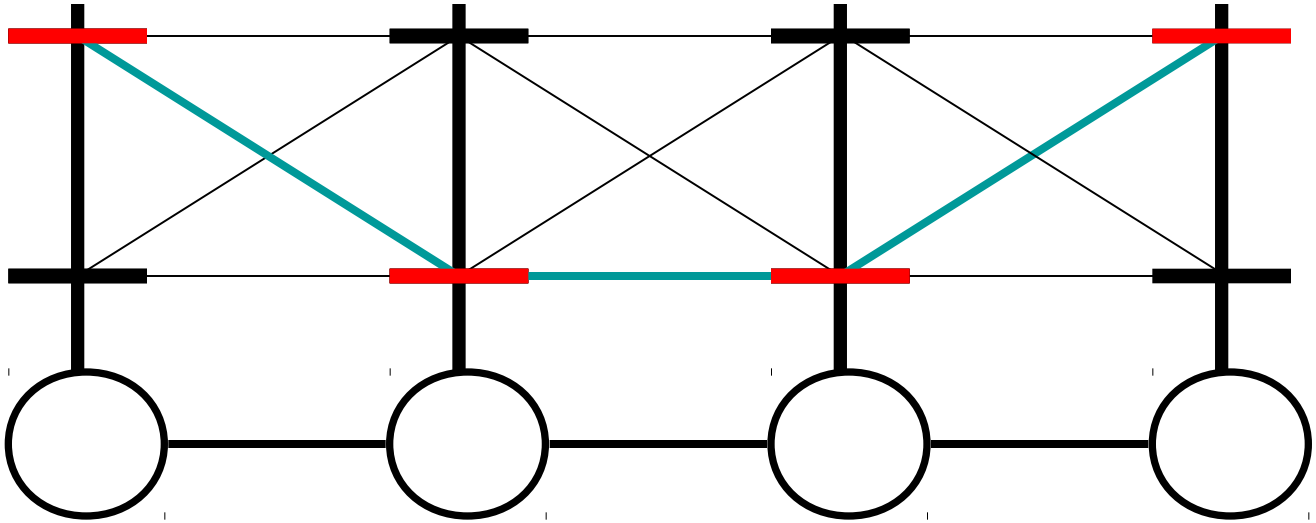


Energy Function



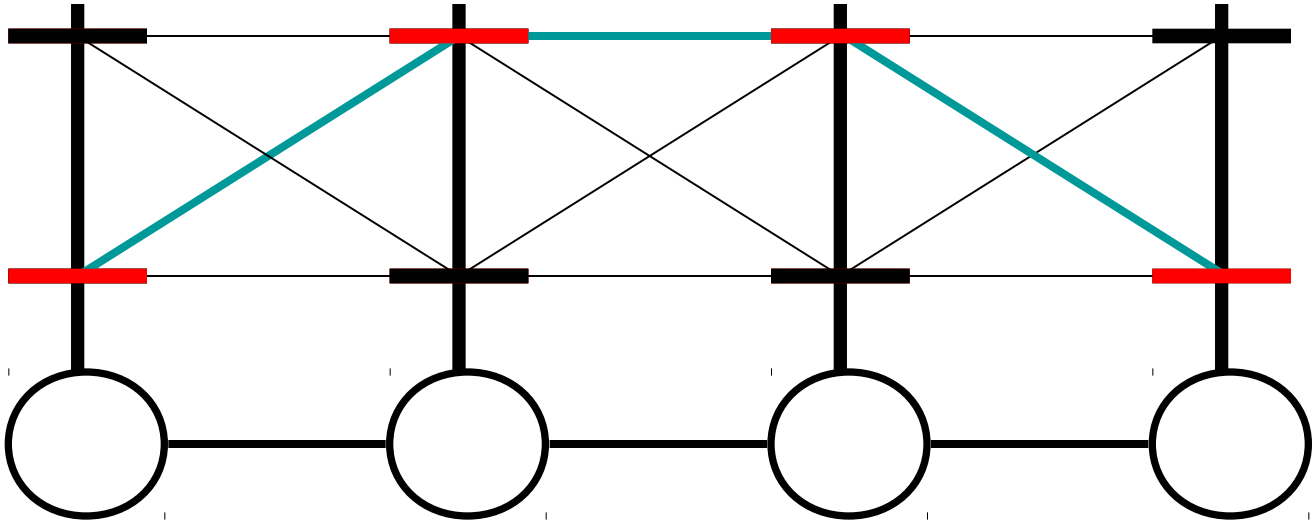
Parameter

MAP Estimation



2 + 1 + 2 + 1 + 3 + 1 + 3 = 13

MAP Estimation



5 + 1 + 4 + 0 + 6 + 4 + 7 = 27

MAP Estimation

$f^* = \{1, 0, 0, 1\}$

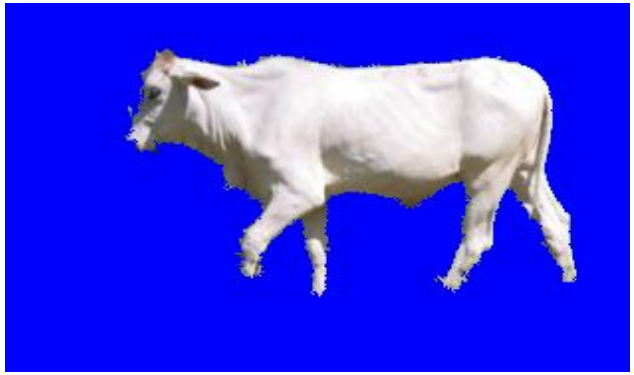
$q^* = 13$

f(a)	f(b)	f(c)	f(d)	Q(f; θ)	f(a)	f(b)	f(c)	f(d)	Q(f; θ)
0	0	0	0	18	1	0	0	0	16
0	0	0	1	15	1	0	0	1	13
0	0	1	0	27	1	0	1	0	25
0	0	1	1	20	1	0	1	1	18
0	1	0	0	22	1	1	0	0	18
0	1	0	1	19	1	1	0	1	15
0	1	1	0	27	1	1	1	0	23
0	1	1	1	20	1	1	1	1	16

Computational Complexity

Segmentation

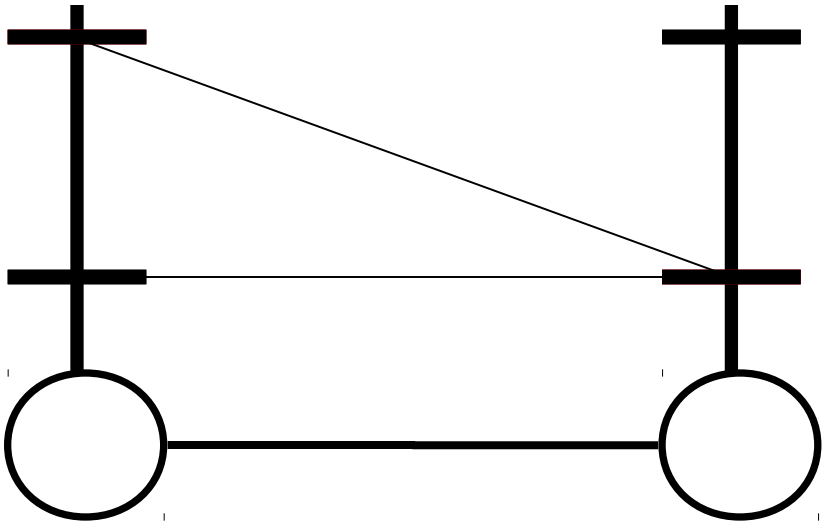
$2^{|V|}$

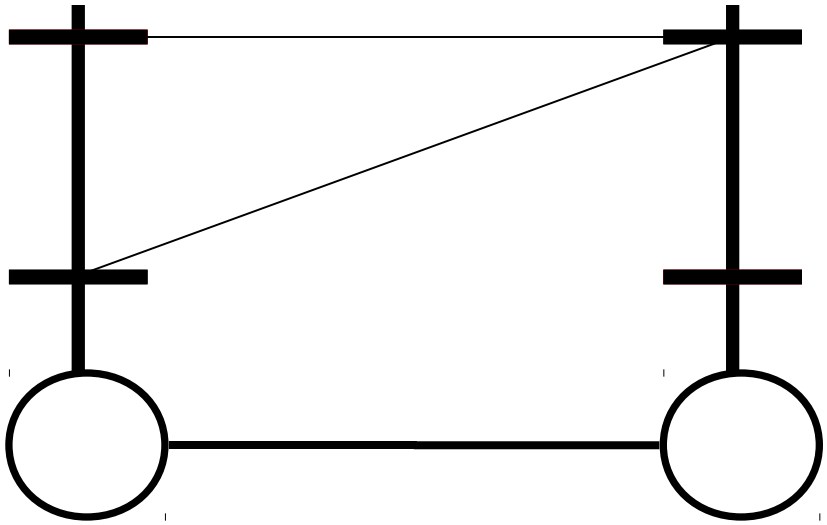


$|V|$ = number of pixels $\approx 320 * 480 = 153600$

number of atoms in universe ~= **2^{270}**

Belief propagation: Two Variables

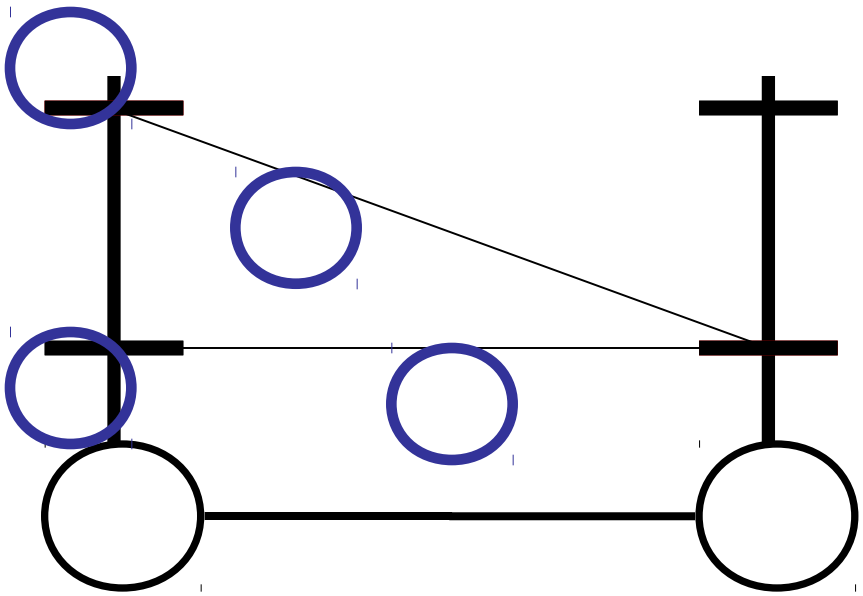


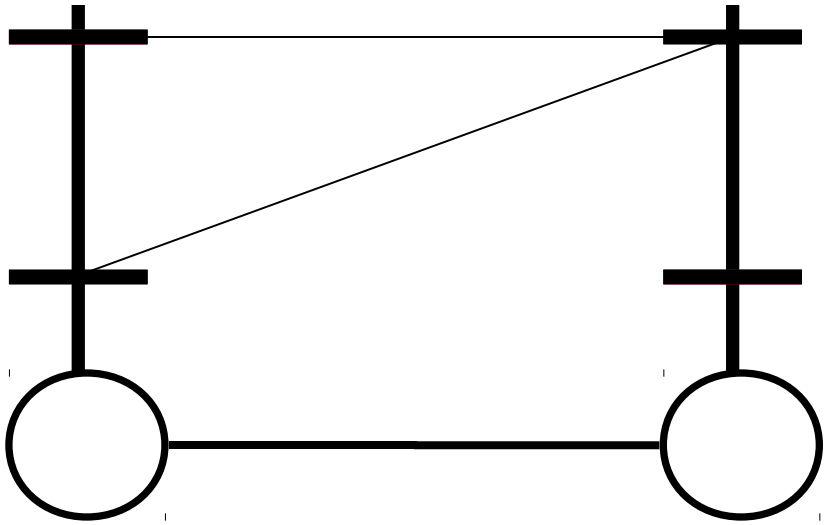


Add a constant to one $\theta_{b;k}$

ubtract that constant from $\theta_{ab;ik}$ for all ‘i’

Two Variables



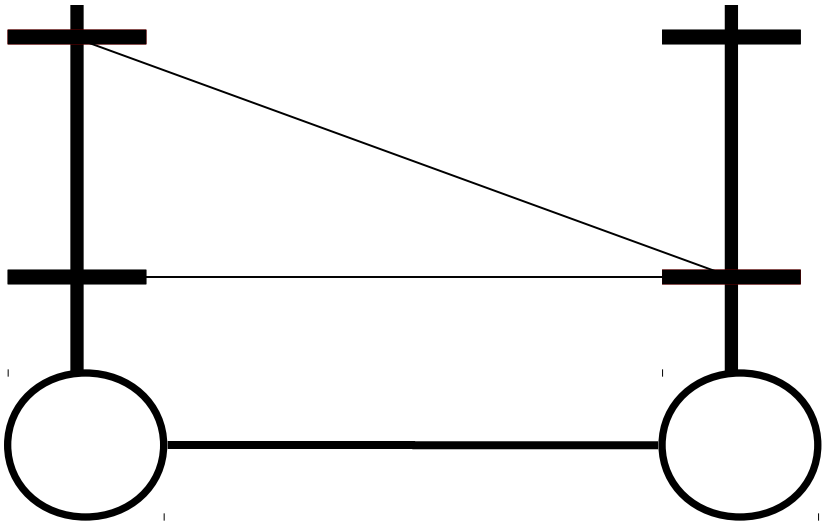


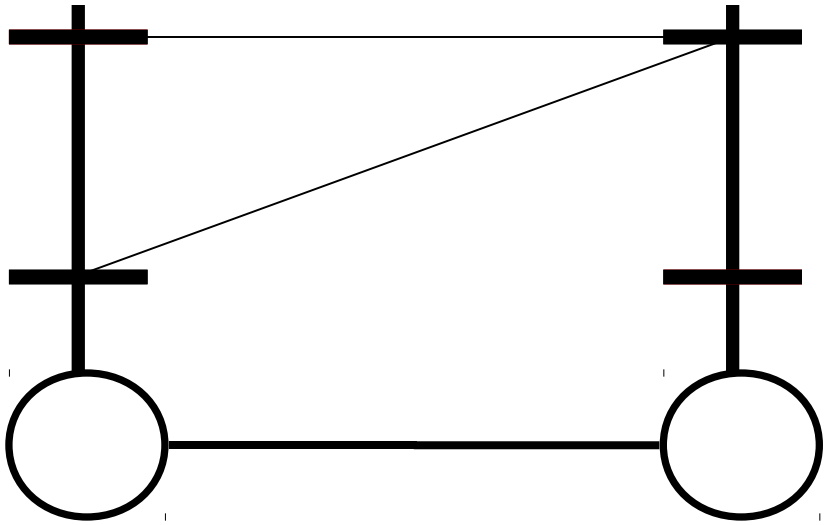
Mab;0 = min

right

θ’b;k = qb;k

Two Variables



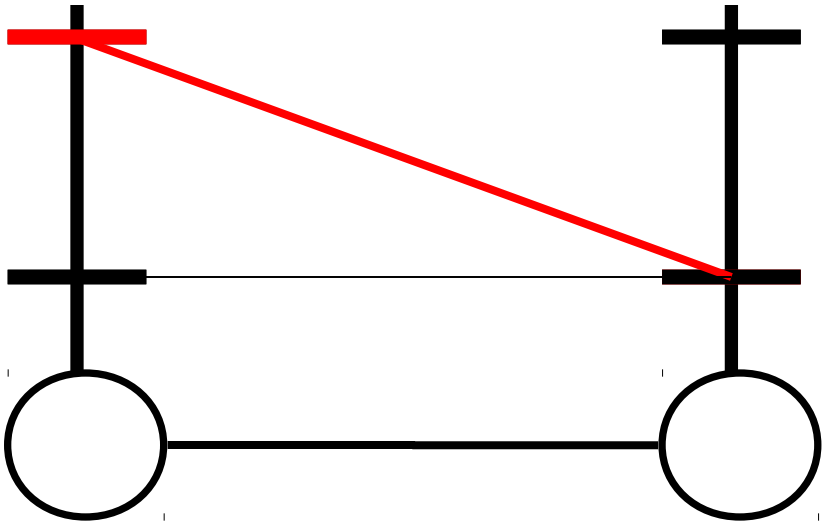


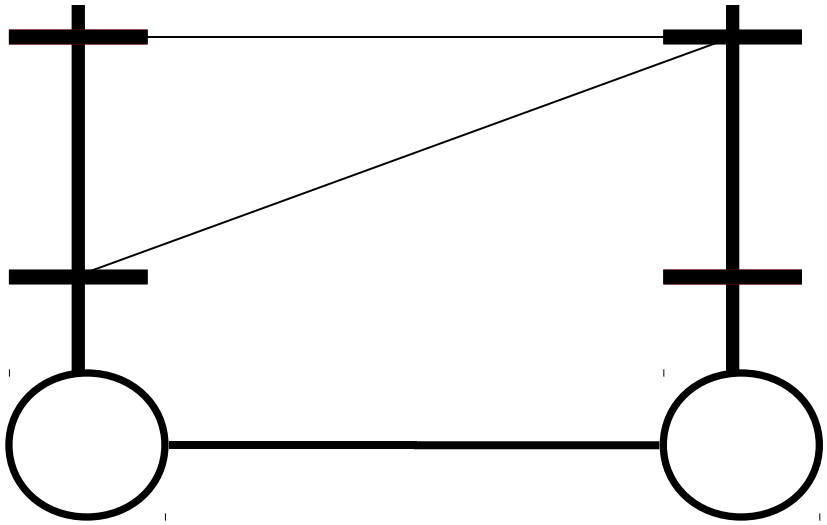
right

$\theta'_{b;k} = q_{b;k}$

Two Variables

f(a) = 1





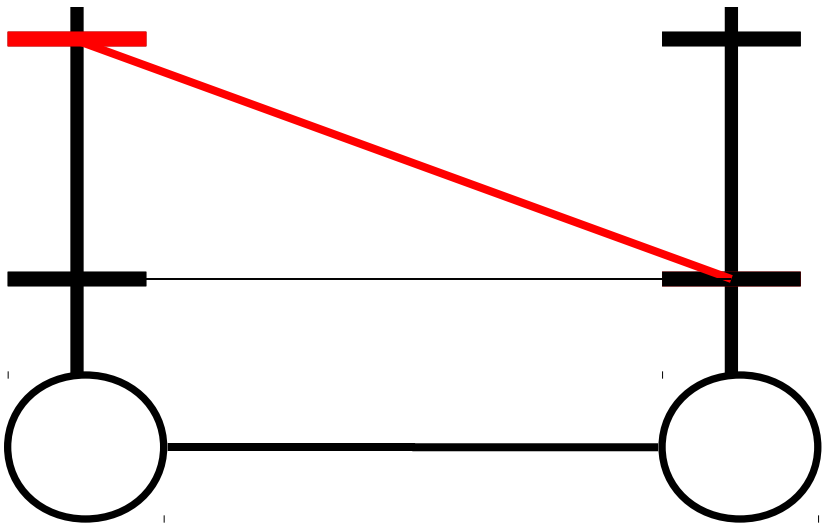
$\theta'_{b;0} = q_{b;0}$

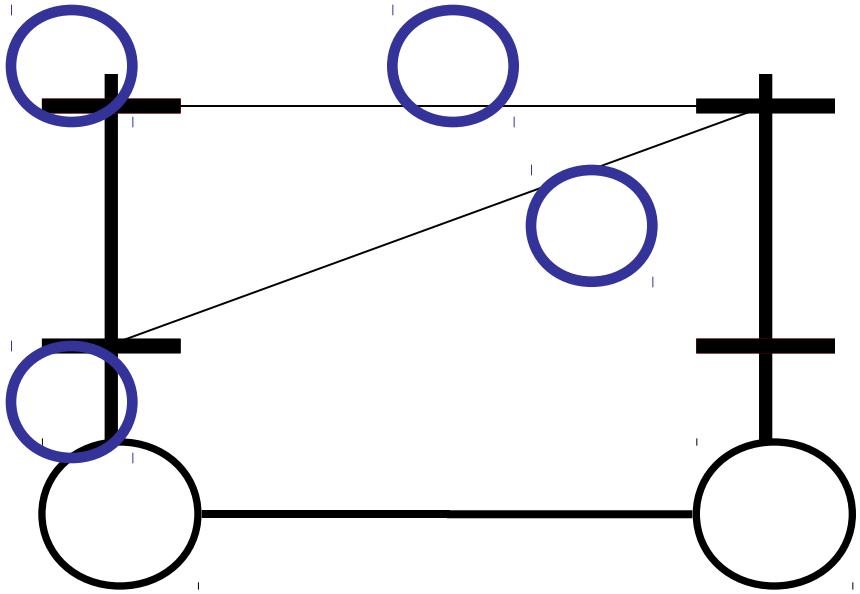
Potentials along the red path add up to 0

right

$\theta'_{b;k} = q_{b;k}$

Two Variables





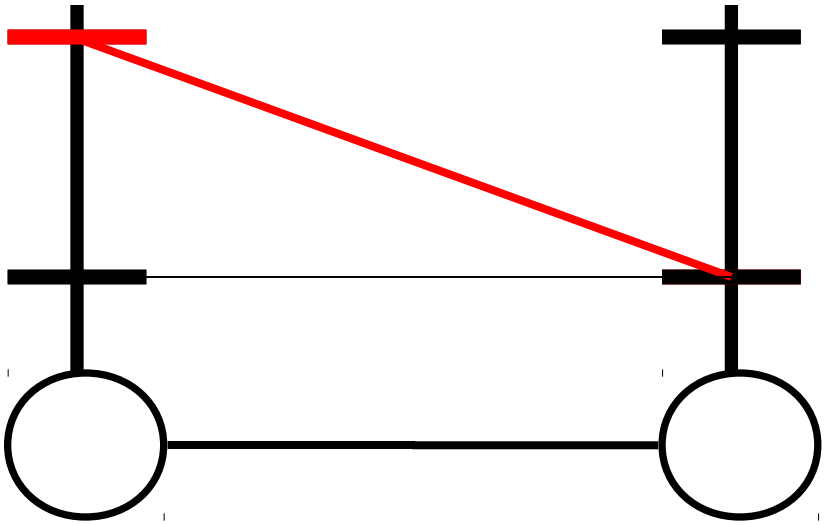
Mab;1 = min

right

θ’b;k = qb;k

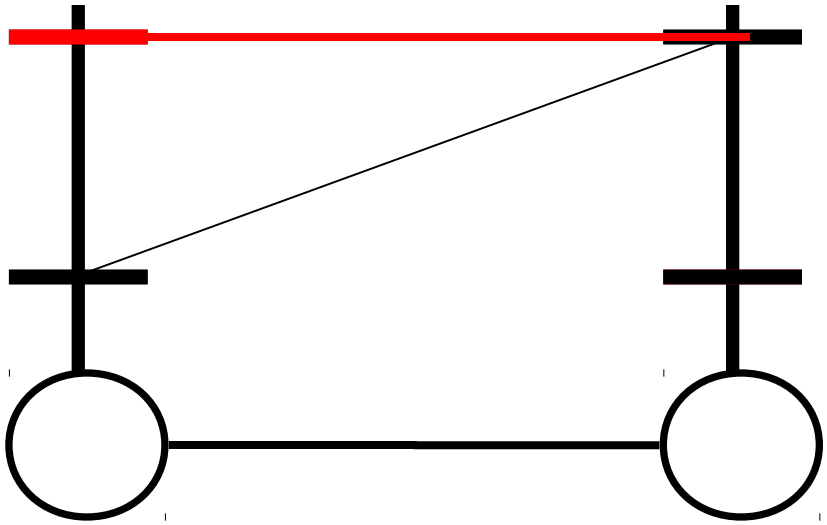
Two Variables

f(a) = 1



$\theta'_{b;0} = q_{b;0}$

f(a) = 1



$\theta'_{b;1} = q_{b;1}$

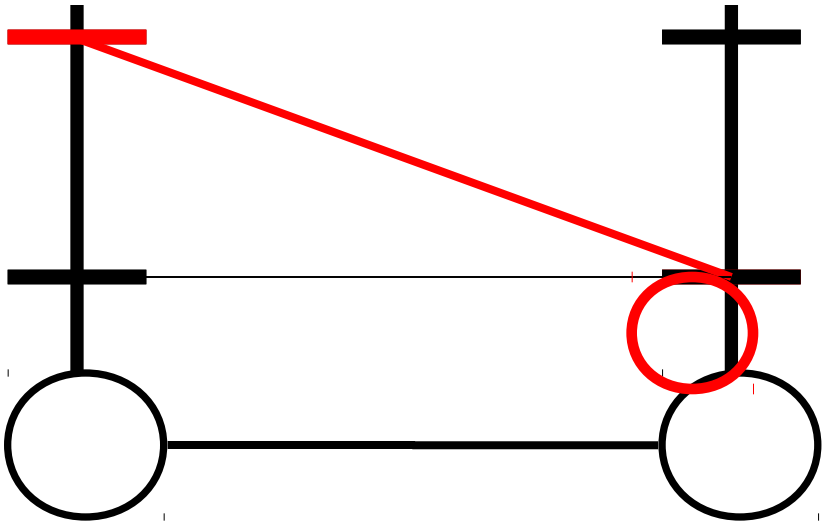
Minimum of min-marginals = MAP estimate

right

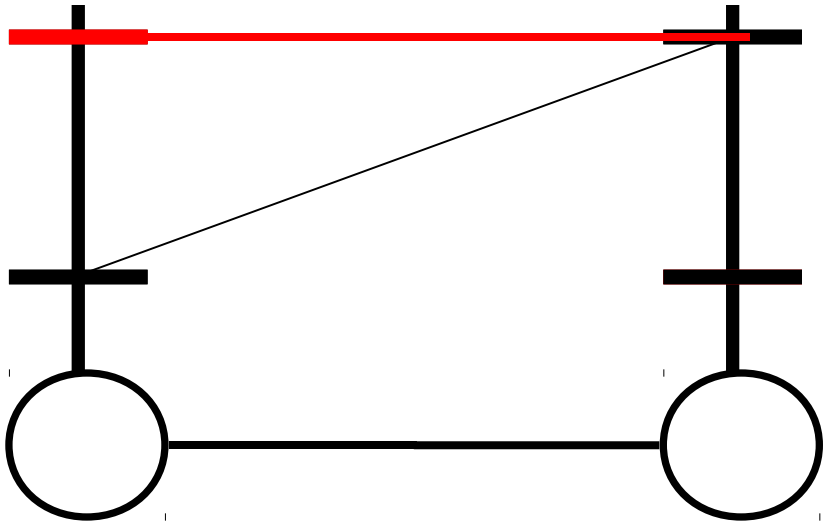
$\theta'_{b;k} = q_{b;k}$

Two Variables

f(a) = 1



f(a) = 1



θ’b;0 = qb;0

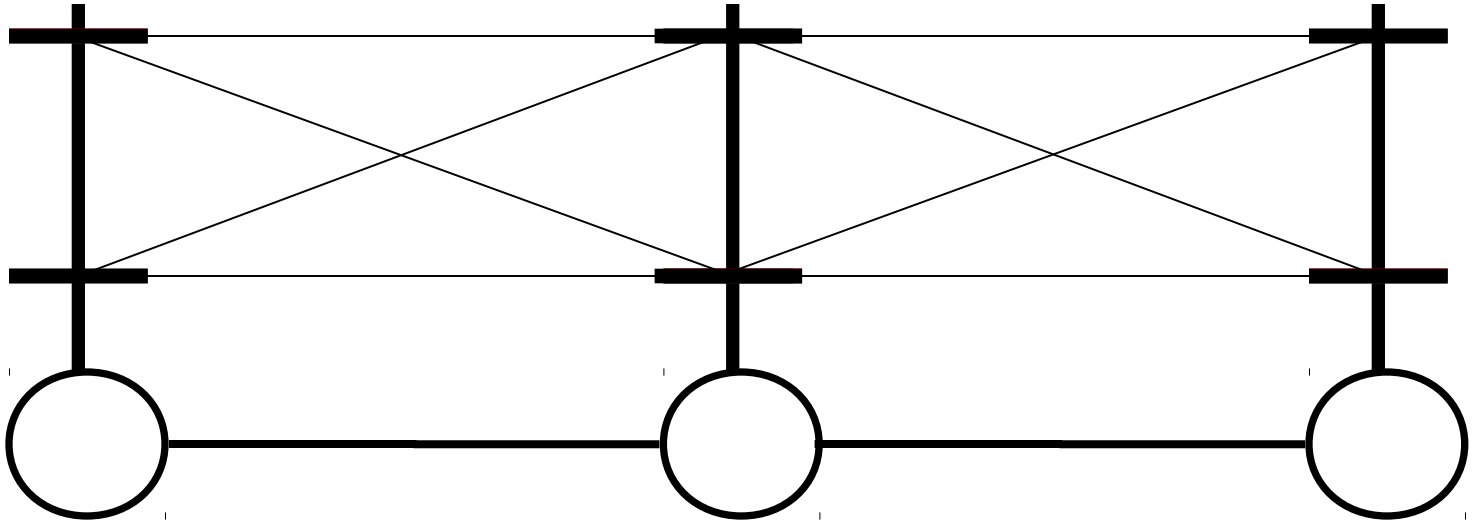
θ’b;1 = qb;1

f*(b) = 0f*(a) = 1

right

θ’b;k = qb;k

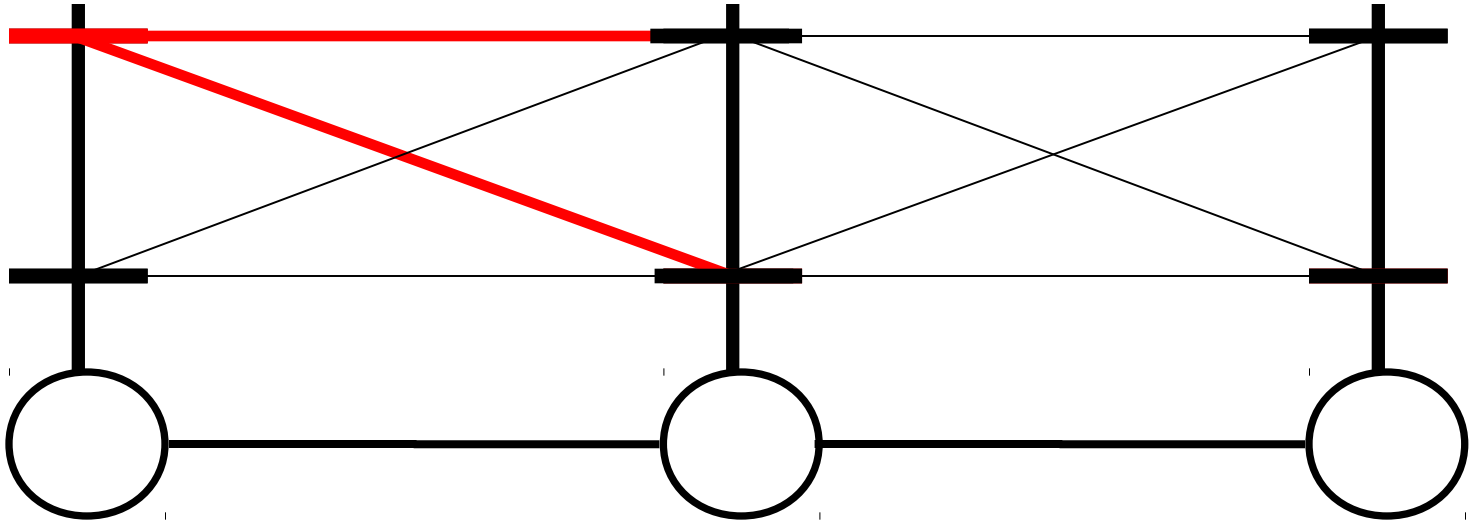
Three Variables



Reparameterize the edge (a,b) as before

Three Variables

f(a) = 1



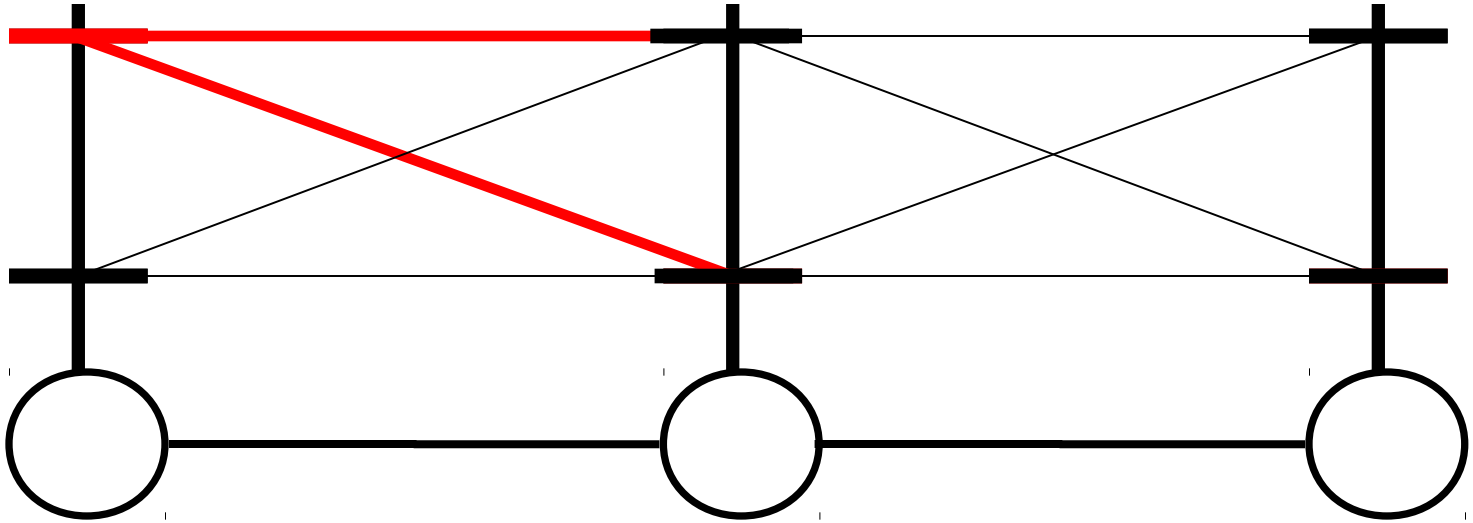
f(a) = 1

Reparameterize the edge (a,b) as before

Potentials along the red path add up to 0

Three Variables

f(a) = 1



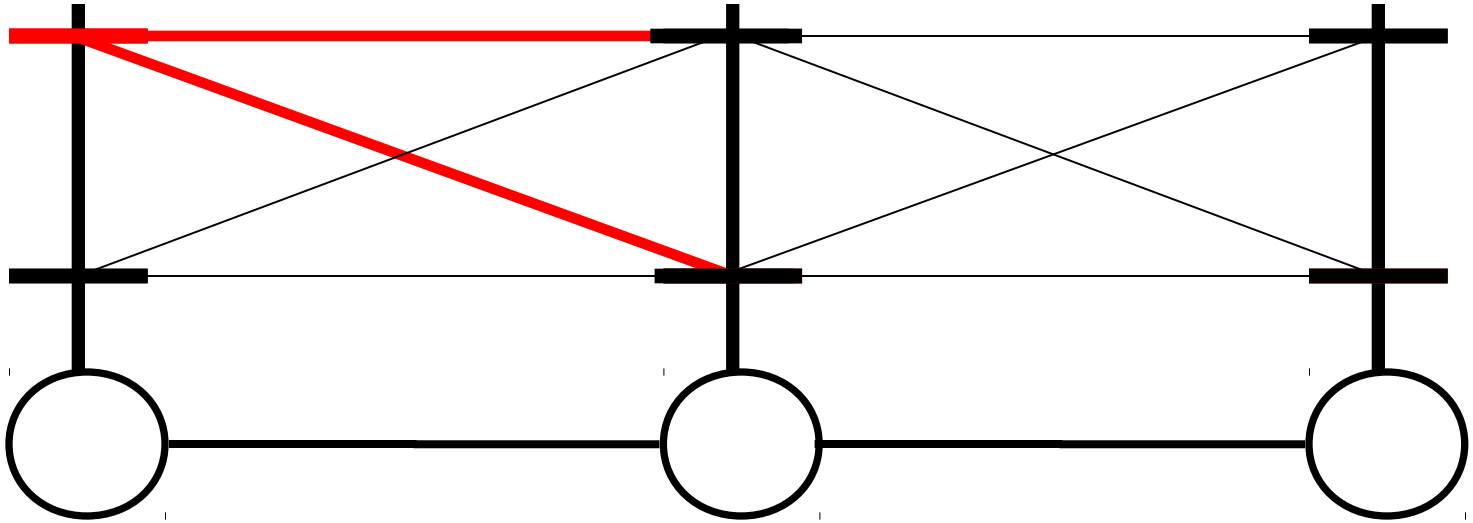
f(a) = 1

Reparameterize the edge (b,c) as before

Potentials along the red path add up to 0

Three Variables

f(a) = 1



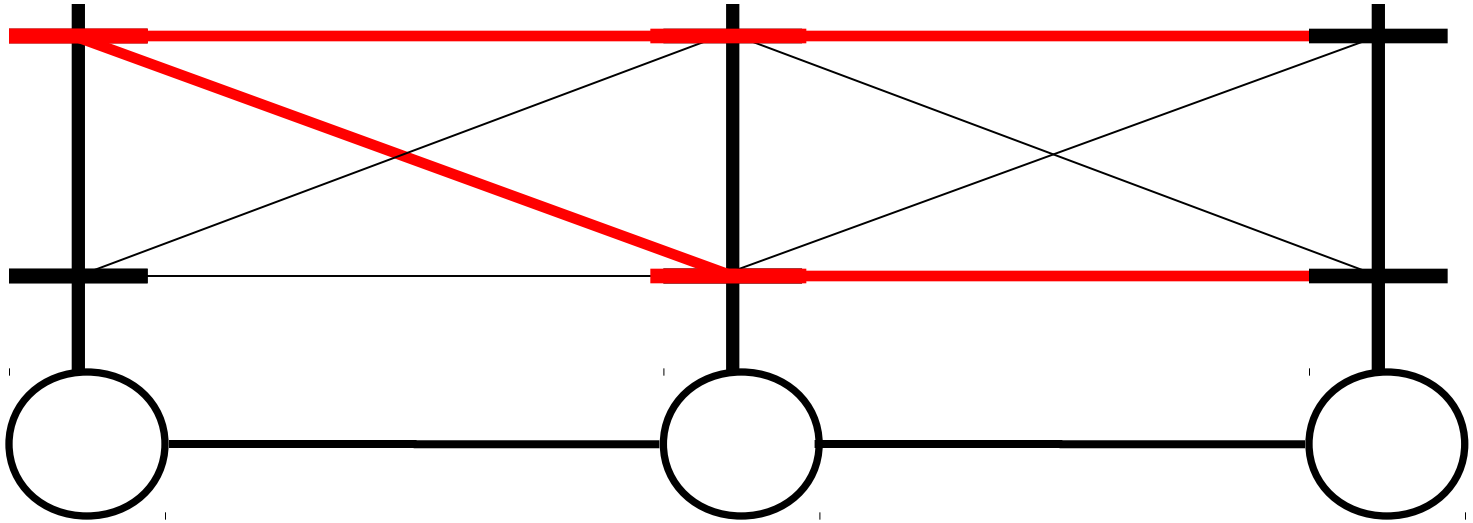
f(a) = 1

Reparameterize the edge (b,c) as before

Potentials along the red path add up to 0

Three Variables

f(a) = 1 f(b) = 1



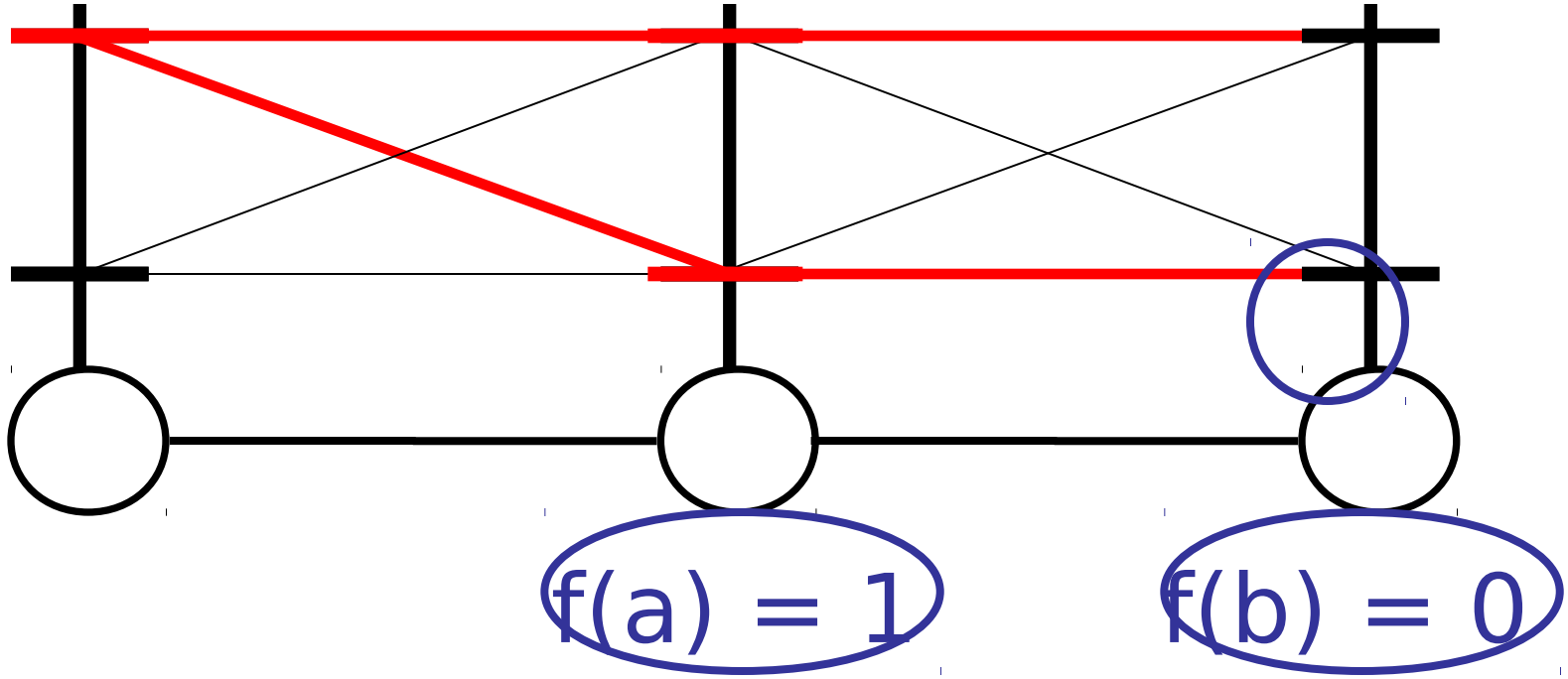
f(a) = 1 f(b) = 0

Reparameterize the edge (b,c) as before

Potentials along the red path add up to 0

Three Variables

f(a) = 1 f(b) = 1



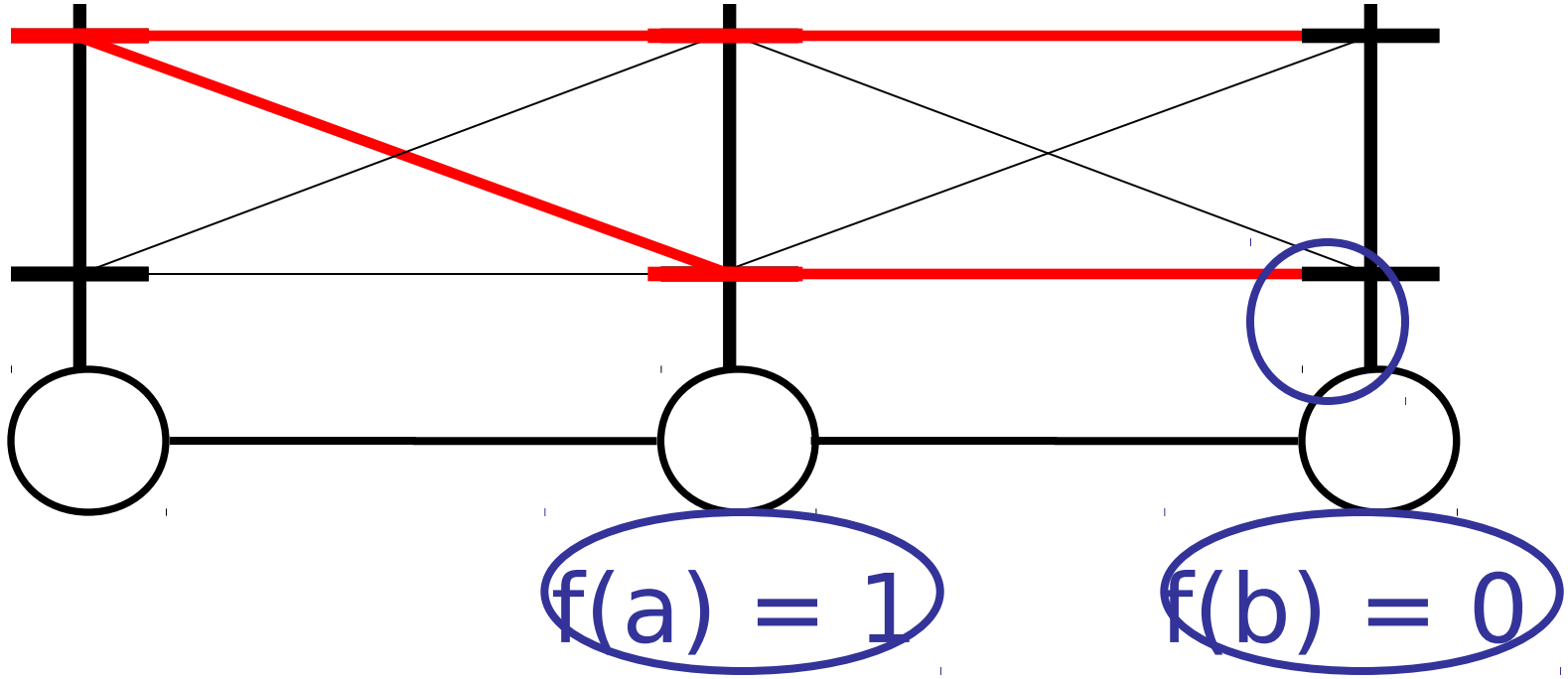
f(a) = 1 f(b) = 0

f*(c) = 0 f*(b) = 0 f*(a) = 1

Generalizes to any length chain

Three Variables

f(a) = 1 f(b) = 1

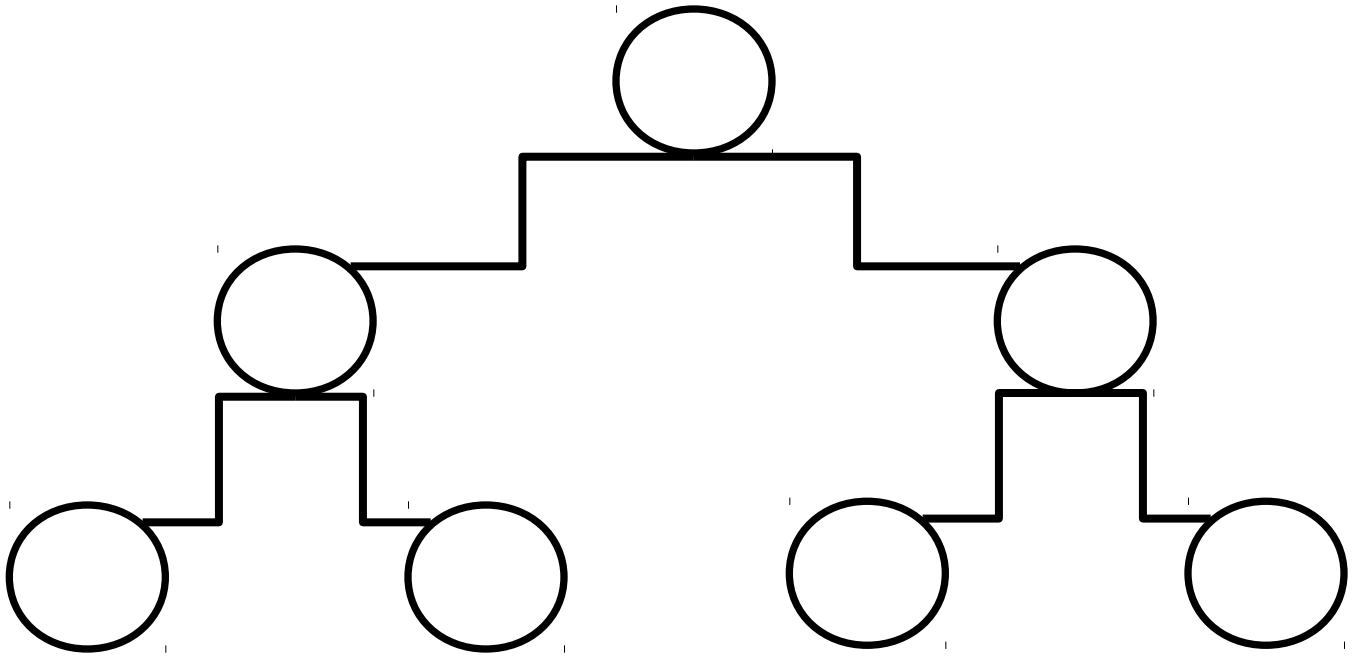


f(a) = 1 f(b) = 0

f*(c) = 0 f*(b) = 0 f*(a) = 1

Only Dynamic Programming

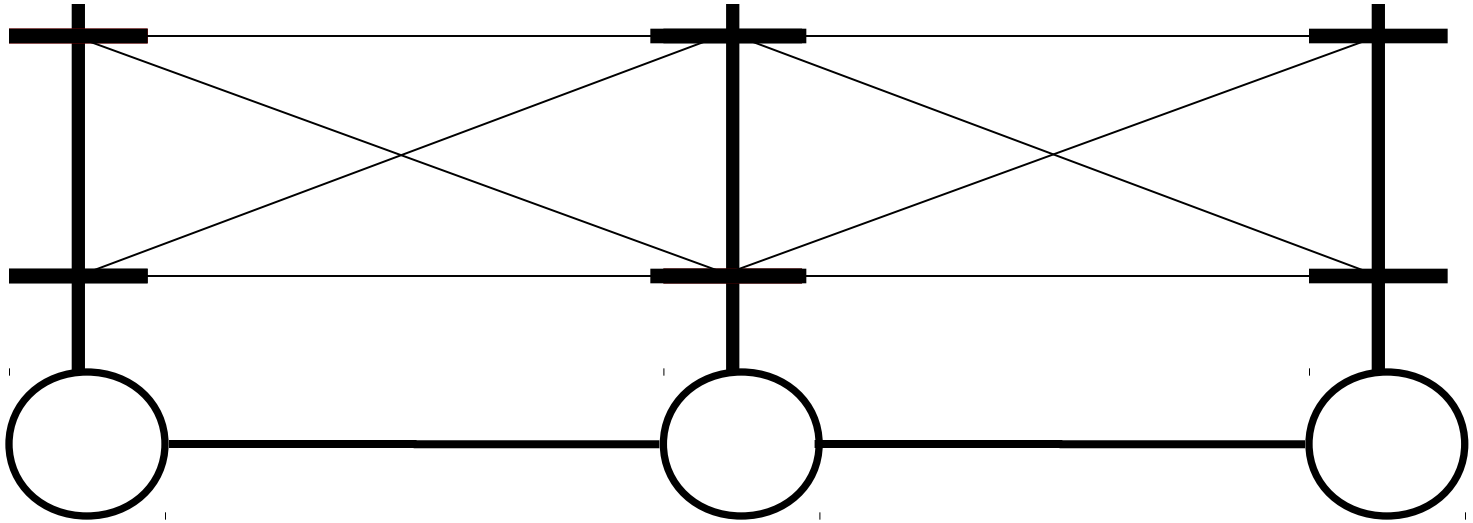
Belief Propagation on Trees



Forward Pass: Leaf □ Root

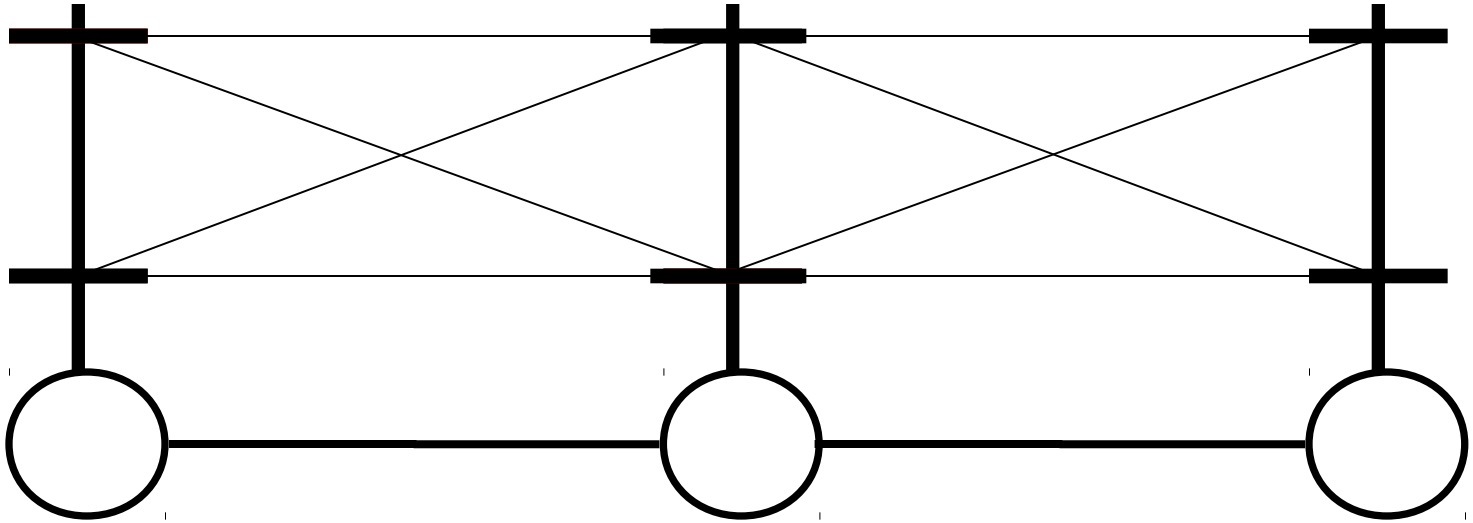
Backward Pass: Root □ Leaf

Three Variables



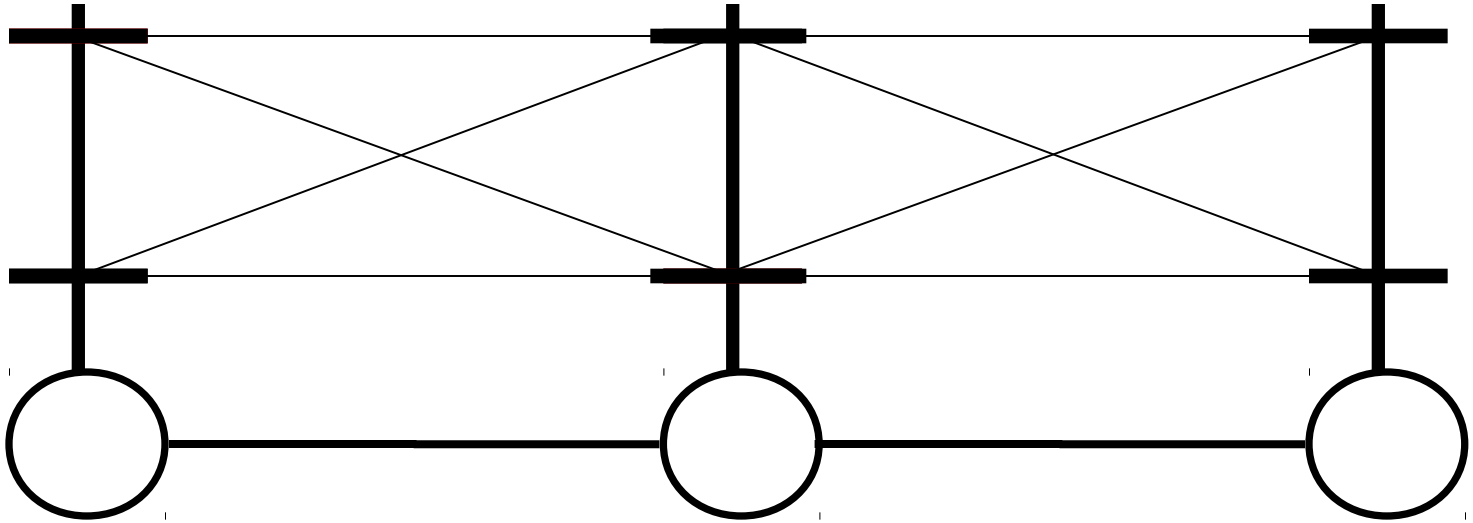
Reparameterize the edge (c,b) as before

Three Variables



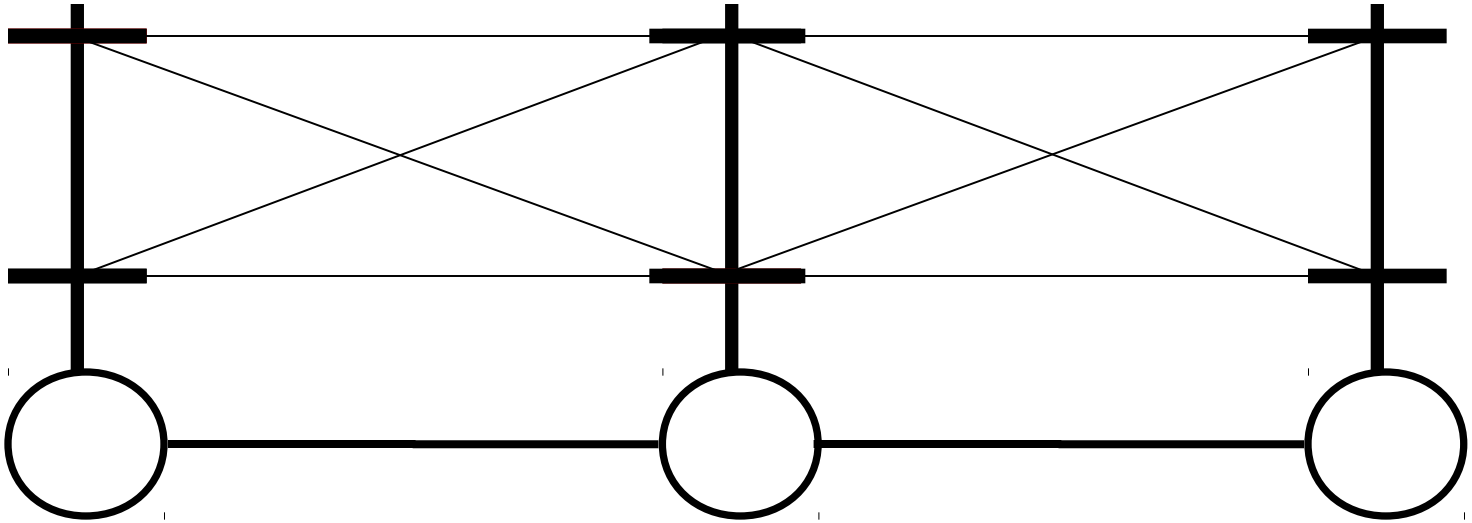
Reparameterize the edge (c,b) as before

Three Variables



Reparameterize the edge (b,a) as before

Three Variables



Forward Pass Backward Pass

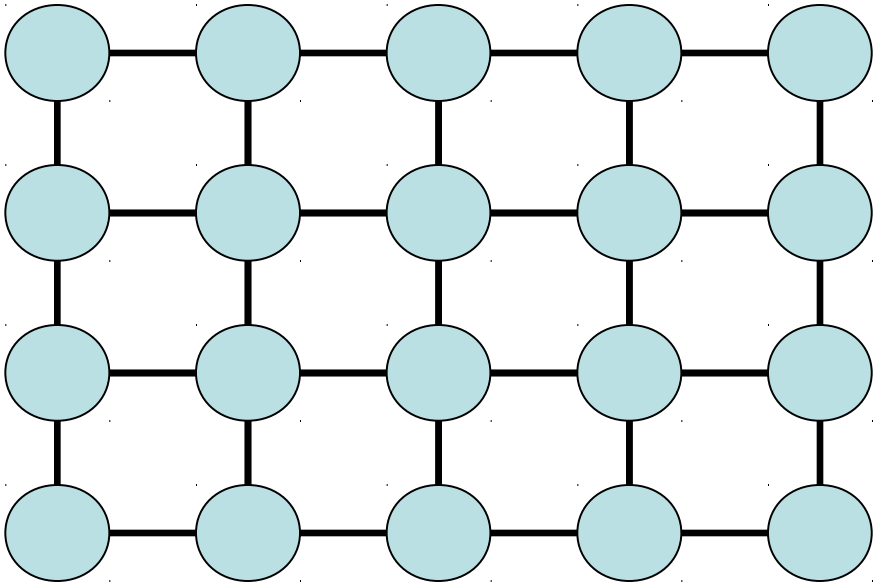
All min-marginals are computed

Belief Propagation

- $O(|E||L|2)$
- $O(|E||L|)$

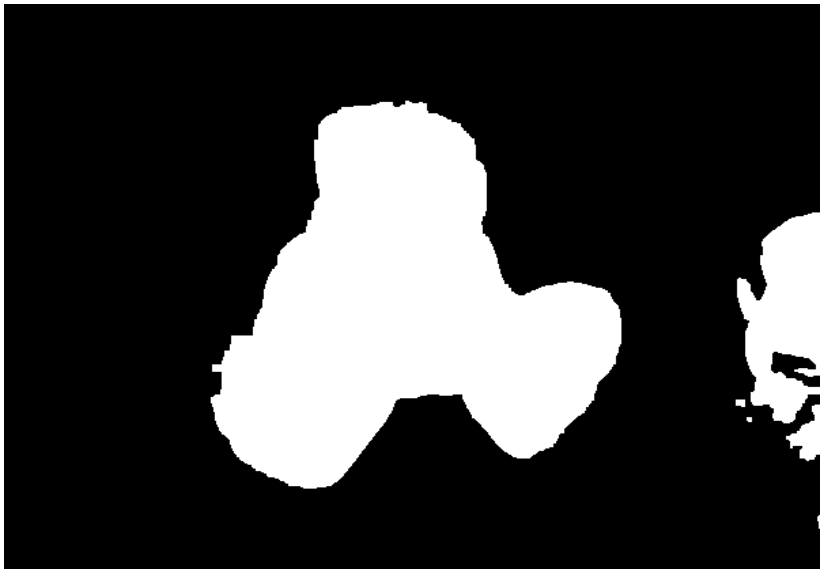
Results

Szeliski et al. , 2008



Results

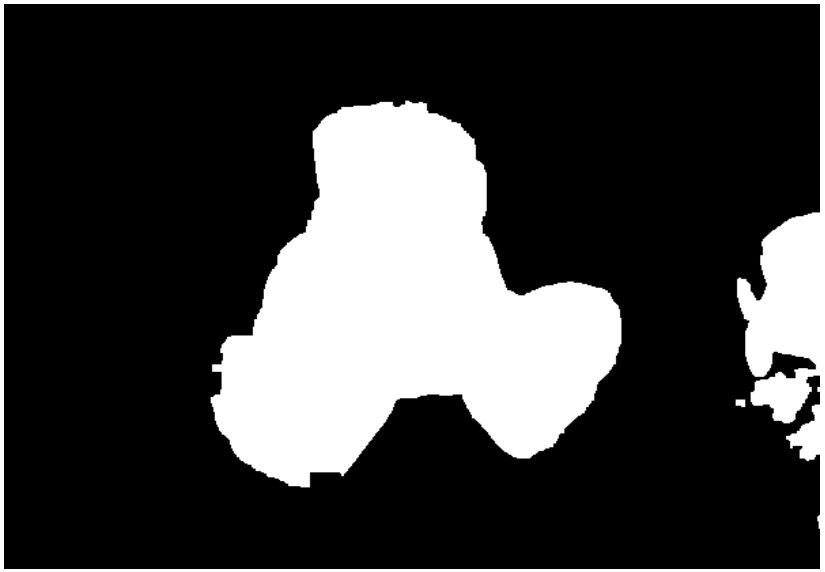
Szeliski et al. , 2008



Belief Propagation

Results

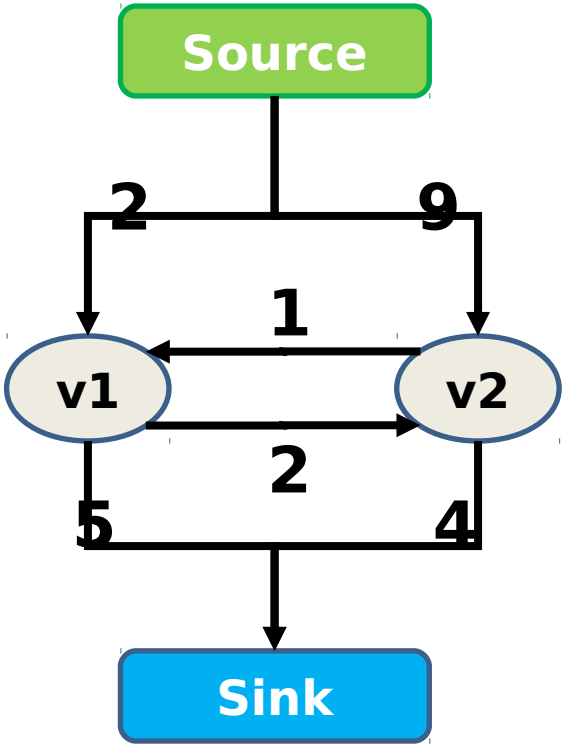
Szeliski et al. , 2008



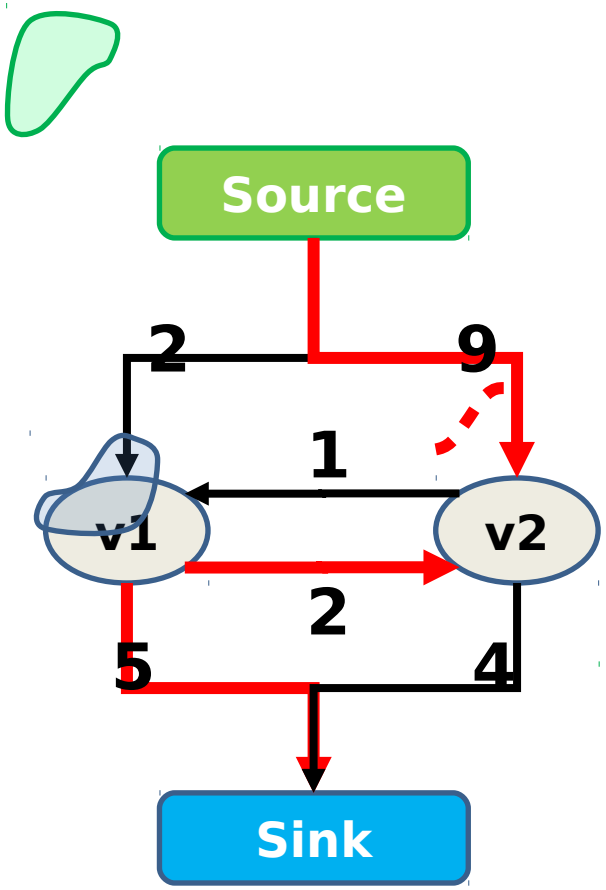
Global optimum

The st-Mincut Problem

What is an st-cut?



The st-Mincut Problem



5 + 2 + 9 = 16

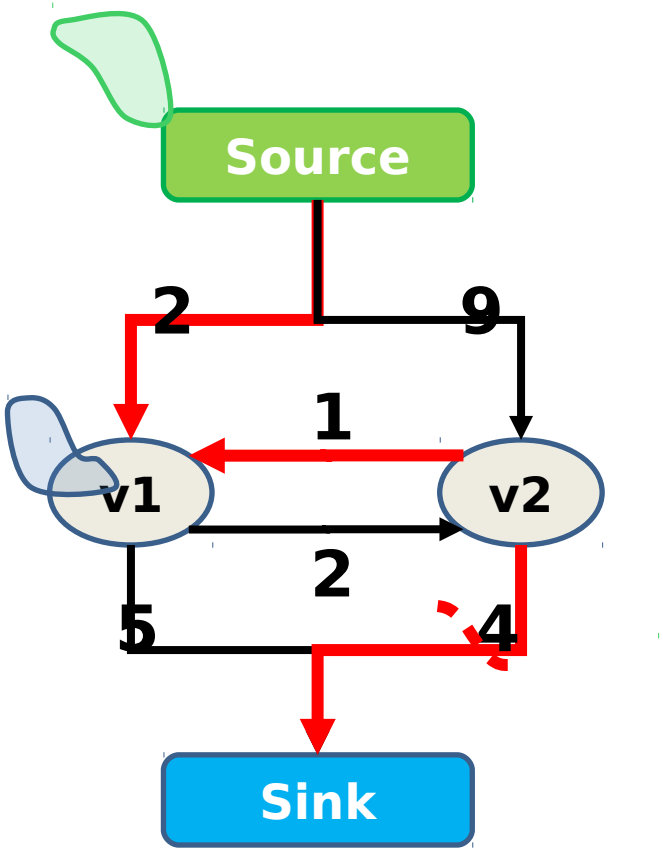
What is an st-cut?

An st-cut (**S**,**T**) divides the nodes between source and sink.

What is the cost of an st-cut?

Sum of cost of all edges going from S to T

The st-Mincut Problem



2 + 1 + 4 = 7

What is an st-cut?

An st-cut (**S**,**T**) divides the nodes between source and sink.

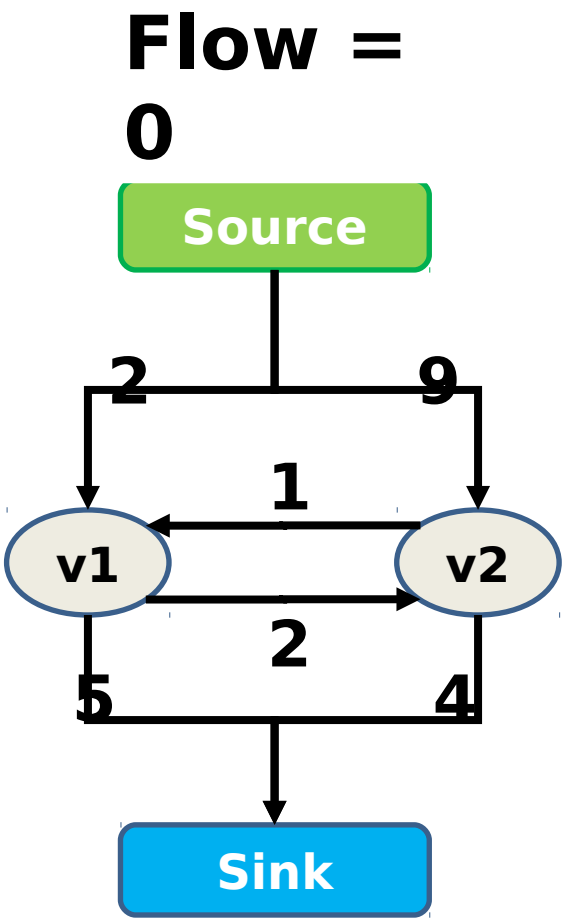
What is the cost of an st-cut?

Sum of cost of all edges going from S to T

What is the st-mincut?

st-cut with the minimum cost

Maxflow Algorithms

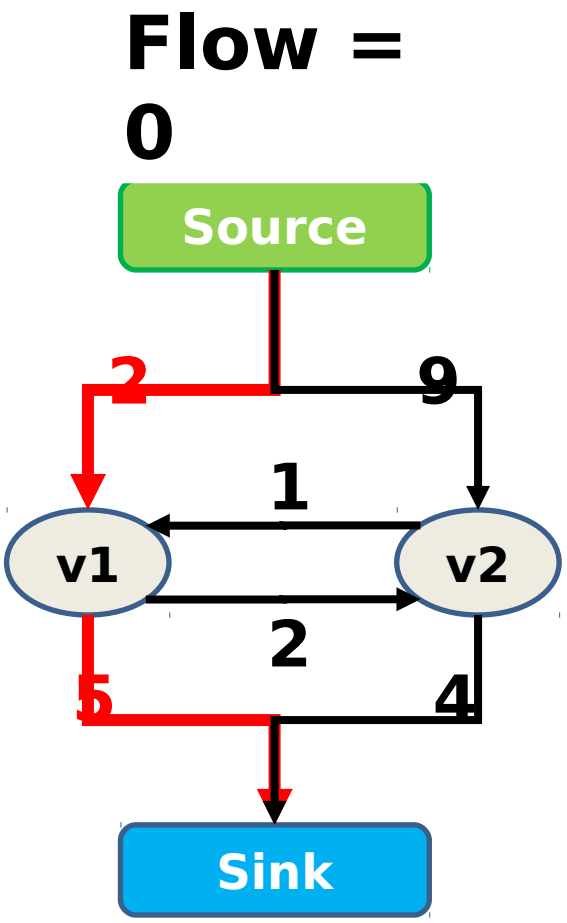


Augmenting Path Based Algorithms

1. Find path from source to sink with positive capacity
2. Push maximum possible flow through this path
3. Repeat until no path can be found

Algorithms assume non-negative capacity

Maxflow Algorithms

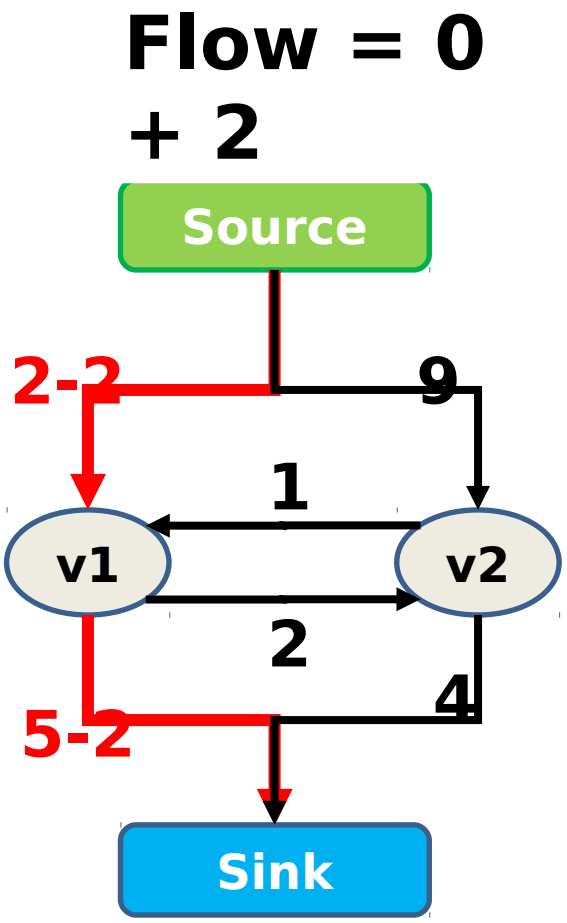


Augmenting Path Based Algorithms

1. Find path from source to sink with positive capacity
2. Push maximum possible flow through this path
3. Repeat until no path can be found

Algorithms assume non-negative capacity

Maxflow Algorithms

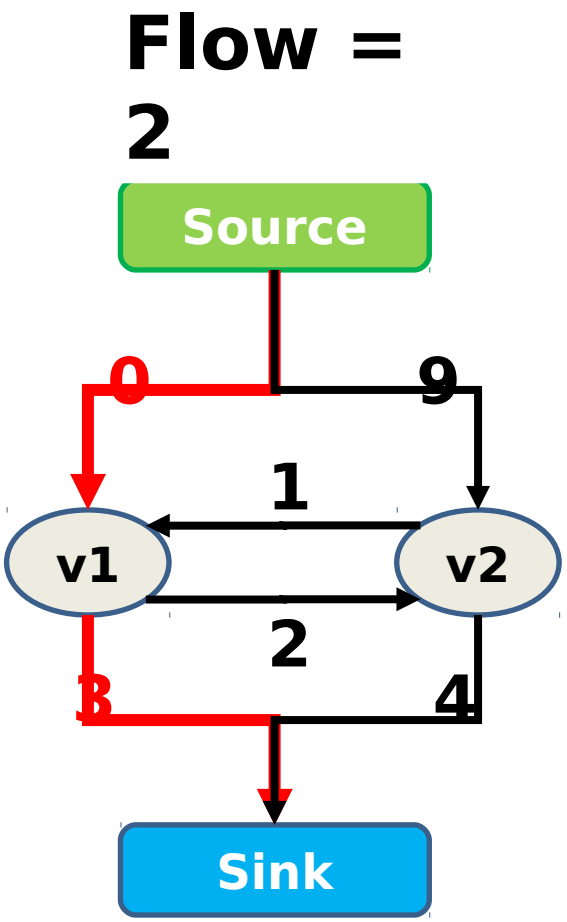


Augmenting Path Based Algorithms

1. Find path from source to sink with positive capacity
2. Push maximum possible flow through this path
3. Repeat until no path can be found

Algorithms assume non-negative capacity

Maxflow Algorithms

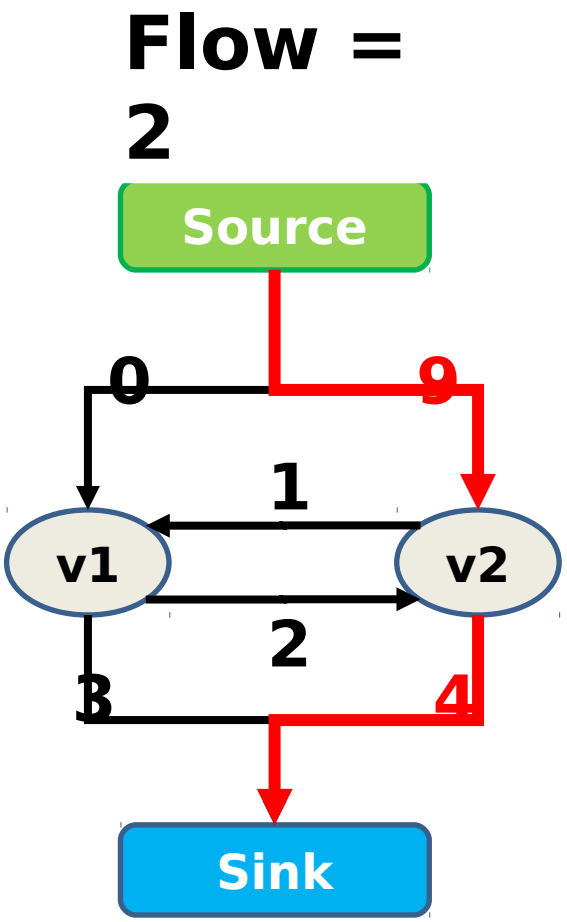


Augmenting Path Based Algorithms

1. Find path from source to sink with positive capacity
2. Push maximum possible flow through this path
3. Repeat until no path can be found

Algorithms assume non-negative capacity

Maxflow Algorithms

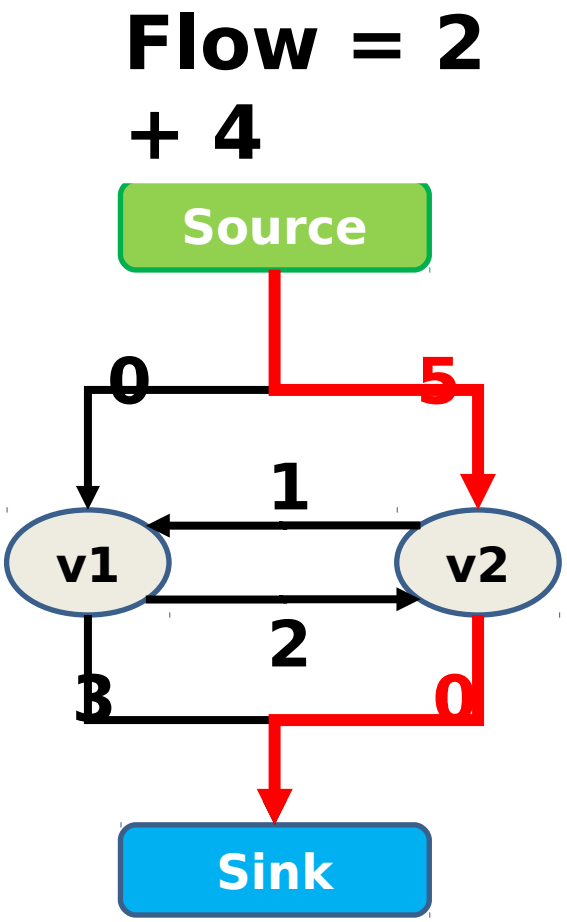


Augmenting Path Based Algorithms

1. Find path from source to sink with positive capacity
2. Push maximum possible flow through this path
3. Repeat until no path can be found

Algorithms assume non-negative capacity

Maxflow Algorithms



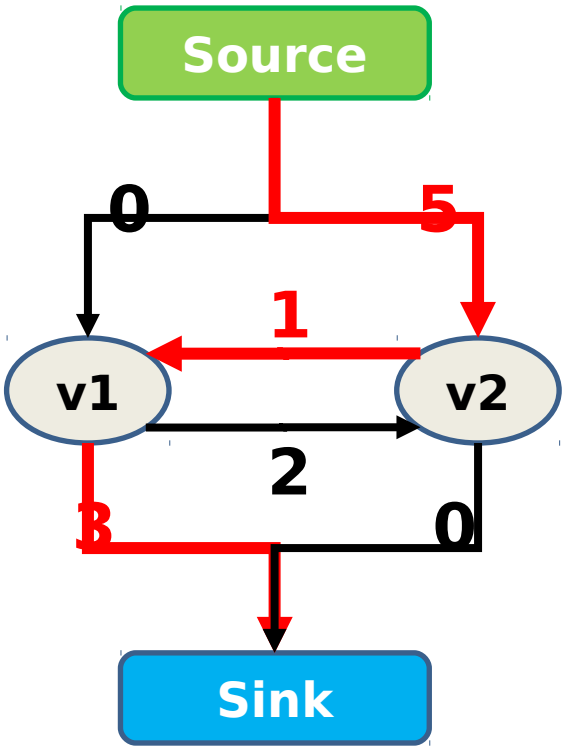
Augmenting Path Based Algorithms

1. Find path from source to sink with positive capacity
2. Push maximum possible flow through this path
3. Repeat until no path can be found

Algorithms assume non-negative capacity

Maxflow Algorithms

Flow = 6

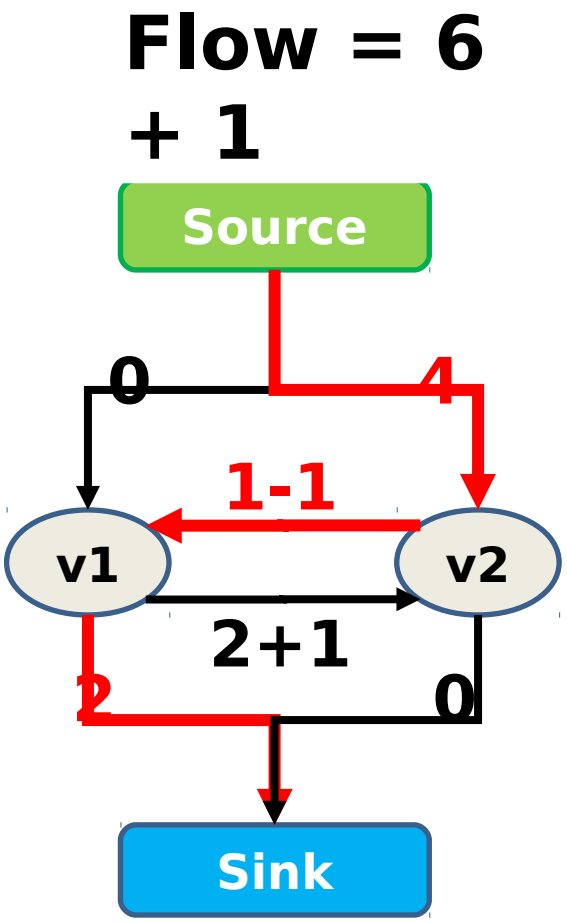


Augmenting Path Based Algorithms

1. Find path from source to sink with positive capacity
2. Push maximum possible flow through this path
3. Repeat until no path can be found

Algorithms assume non-negative capacity

Maxflow Algorithms



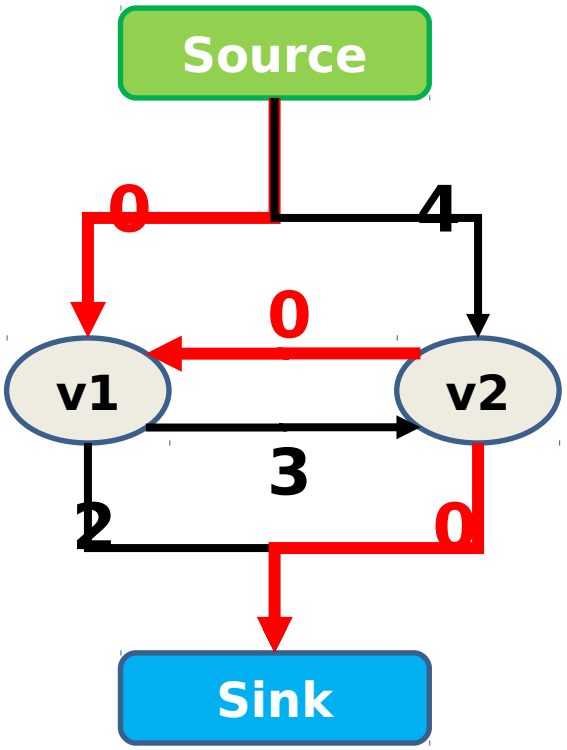
Augmenting Path Based Algorithms

1. Find path from source to sink with positive capacity
2. Push maximum possible flow through this path
3. Repeat until no path can be found

Algorithms assume non-negative capacity

Maxflow Algorithms

Flow = 7

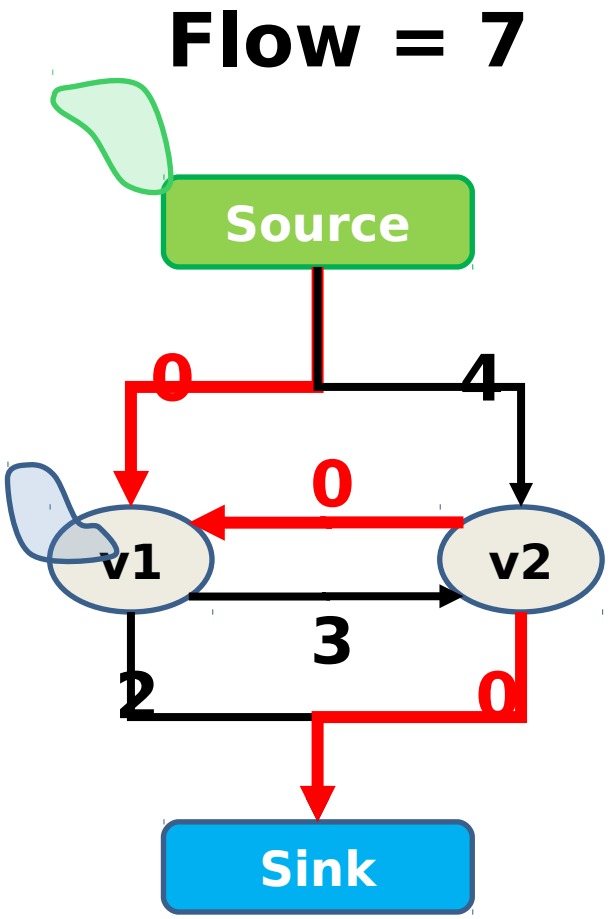


Augmenting Path Based Algorithms

1. Find path from source to sink with positive capacity
2. Push maximum possible flow through this path
3. Repeat until no path can be found

Algorithms assume non-negative capacity

Maxflow Algorithms



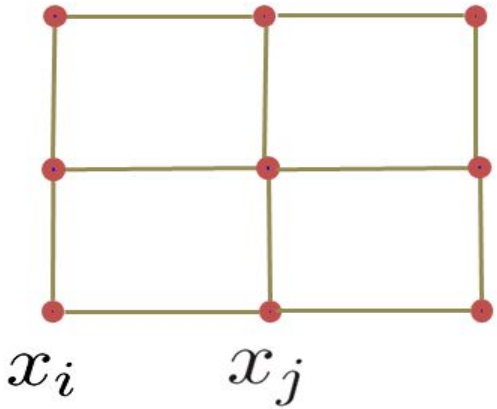
Augmenting Path Based Algorithms

1. Find path from source to sink with positive capacity
2. Push maximum possible flow through this path
3. Repeat until no path can be found

Algorithms assume non-negative capacity

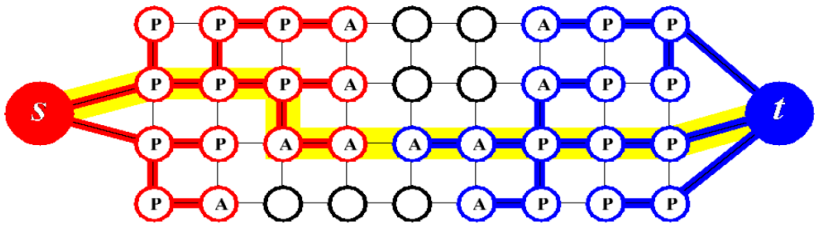
Maxflow in Computer Vision

- **Specialized algorithms for vision problems**
  - Grid graphs
  - Low connectivity (m ~ O(n))



- **Dual search tree augmenting path algorithm**

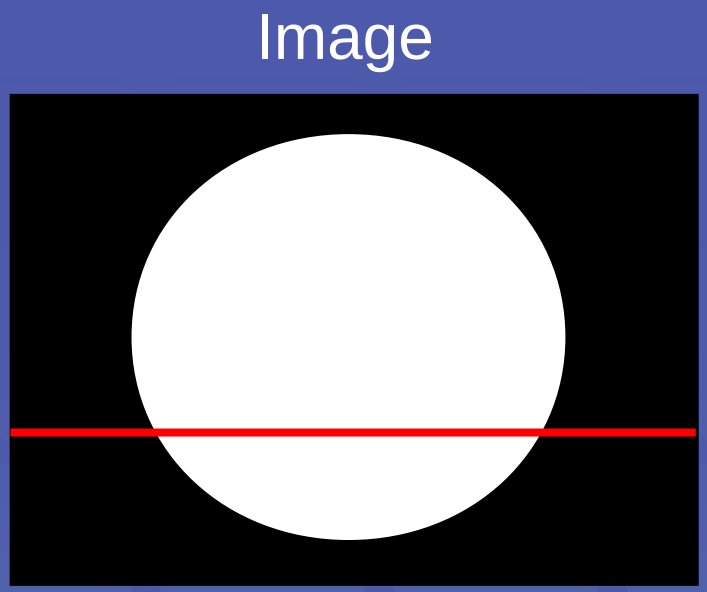
  [Boykov and Kolmogorov PAMI 2004]
  - Finds approximate shortest augmenting paths efficiently
  - High worst-case time complexity
  - Empirically outperforms other algorithms on vision problems

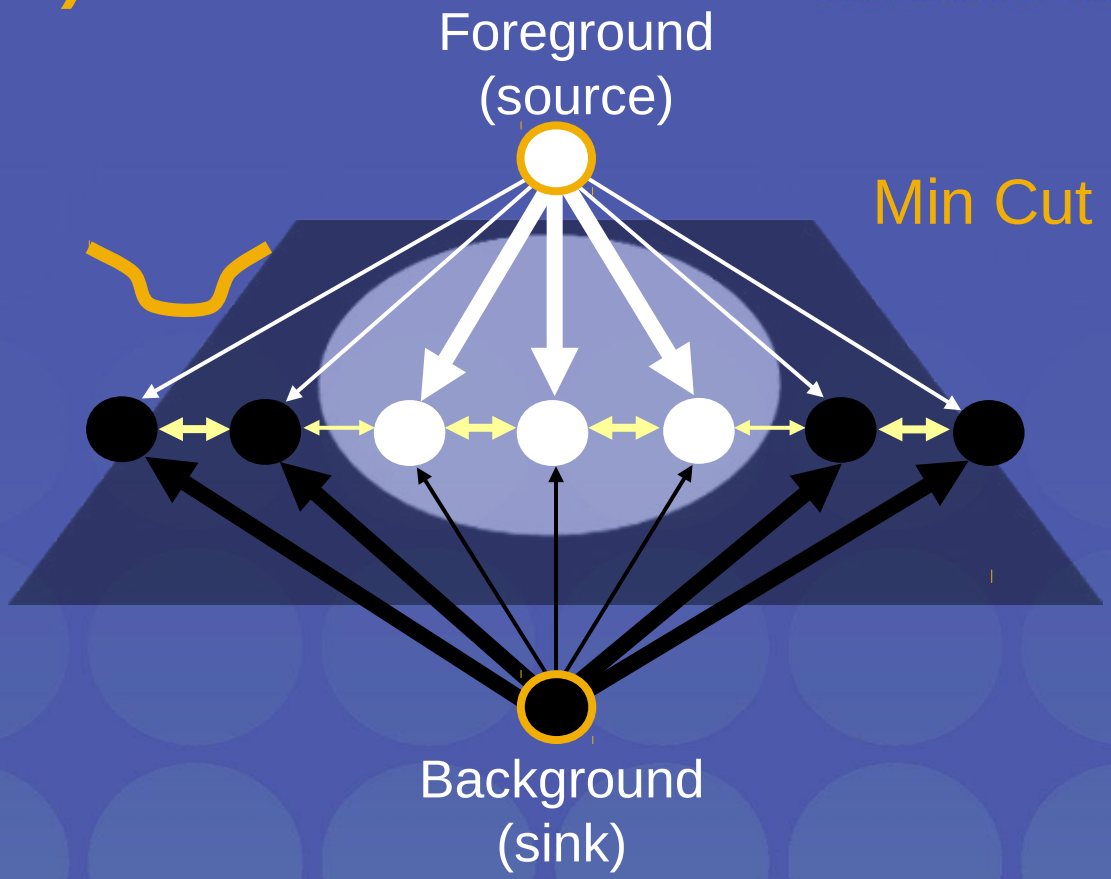


Multilabel: binary choice at a time

Graph Cuts

Boykov and Jolly (2001)





Cut: separating source and sink; Energy: collection of edges

Min Cut: Global minimal energy in polynomial time

Iterated Graph Cut

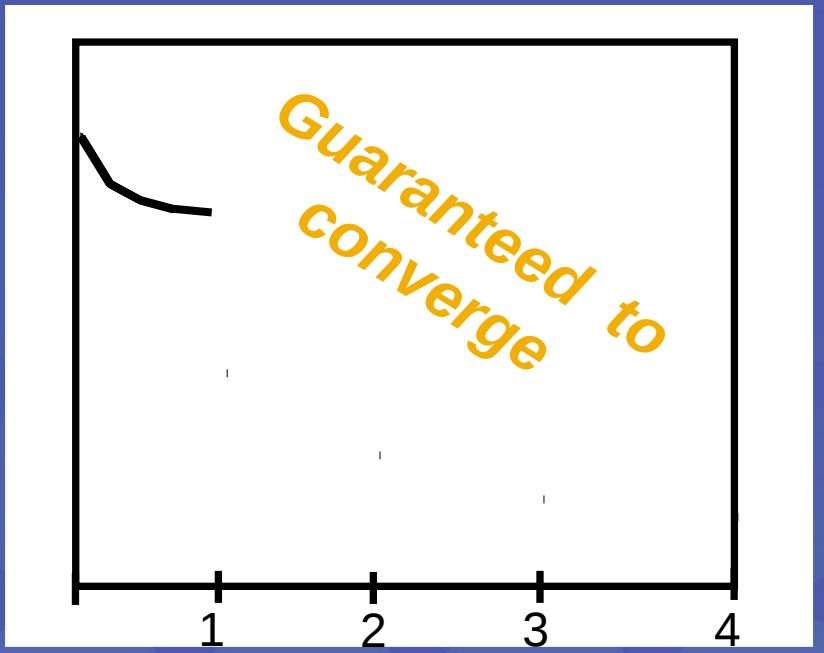
User Initialisation

K-means for learning colour distributions

Graph cuts to infer the segmentation

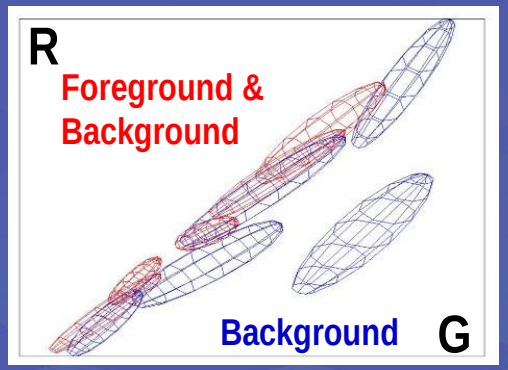
Iterated Graph Cuts

Result

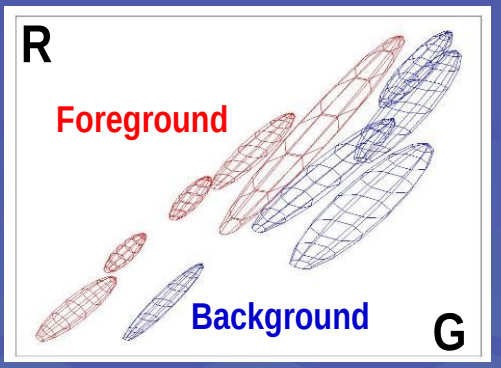


Energy after each Iteration

Colour Model

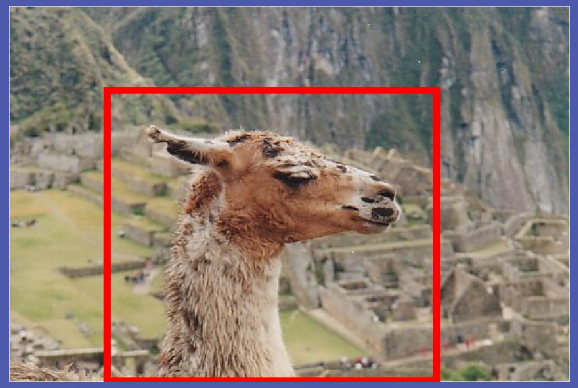


Iterated
graph cut



Gaussian Mixture Model (typically 5-8 components)

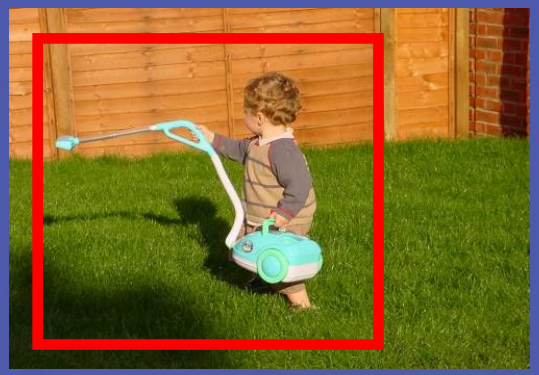
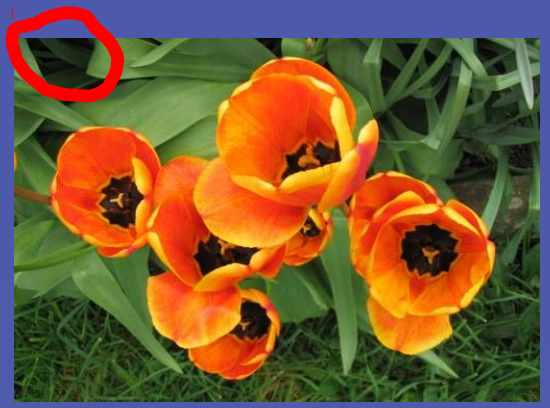
Coherence Model



An object is a coherent set of pixels:

Blake et al. (2004): Learn jointly

Moderately straightforward examples



… GrabCut completes automatically

Difficult Examples

SIGGRAPH2004

Camouflage & Low Contrast

Fine structure

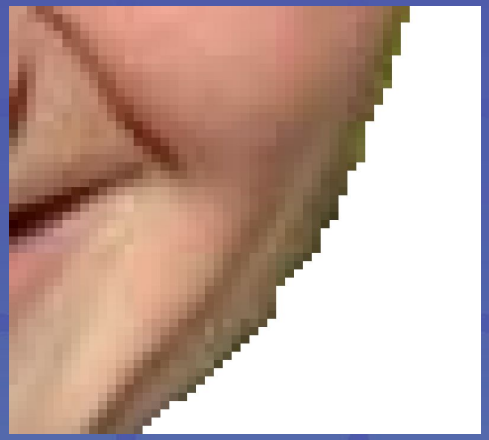
No telepathy

Initial Rectangle

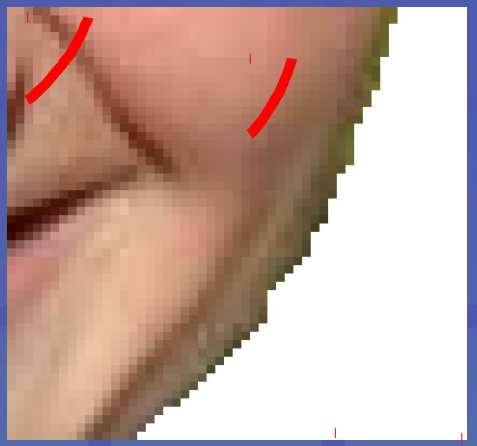
Initial Result

Border Matting

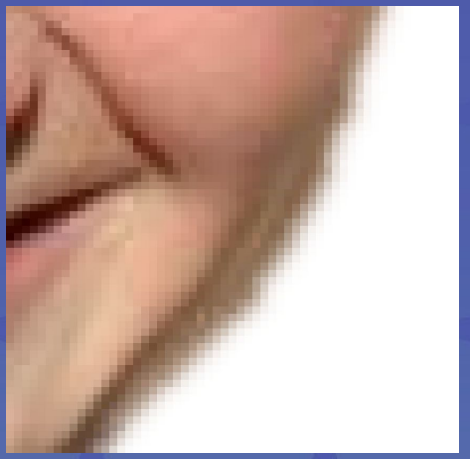
Dave covers matting next week



Hard Segmentation



Automatic Trimap



Soft Segmentation

Software pointers

- http://opencv.itseez.com/modules/imgproc/doc/miscellaneous_grabcut
- http://www.morethantechnical.com/2010/05/05/bust-out-your-o
- http://www.csd.uwo.ca/~olga/code.html
- http://vision.middlebury.edu/MRF/eccv06/

Fast object cut-out

- Minimize the interaction
  - only a few, imprecise strokes

Video Demo

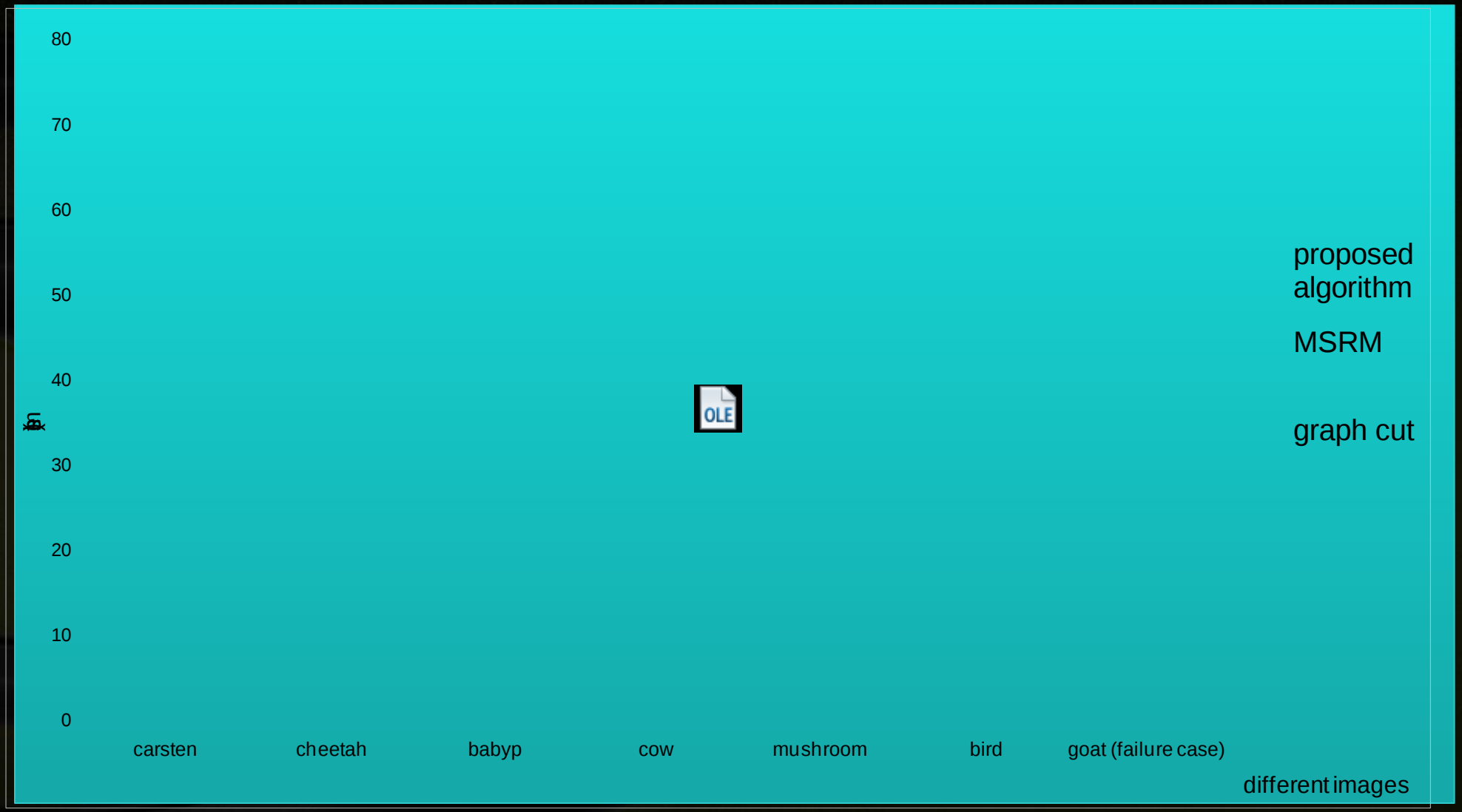
Segmentation time comparison

Table 1. Comparison of region labelling time of the three algorithms

Time (s)	Man	Cheetah	Baby	Cow	Mushroom	Bird
Number of pre-segmented regions	954	1021	379	344	496	637
Our algorithm	0.203	0.313	0.078	0.062	0.125	0.079
MSRM	22.56	48.342	5.391	6.482	15.039	9.143
Graph cuts	30.607	72.137	10.47	7.79	15.808	12.923

MAXIMAL SIMILAR REGION MERGING

Segmentation time comparison



Segmentation time comparison

Table 3. Time measurement on Nokia N900

Time (s)	Man	Cheetah	Baby	Cow	Mushroom	Bird
Number of pre-segmented regions	954	1021	379	344	496	637
Our algorithm	1.73	4.28	0.95	0.77	1.58	0.77

Overview of the algorithm

1. Pre-segmentation by the Mean-Shift algorithm
2. Merge regions by discriminative clustering
3. Local neighborhood region classification and pruning

Dingding Liu, Kari Pulli, Linda Shapiro, Yingen Xiong
Fast Interactive Image Segmentation by Discriminative Clustering
ACM Workshop on Cloud Media Computing (at ACM Multimedia), 2010.

Segmentation quality comparison

(a) Input image

(b) Graph-cut over regions

(c) Maximal Similar Region Merging

(d) Proposed method

Segmentation quality comparison

(a) Input image

(b) Graph-cut over regions

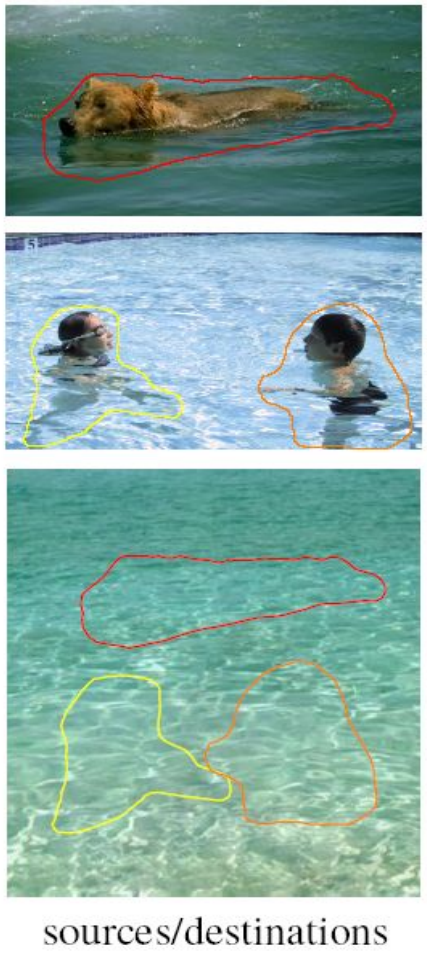
(c) Maximal Similar Region Merging

(d) Proposed method

POISSON IMAGE EDITING

Problems with direct cloning

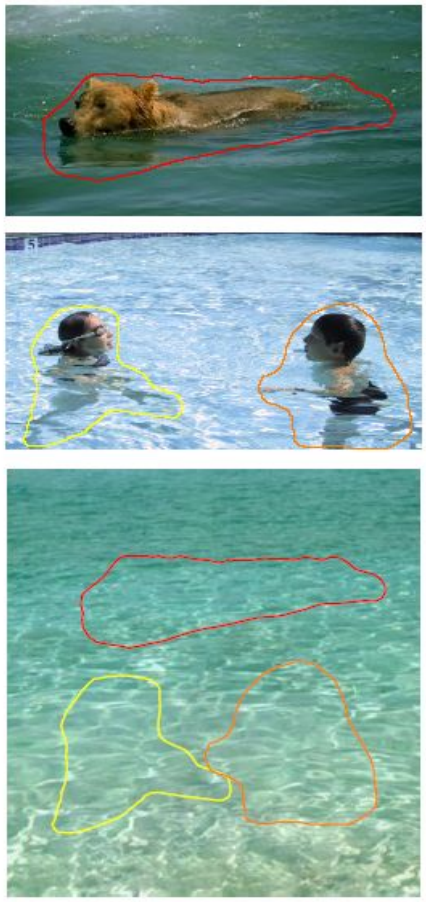
P. Pérez, M. Gangnet, A. Blake. Poisson image editing. SIGGRAPH 2003
http://www.irisa.fr/vista/Papers/2003_siggraph_perez.pdf



sources/destinations

cloning

Solution: clone gradient, integrate colors



sources/destinations

cloning

seamless cloning

Seamless Poisson cloning

- **Given vector field *v* (pasted gradient), find the value of *f* in unknown region that optimizes:**

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

Poisson equation with Dirichlet conditions

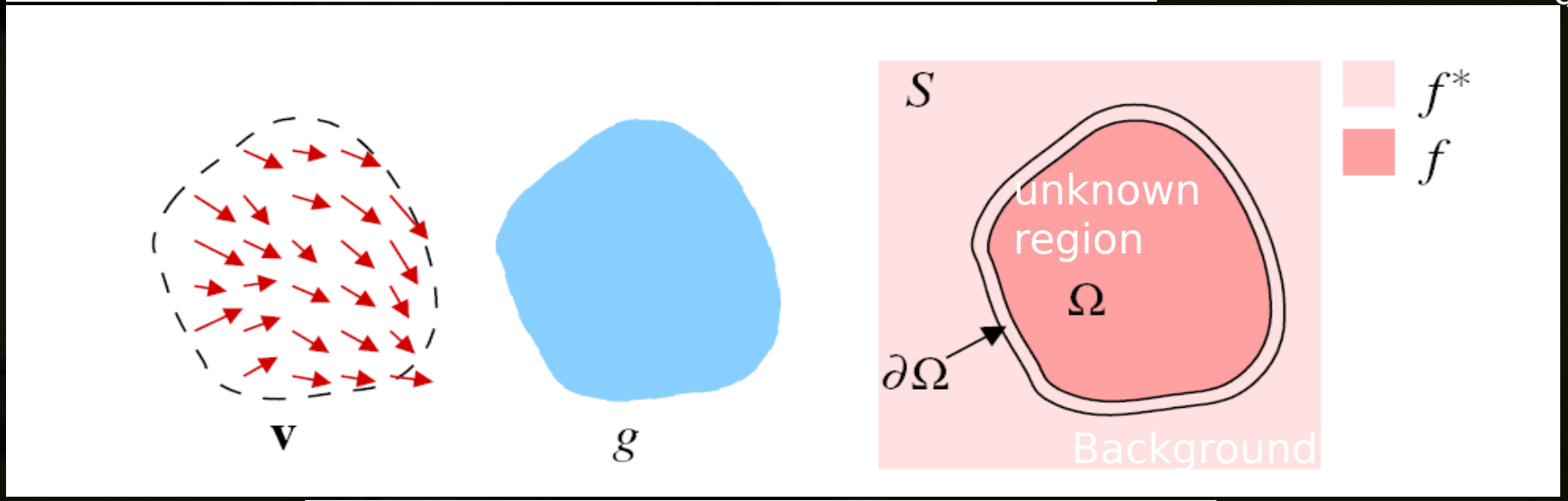


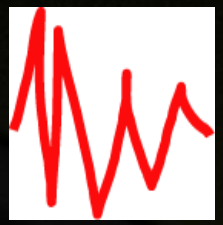
Figure 1: **Guided interpolation notations**. Unknown function f interpolates in domain Ω the destination function f^*, under guidance of vector field $\mathbf{v}$, which might be or not the gradient field of a source function g.

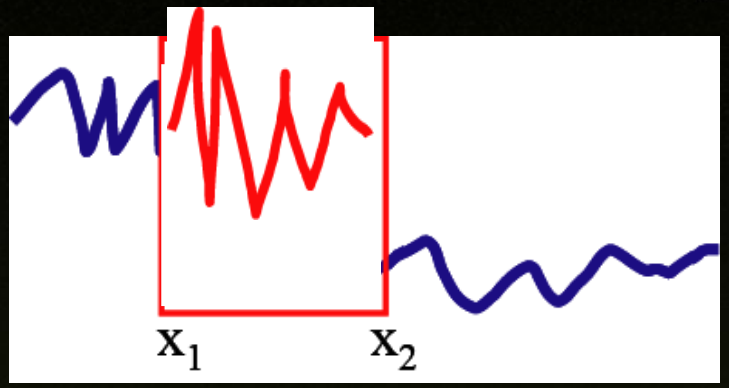
Membrane interpolation

- What if *v* is null?
- Laplace equation (a.k.a. membrane equation)

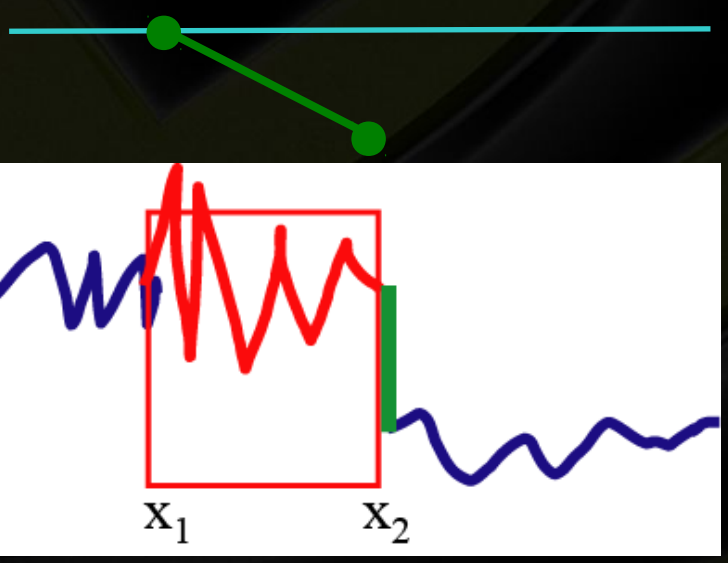
$$\min_f \iint_\Omega |\nabla f|^2 \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

What if *v* is not null

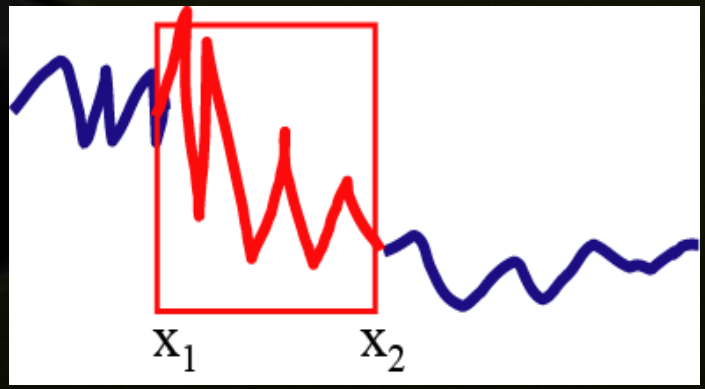
Seamlessly paste  onto



Just add a linear function so that the boundary condition is respected



Gradients didn't change much, and function is continuous



Guided Interpolation

- Find *f* so its gradient matches v and the boundary

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

- This solution is the unique solution of following Poisson equation with Dirichlet boundary condition:

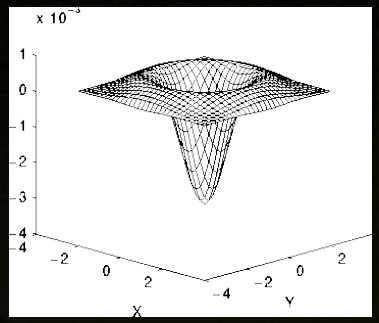
$$\Delta f = \operatorname{div} \mathrm{v} \quad \text{over } \Omega, \qquad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

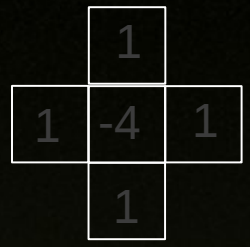
$$\text{where } \operatorname{div} \mathrm{v} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \text{ is the divergence of } \mathrm{v} = (u, v)$$

- Three Poisson equations of this form are solved independently in three color channels

Discretize

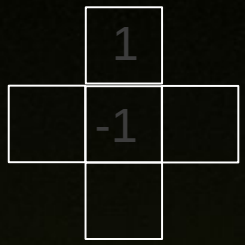
- Laplacian of f



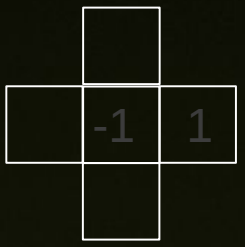


- Source image g, its gradient field is v = (g_x, g_y)

g_y



g_x



- Divergence of v = $u_x + v_y = g_{xx} + g_{yy}$

g_{yy}

g_{xx}

A giant linear system

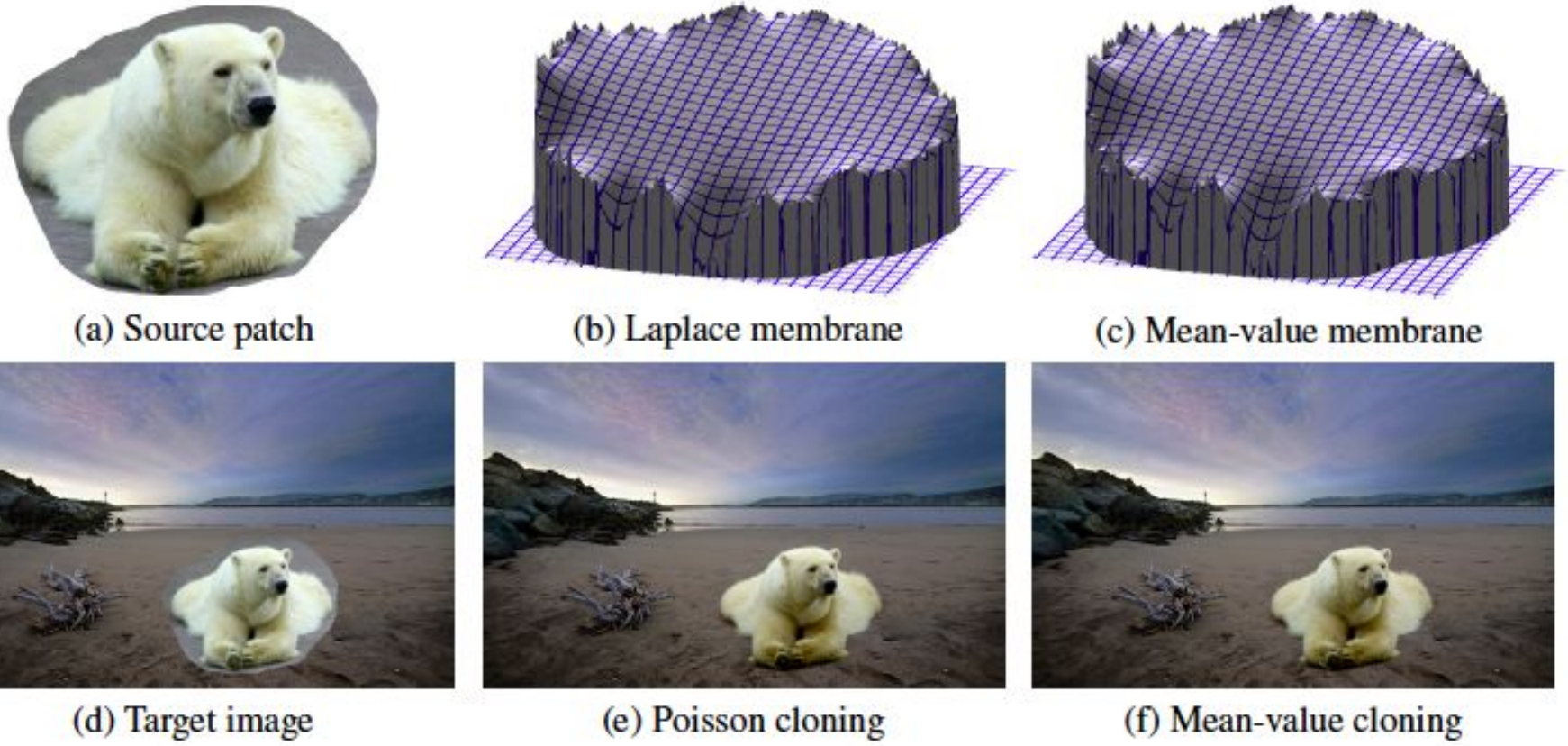
- A f = b f is the pixels in Omega, as a big vector
- Boundary condition
  - a row in A with [0 … 0 1 0 … 0]
  - the same row on b has the fixed value for that pixel in f
- Internal pixels
  - a row in A with all zeros except four 1's and one -4
  - the matching row in b has the Laplacian of g
- Too big to solve directly
- Lots of iterative solvers
  - preconditioned conjugate gradients
  - Gauss-Seidel with successive overrelaxation
  - V-cycle multigrid
  - …

There are much simpler ways

Coordinates for Instant Image Cloning

SIGGRAPH 2009

Zeev Farbman, Hebrew University
Gil Hoffer, Tel Aviv University
Yaron Lipman, Princeton University
Daniel Cohen-Or, Tel Aviv University
Dani Lischinski, Hebrew University



(a) Source patch (b) Laplace membrane (c) Mean-value membrane

(d) Target image (e) Poisson cloning (f) Mean-value cloning

Figure 1: *Poisson cloning smoothly interpolates the error along the boundary of the source and the target regions across the entire cloned region (the resulting membrane is shown in (b)), yielding a seamless composite (e). A qualitatively similar membrane (c) may be achieved via transfinite interpolation, without solving a linear system. (f) Seamless cloning obtained instantly using the mean-value interpolant.*

Mean-value coordinates for smooth interpolation

these coordinates may be used to smoothly interpolate any function f defined at the boundary vertices:

$$\tilde{f}(\mathbf{x}) = \sum_{i=0}^{m-1} \lambda_i(\mathbf{x}) f(\mathbf{p}_i). \qquad (3)$$

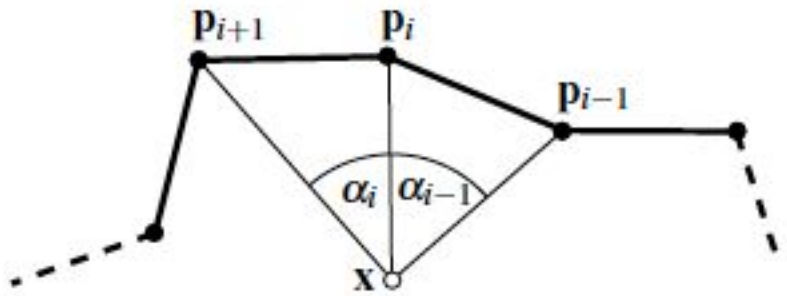


Figure 2: *Angle definitions for mean-value coordinates.*

- **Evaluate them sparsely…**
  - **… and interpolate within triangles**

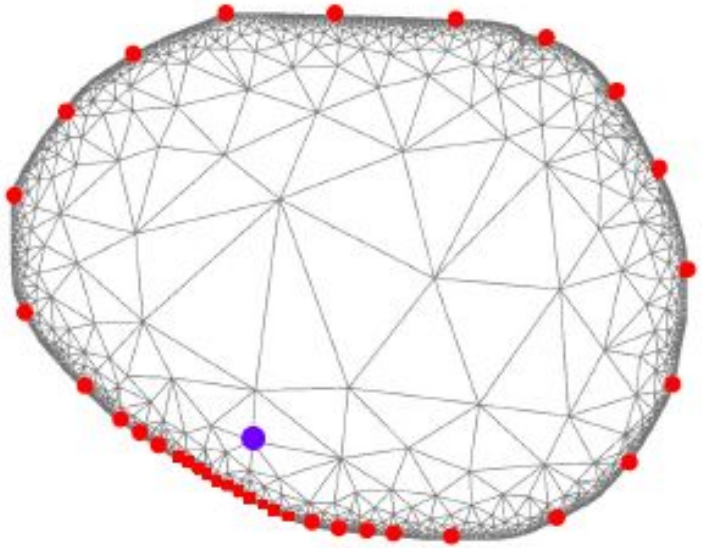


Figure 3: *An adaptive triangular mesh constructed over the region to be cloned. The red dots on the boundary show the positions of boundary vertices that were selected by adaptive hierarchical subsampling for the mesh vertex indicated in blue.*

Interactive Poisson cloning!

Table 1: *Performance statistics for MVC cloning. Times exclude disk I/O and sending the images to the graphics subsystem. Cloning rate is the number of region updates per second.*

#cloned pixels	#bdry pixels	#mesh vertices	coords /vertex	prep. time(s)	cloning rate CPU	GPU
51,820	1,113	2,063	38.63	0.15	199.0	163
133,408	1,562	2,963	44.21	0.30	92.1	134
465,134	2,683	5,323	45.50	0.63	22.6	82
1,076,572	4,145	8,241	44.59	1.16	9.7	44
4,248,461	8,133	16,369	57.71	3.63	2.7	26
12,328,289	14,005	28,240	58.68	8.99	0.94	–

- **Faster than “real” Poisson**

2008], as well as our own experiments, indicate that common Poisson solvers on the CPU are able to handle regions with 256^2 pixels at a rate of 3–5 solutions per second. Another possibility, which we have not seen mentioned in the literature, is to precompute a factorization of the Poisson equation matrix during the preprocessing stage, and then quickly compute the solution via back-substitution at each target location. In our experiments, for a region with 125K pixels, computing the back-substitution takes 0.3 seconds. Thus, all of the above are significantly slower than the rates we are able to achieve.

It gets still faster!

SIGGRAPH ASIA 2011

Convolution Pyramids

Zeev Farbman
The Hebrew University

Raanan Fattal
The Hebrew University

Dani Lischinski
The Hebrew University

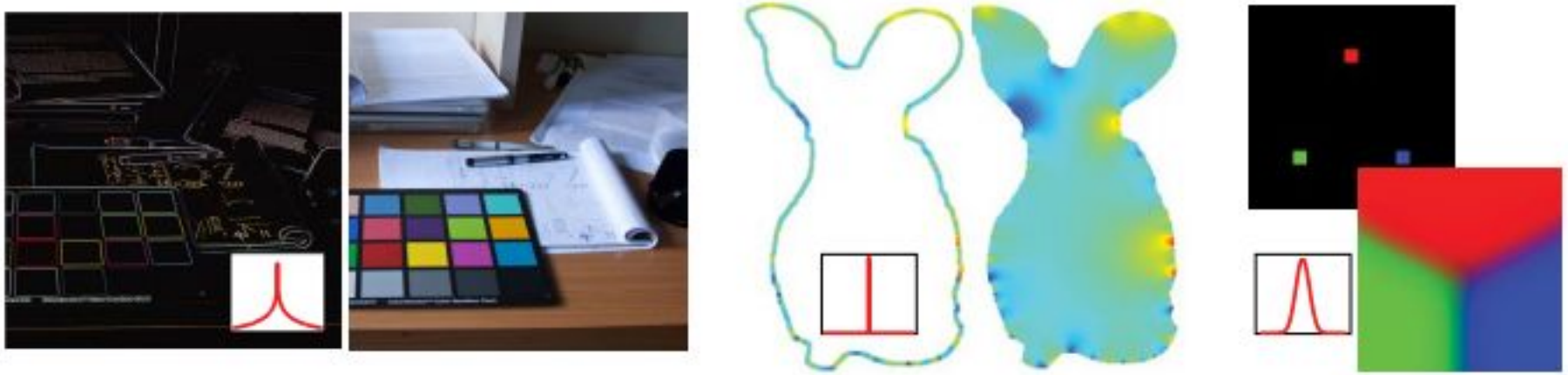


Figure 1: *Three examples of applications that benefit from our fast convolution approximation. Left: gradient field integration; Middle: membrane interpolation; Right: scattered data interpolation. The insets show the shapes of the corresponding kernels.*

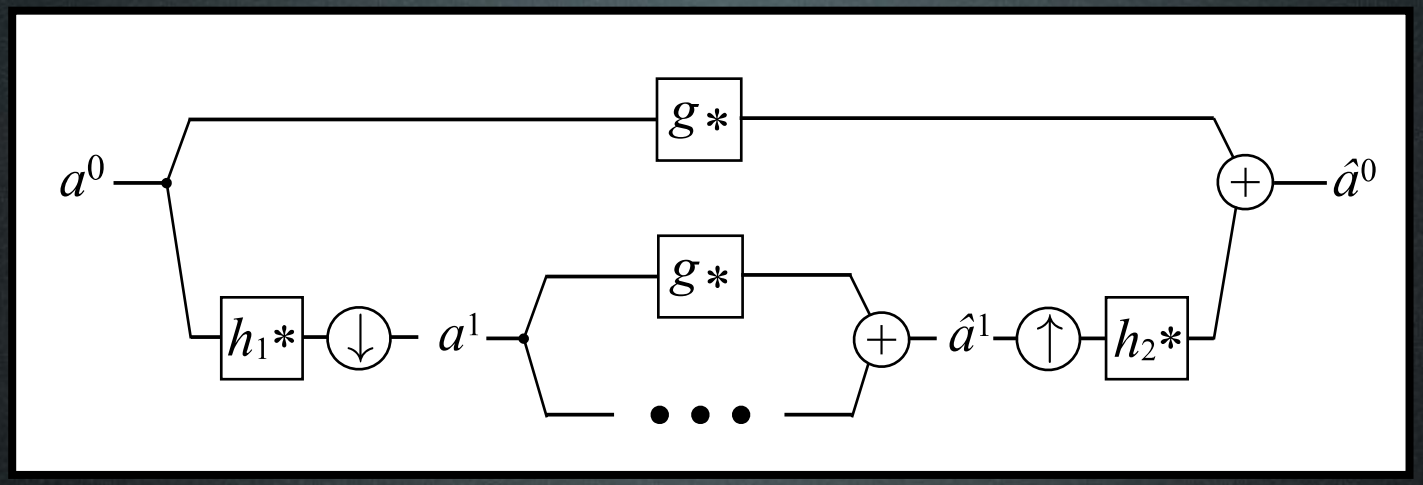
Motivation:
Linear time convolution

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

	Complexity	Implementation	General
Direct Evaluation	$O(N^2)$	Simple	Yes
FFT	$O(NlogN)$	Hard	Yes
Our Method	$O(N)$	Simple	No

Example:
Image from divergence

$f * a$



Subband Architecture

Algorithm 1 *Our multiscale transform*

Determine the number of levels L

{Forward transform (analysis)}

$a^0 = a$

for each level $l = 0 \ldots L-1$ **do**

$a_0^l = a^l$

$a^{l+1} = \downarrow (h_1 * a^l)$

end for

{Backward transform (synthesis)}

$\hat{a}^L = g * a^L$

for each level $l = L-1 \ldots 0$ **do**

$\hat{a}^l = h_2 * (\uparrow \hat{a}^{l+1}) + g * a_0^l$

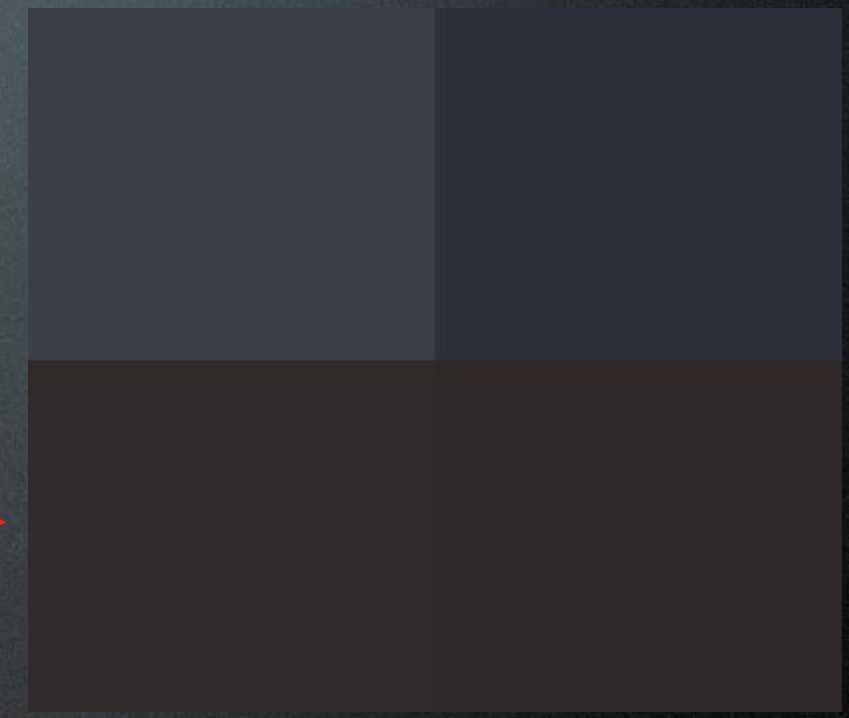
end for

Analysis

Synthesis

Multiscale Transform

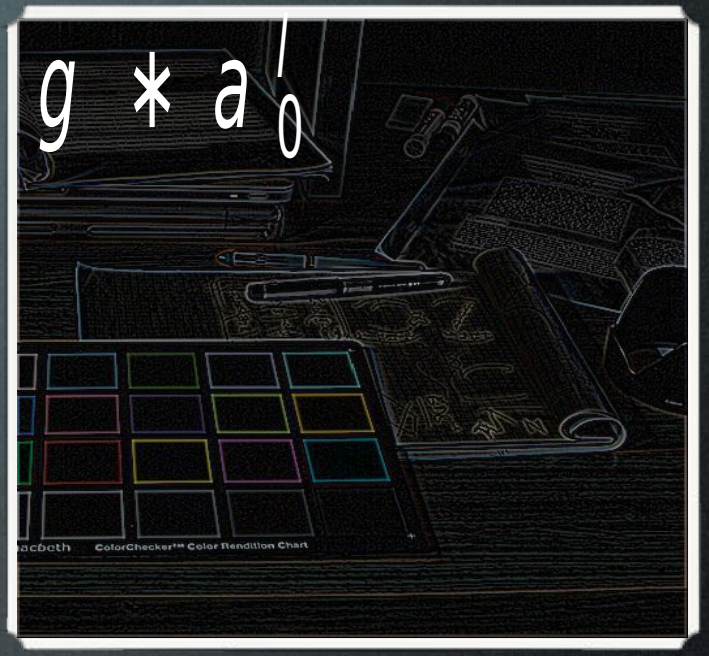
Determine the number of levels L
{Forward transform (analysis)}
$a^0 = a$
for each level $l = 0 \ldots L-1$ do
  $a_0^l = a^l$
  $a^{l+1} = \downarrow(h_1 * a^l)$
end for



Analysis

a^{l+1}

+



=

{Backward transform (synthesis)}
$\hat{a}^L = g * a^L$
for each level $l = L - 1 \ldots 0$ do
$\hat{a}^l = h_2 * (\uparrow \hat{a}^{l+1}) + g * a_0^l$
end for

Synthesis

Optimization

$$\mathcal{F} = \{h_1, h_2, g\}$$

$$\arg\min_{\mathcal{F}} \|\mathcal{F}(a) - f * a\|_2$$

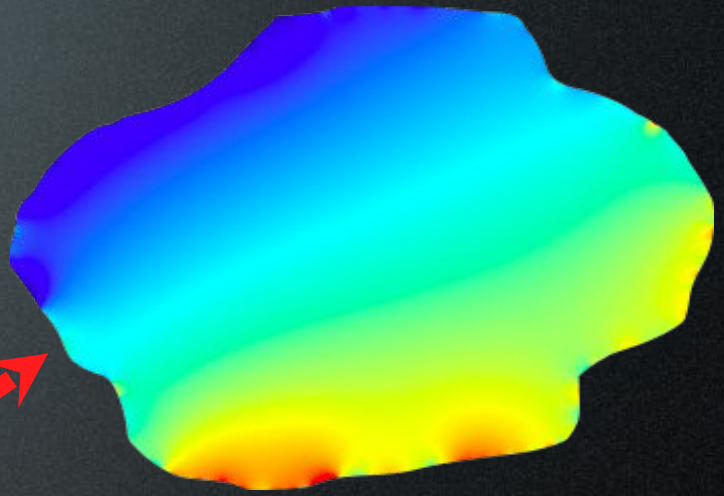
Optimization

$$\mathcal{F} = \{h_1, h_2, g\}$$

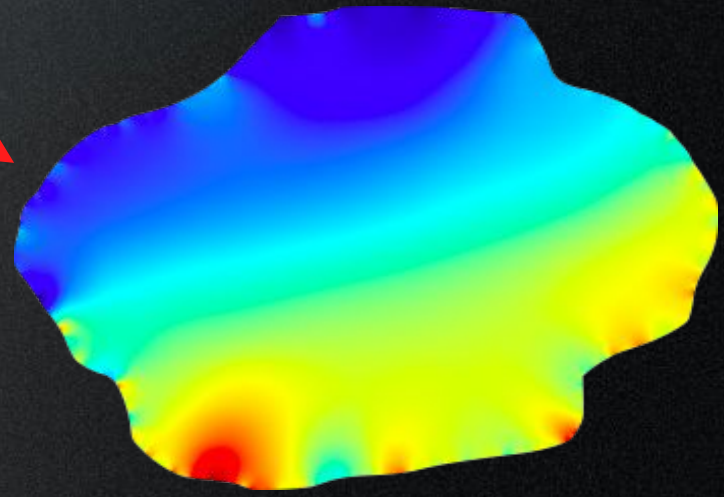
$$\arg\min_{\mathcal{F}} \|\mathcal{F}(a) - f * a\|_2$$

Seamless Cloning

Target Image



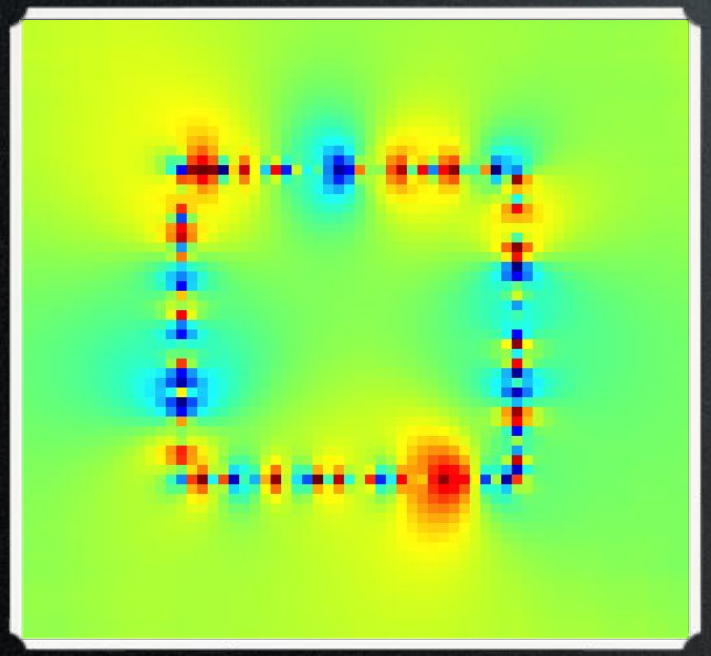
Laplacian Membrane

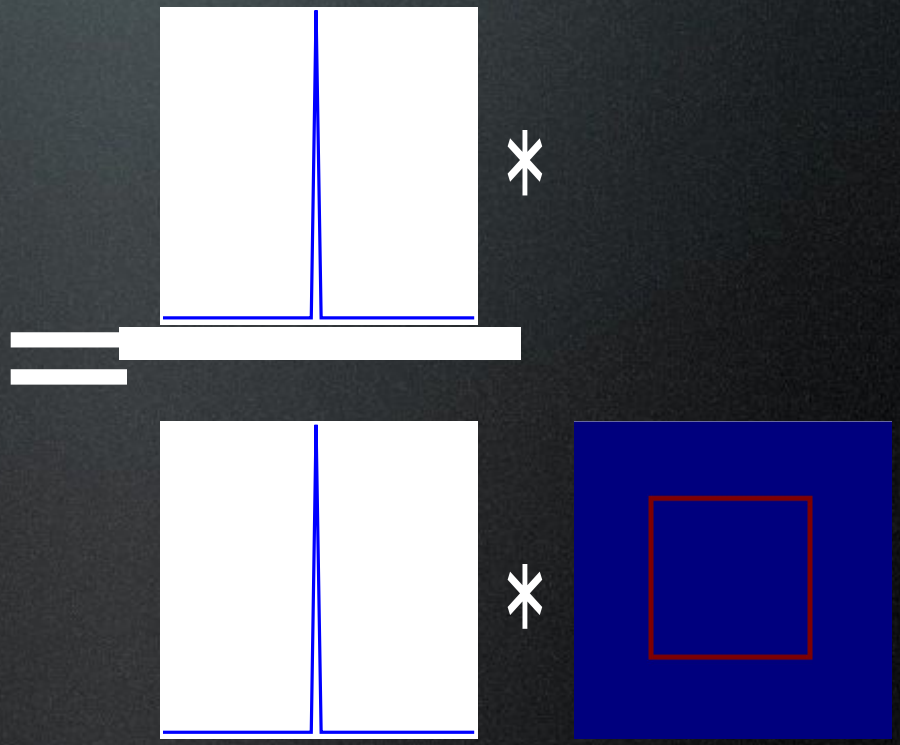


MVC Membrane

Boundary Interpolation

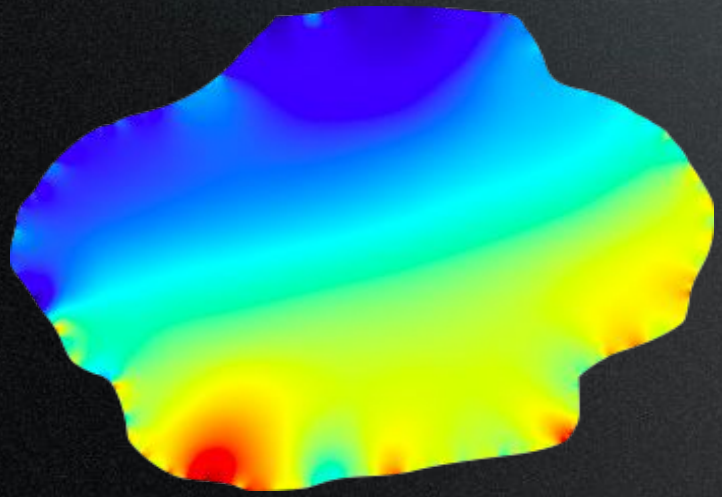
$$r(\mathbf{x}) = \frac{w * \hat{r}}{w * \chi_{\hat{r}}} \qquad w_k(\mathbf{x}) = w(\mathbf{x}_k, \mathbf{x}) = 1/d(\mathbf{x}_k, \mathbf{x})^3$$



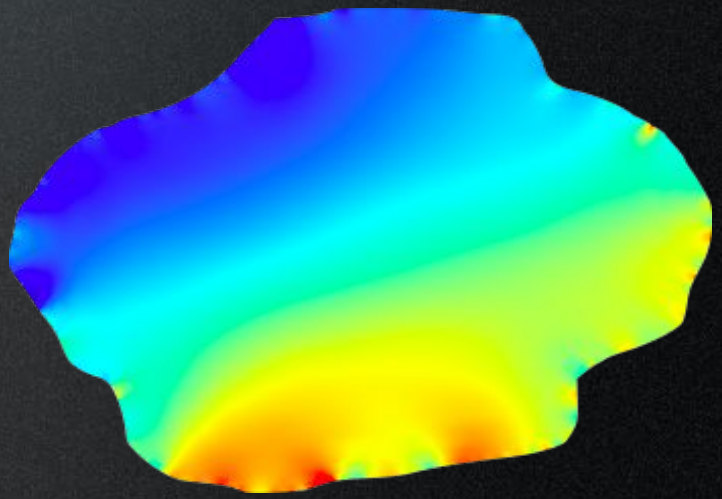


Boundary Interpolation

Source Patch + Target Image



MVC



Convolution Pyramid

Boundary Interpolation

Source Patch + Target Image

MVC

Convolution Pyramid

Running times for seamless cloning

Table 1 summarizes the running times of our optimized CPU implementation for kernel sets $\mathcal{F}_{5,3}$ and $\mathcal{F}_{7,5}$ on various image sizes.

grid size (millions)	time (5x5/3x3) (sec, single core)	time (7x7/5x5) (sec, single core)
0.26	0.0019	0.00285
1.04	0.010	0.015
4.19	0.047	0.065
16.77	0.19	0.26
67.1	0.99	1.38

Table 1: *Performance statistics for convolution pyramids. Reported times exclude disk I/O and were measured on a 2.3GHz Intel Core i7 (2820qm) MacBook Pro.*

hierarchical sampling of the boundary. Farbman *et al.* [2009] report that after 3.6 seconds of preprocessing time, it takes 0.37 seconds to seamlessly clone a 4.2 megapixel region (on a 2.5 MHz Athlon CPU). In comparison, our method does not involve preprocessing, and clones the same number of pixels in 0.15 seconds.

Basic application: insertion

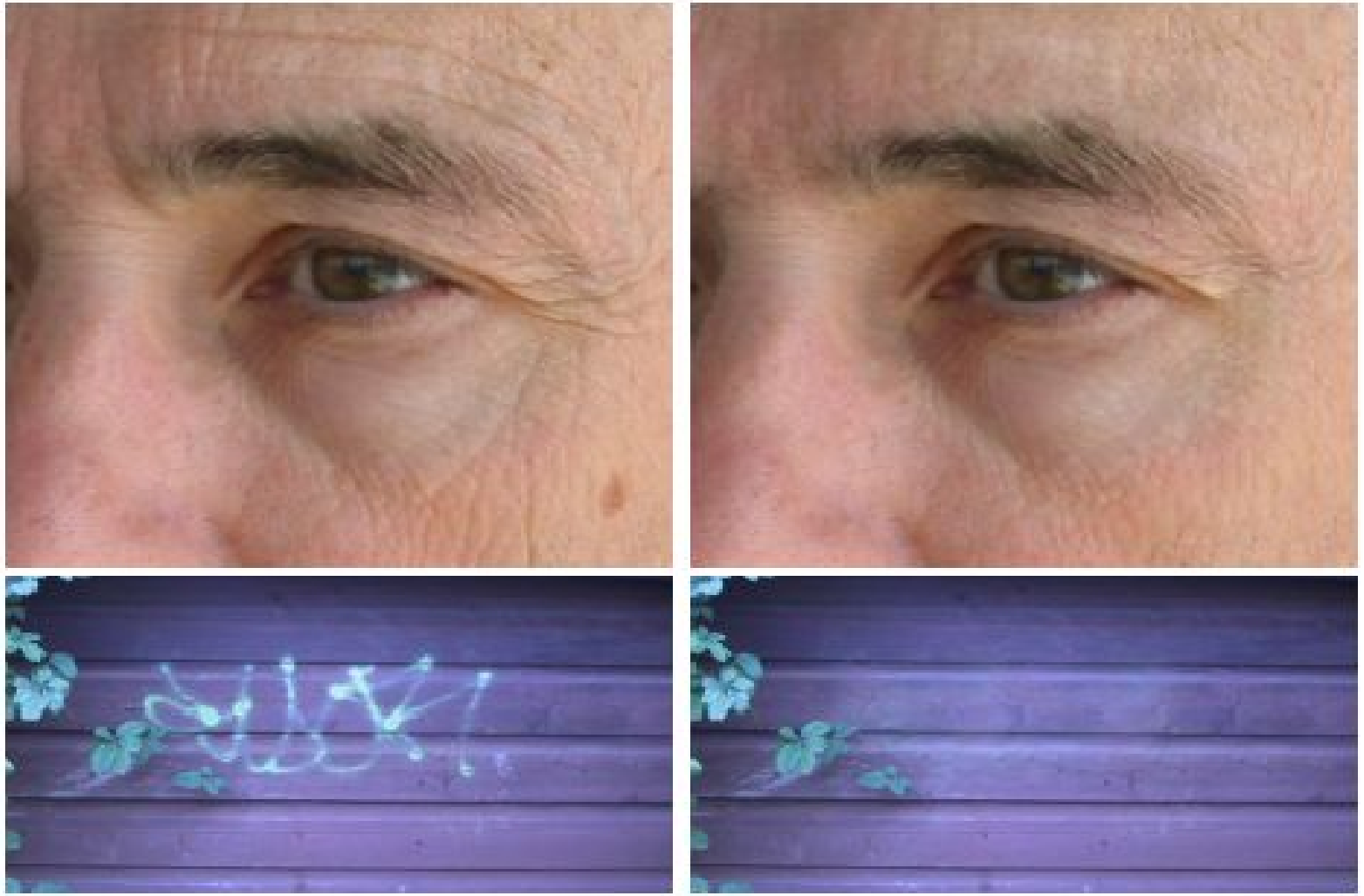
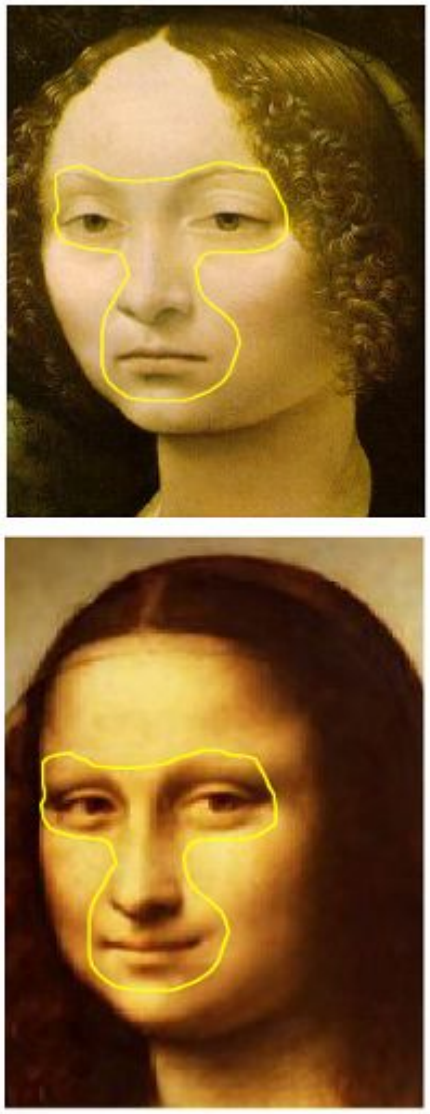


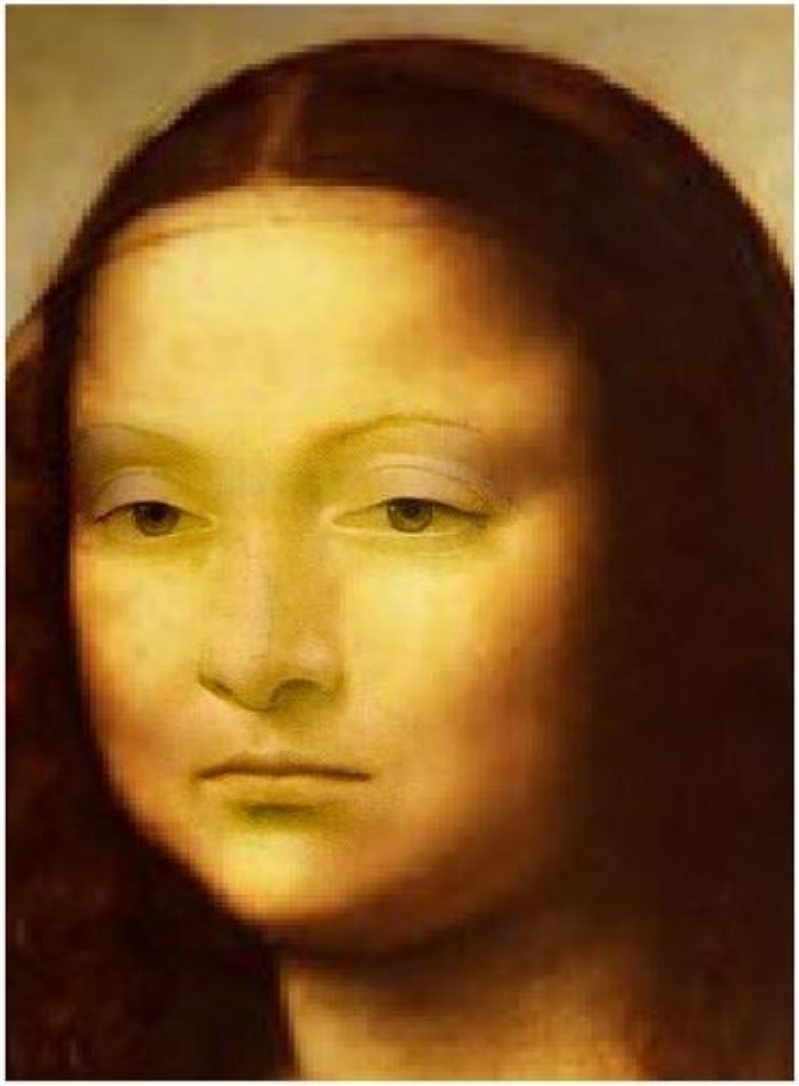
Figure 2: **Concealment**. By importing seamlessly a piece of the background, complete objects, parts of objects, and undesirable artifacts can easily be hidden. In both examples, multiple strokes (not shown) were used.

Feature exchange



source/destination

cloning



seamless cloning

Problems (non-homogeneous background) and solutions

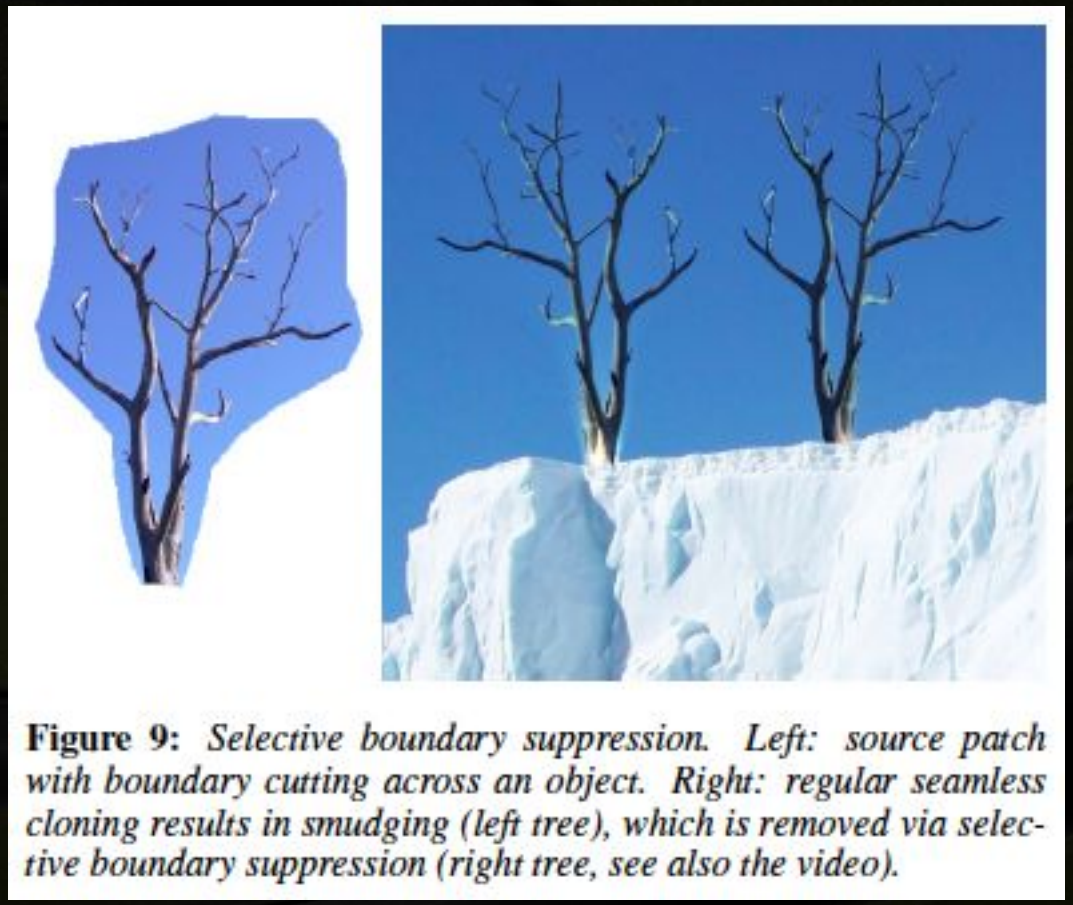


Figure 9: *Selective boundary suppression. Left: source patch with boundary cutting across an object. Right: regular seamless cloning results in smudging (left tree), which is removed via selective boundary suppression (right tree, see also the video).*

(a) input image (b) trimap

(c) Cloning the eagle over a non-homogeneous image.

(d) Applying a matte to the cloned region.

Additional tricks

- The following ones require really solving the Poisson equation
  - instead of taking the fast shortcut of seamless cloning

Manipulate the gradient

- Mix gradients of g & f: take the max

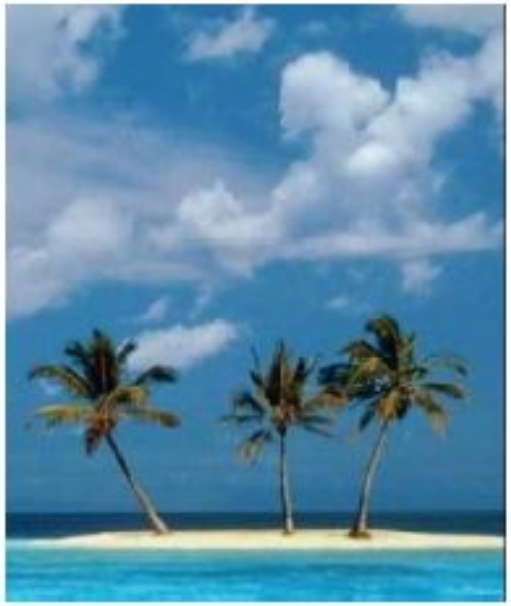
source/destination

seamless cloning

mixed seamless cloning

Figure 8: **Inserting one object close to another**. With seamless cloning, an object in the destination image touching the selected region Ω bleeds into it. Bleeding is inhibited by using mixed gradients as the guidance field.

source



destination

Figure 7: **Inserting transparent objects**. Mixed seamless cloning facilitates the transfer of partly transparent objects, such as the rainbow in this example. The non-linear mixing of gradient fields picks out whichever of source or destination structure is the more salient at each location.

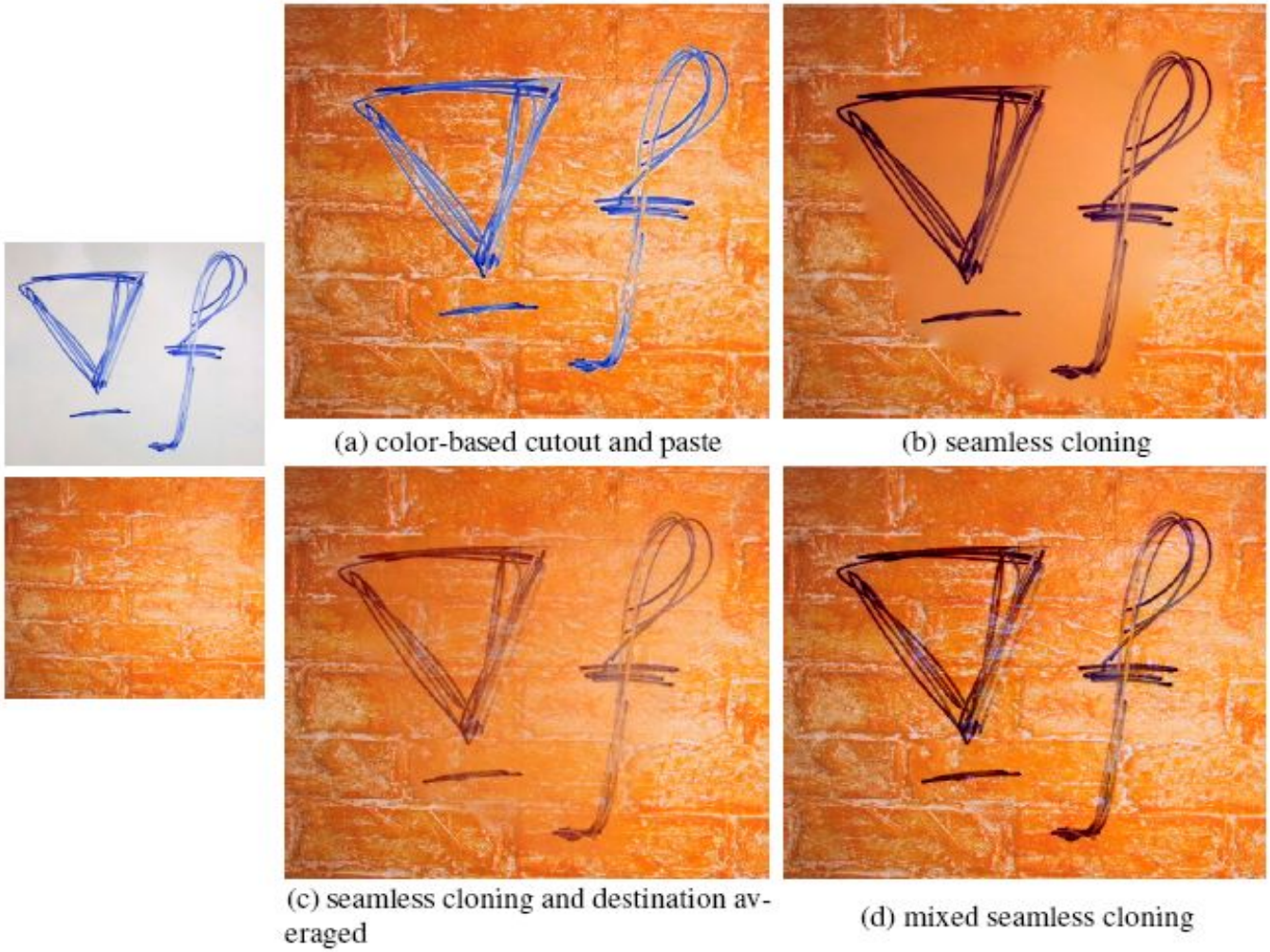
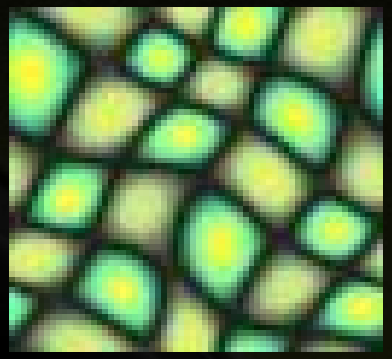


Figure 6: **Inserting objects with holes**. (a) The classic method, color-based selection and alpha masking might be time consuming and often leaves an undesirable halo; (b-c) seamless cloning, even averaged with the original image, is not effective; (d) mixed seamless cloning based on a loose selection proves effective.

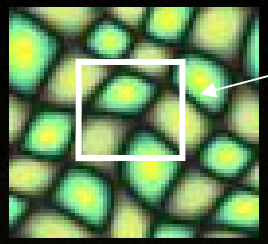
Cheaper seam finding

- Calculating seams using accurate segmentation methods such as graph cut / grabcut is an overkill
- If we can do fast seamless cloning
  - find a seam somewhere that foreground and background match
    - the segmentation doesn't have to be so accurate
  - find the seams quickly using dynamic programming
  - then cut there and apply fast seamless cloning

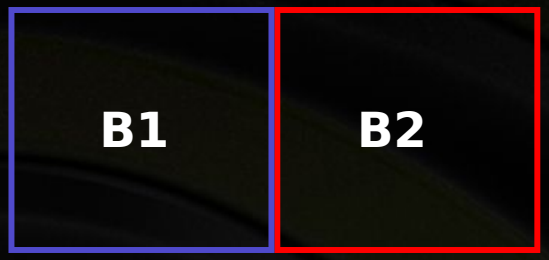
Aside: [Efros & Freeman SIGGRAPH 2001] Image Quilting for Texture Synthesis and Transfer



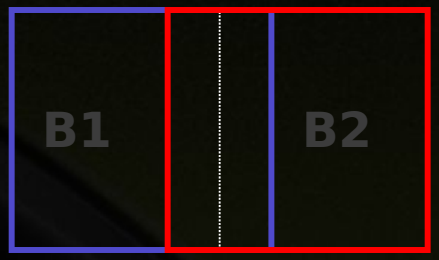
- The “Corrupt Professor’s Algorithm”:
  - Plagiarize as much of the source image as you can
  - Then try to cover up the evidence
- Rationale:
  - Texture blocks are by definition correct samples of texture so the main problem is connecting them together



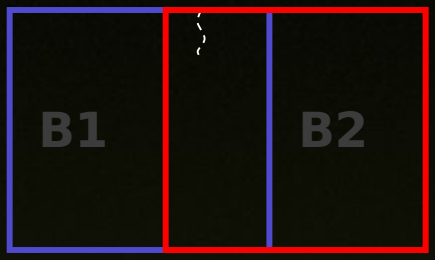
Input texture



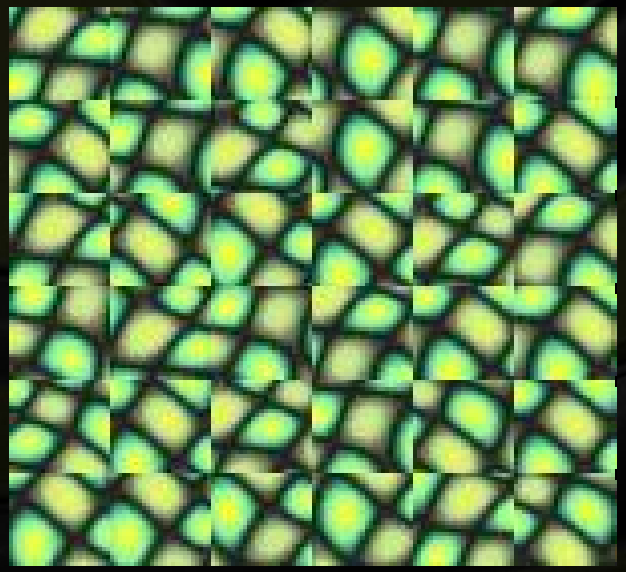
Random placement of blocks

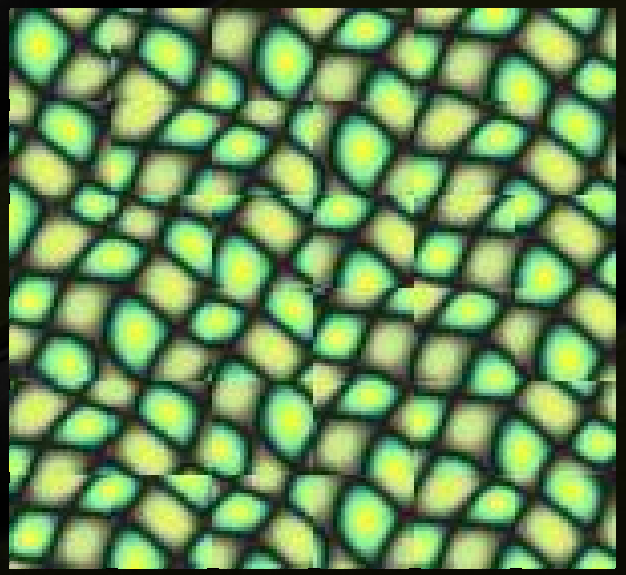


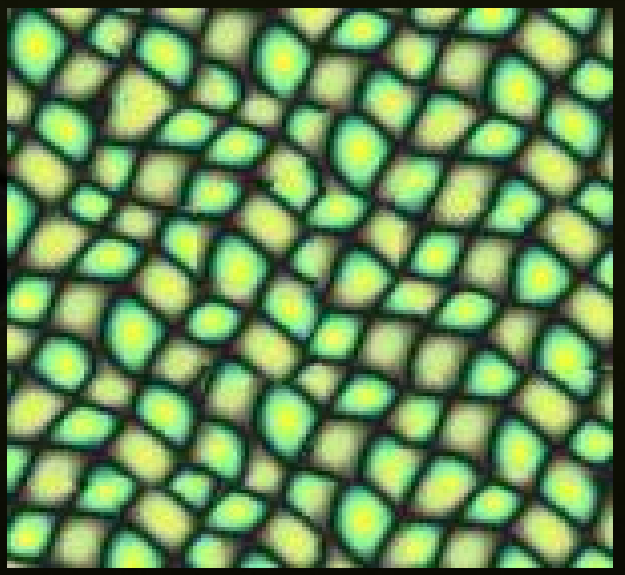
Neighboring blocks constrained by overlap



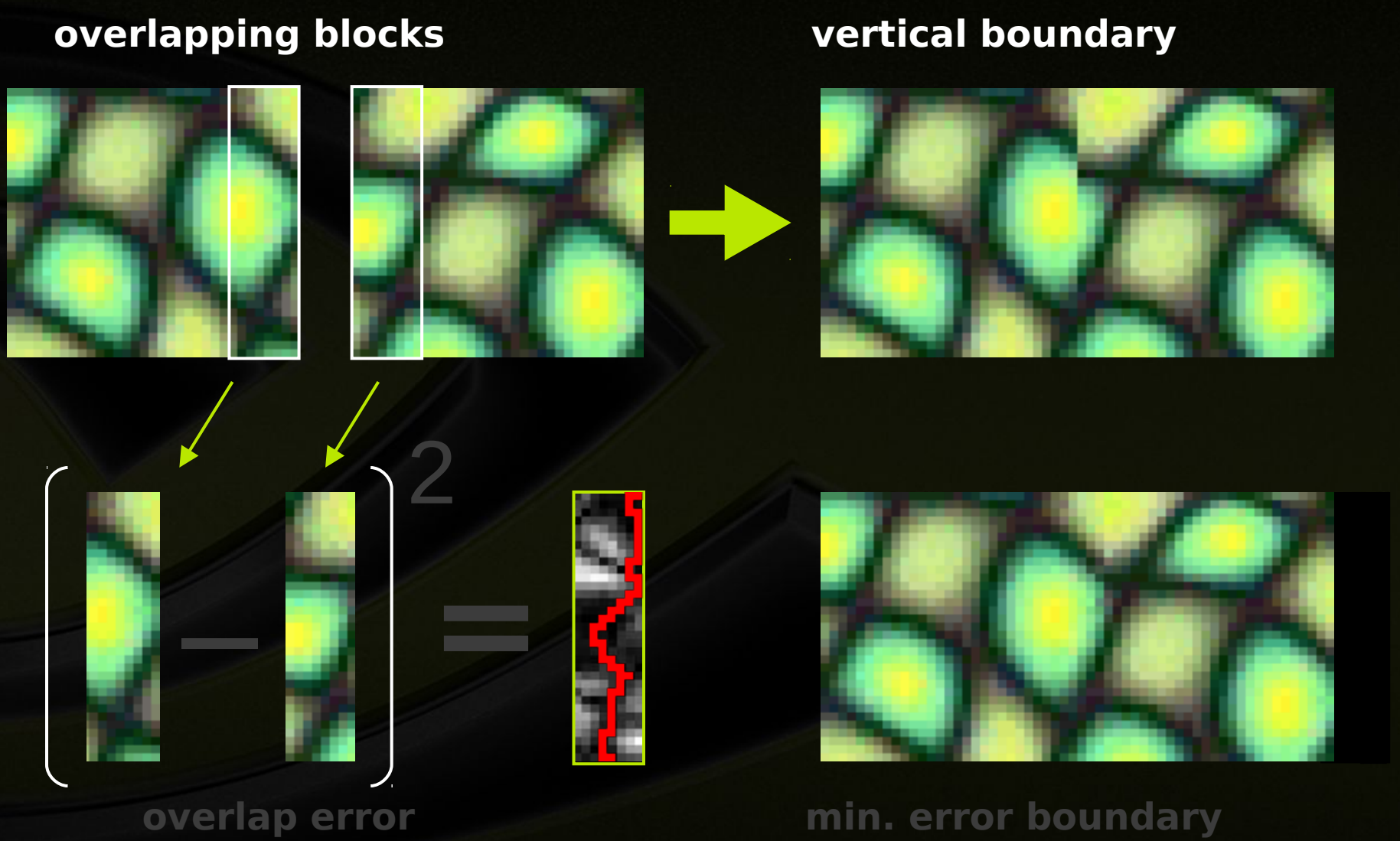
Minimal error boundary cut





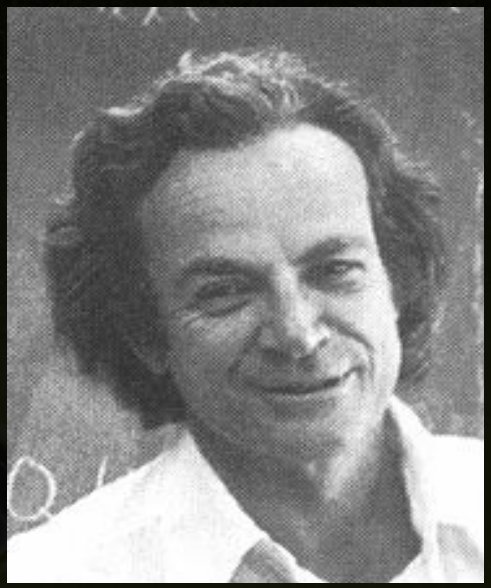


Minimal error boundary with DP



Texture Transfer

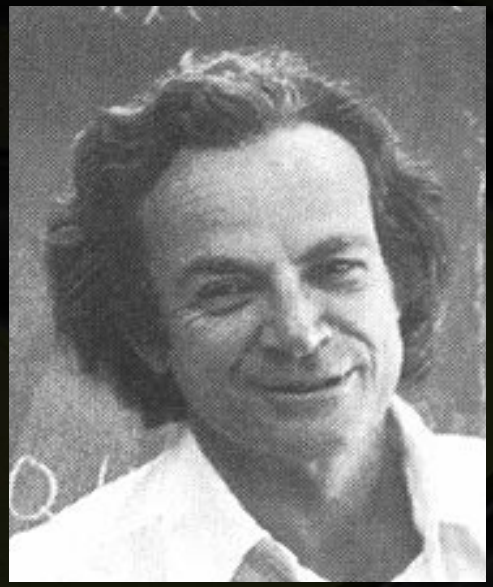
Source texture



Target image

Source correspondence image

Target correspondence ima

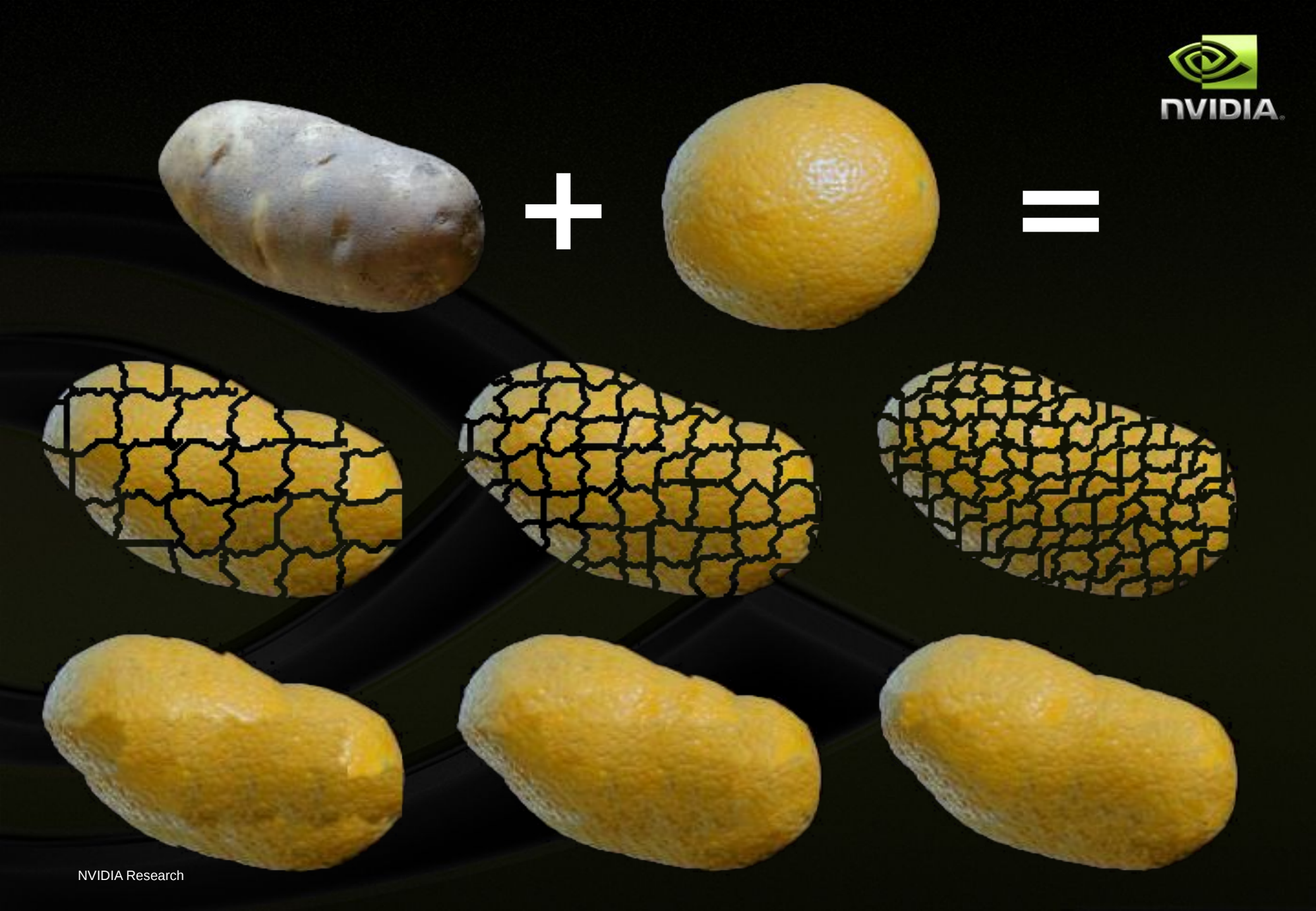


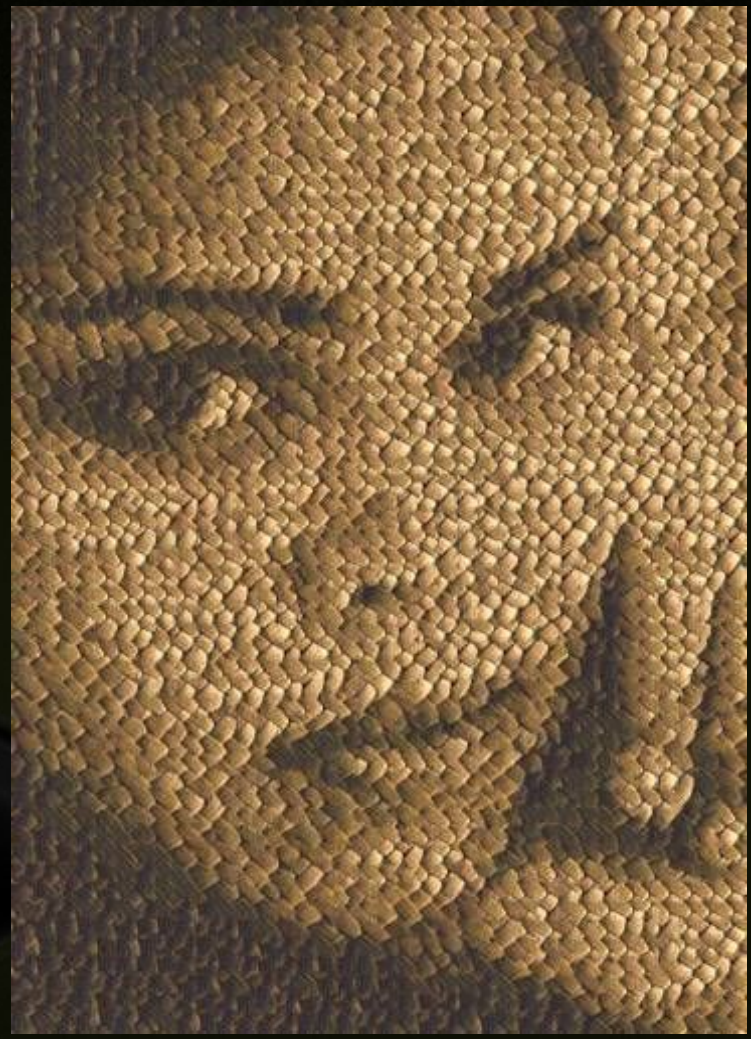
parmesan



rice




NVIDIA
NVIDIA Research



DP to find seams for panoramas

- **Error surface**

$$e = (I_1^o - I_2^o + S_c^o)^2$$

Overlapping area in the current composite image

Overlapping area in the current source image

- **Cumulative minimum error surface**

$$E(w,h) = e(w,h) + \min(E(w-1,h-1), E(w-1,h), E(w+1,h-1))$$

Yingen Xiong, Kari Pulli
Fast image labelling for producing high resolution panoramic images and its applications on mobile devices

Seam finding gets difficult when colors differ

- **Assume that the same surface has the same color**
  - **may not hold with independent images**
    - **lighting, exposure, white-balance, …**




No color correction

With color correction

Y. Xiong, K. Pulli, “Fast Panorama Stitching on Mobile Devices”, ICCE 2010

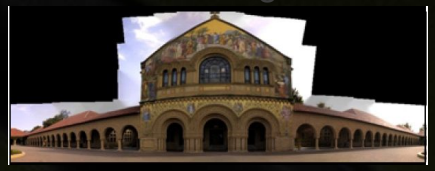
Color Matching

Related Work

Yingen Xiong, Kari Pulli
Color Matching of Image Sequences with Combined Gamma and Linear Corr
ACM Multimedia, 2010

- **Linear-model-based color correction**
  - **Luminance correction**

Meunier and Borgmann 2000



Linearized RGB color space

Brown and Lowe 2007

sRGB color space

  - **Color correction: linear match of averages**

Tian et al. 2002

sRGB color space

Ha et al. 2007

YCbCr color space

Xiong and Pulli 2009

Linearized RGB color space

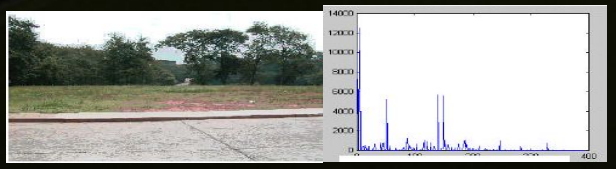
- **Polynomial m rs**

Pham and Pringle 1995

$$r = c_{000} + c_{001}B + c_{010}G + c_{100}R + c_{002}B^2 + c_{020}G^2 + c_{200}R^2 + c_{011}BG + c_{101}BR + c_{110}GR + \cdots + e \quad (1)$$

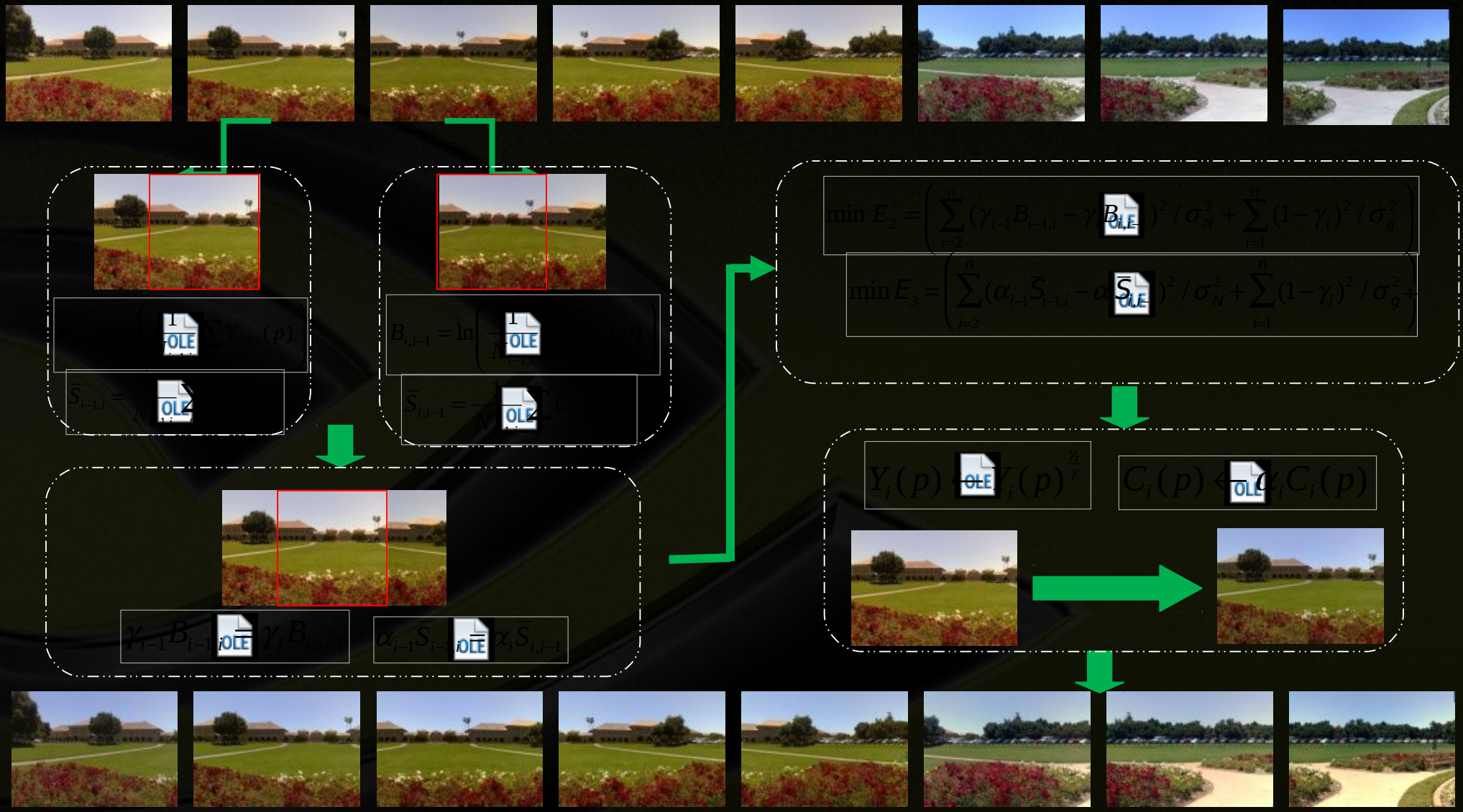
Polynomial mapping

Zhang et al. 2001



Histogram mapping

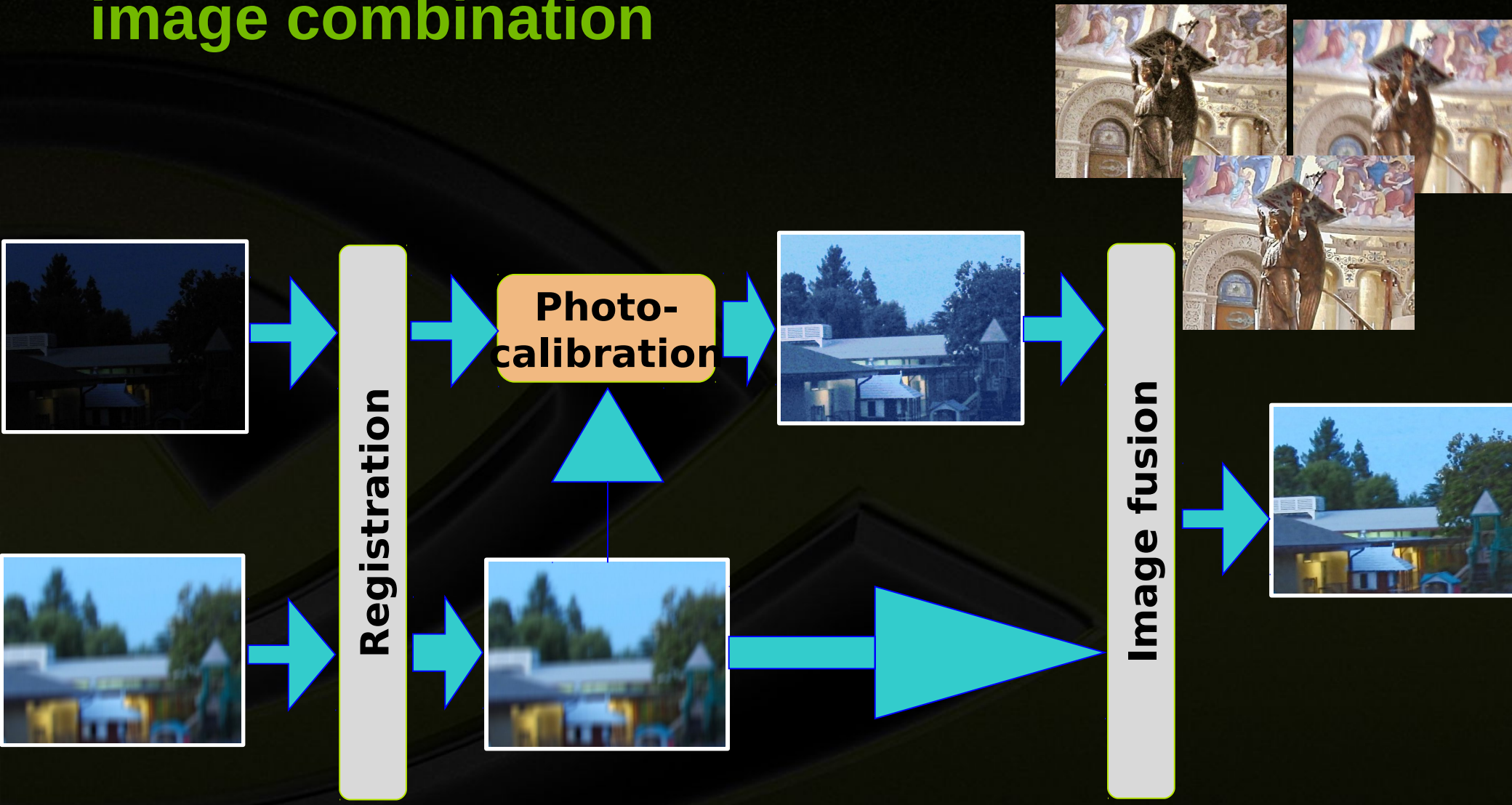
Combine Gamma and Linear Correction



Separate luminance & colors (Y C_b C_r)

- Luminance
  - gamma-corrected value Y
  - calculate an additional gamma so average luminances match
- Chrominance
  - linear chrominance C_b C_r
  - map linearly so that the averages match
- Last stage in prev. page (E_2, E_3)
  - minimize changes in the image sequence
- Advantages in using averages
  - fast; reduce the problem from long vectors to matching scalars
  - no need for per-pixel accurate registration

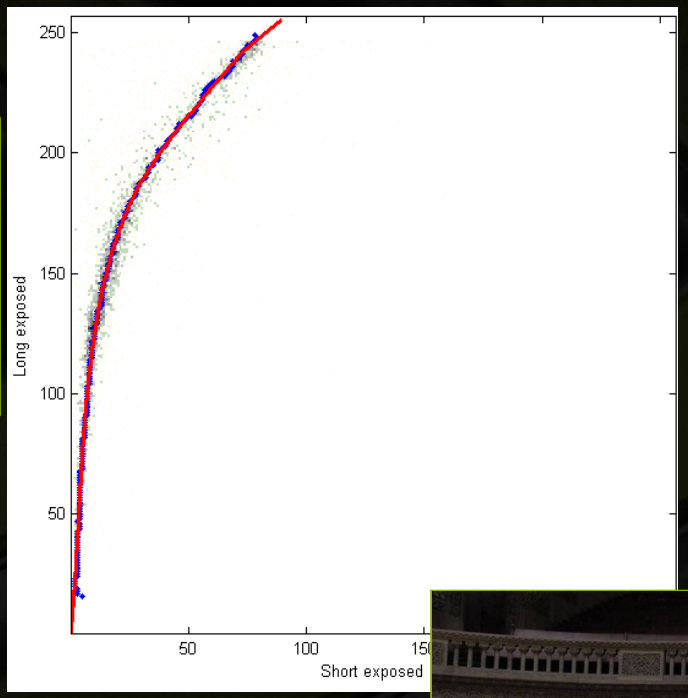
Matching colors for blurry / noisy image combination

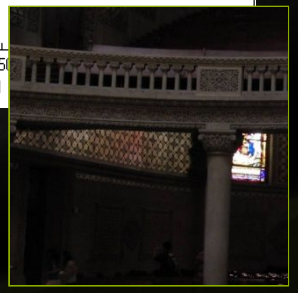


Marius Tico, Kari Pulli
Image Enhancement Method via Blur and Noisy Image Fusion
IEEE International Conference on Image Processing (ICIP'09)

Photometric calibration

- **Calibrate the brightness of the short-exposed image with respect to other images**





- Weighted joint histogram

$$\mathbf{C}(i,j) = \sum_{\mathbf{x}} w(i)w(j)\delta[I_1(\mathbf{x}) - i]\delta[I_2(\mathbf{x}) - j]$$

- Most likely correspondence

$$(i_0, j_0) = \arg\max_{(i,j)} \mathbf{C}(i,j)$$

- Estimate a transfer function

$$j = f(i), \quad f = \arg\max_{f} \left\{ \sum_i \mathbf{C}(i, f(i)) \right\}, \; s.t. \; f' \geq 0$$

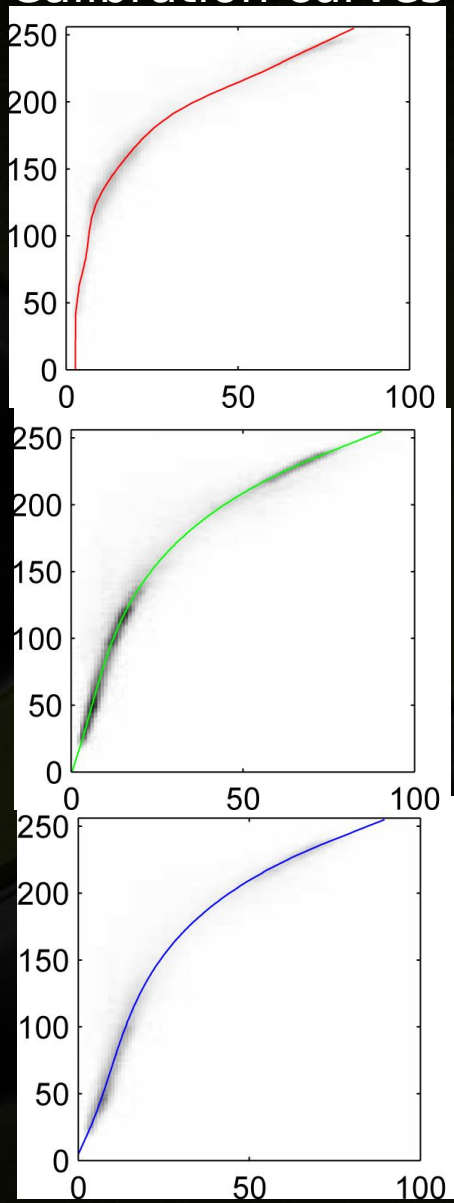
by Dynamic Programming starting from

- Brightness Transfer Function (BTF) obtained by polynomial fitting

Photometric calibration

Short exposure

Calibration curves



Calibrated short exposure

Long exposure

Long exposure

On-device editing

TouchTone for interactive tone mapping

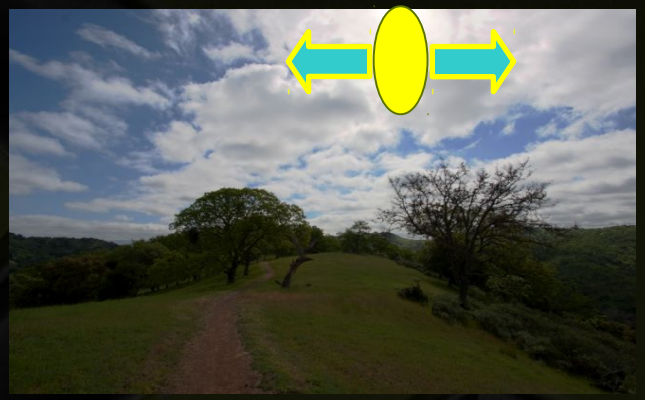
Point-and-Swipe User Interface

- A specified point generated a new selection mask

Chia-Kai Liang, Wei-Chao Chen, Natasha Gelfand
TouchTone: Interactive Local Image Adjustment Using Point and Swipe
Eurographics 2010

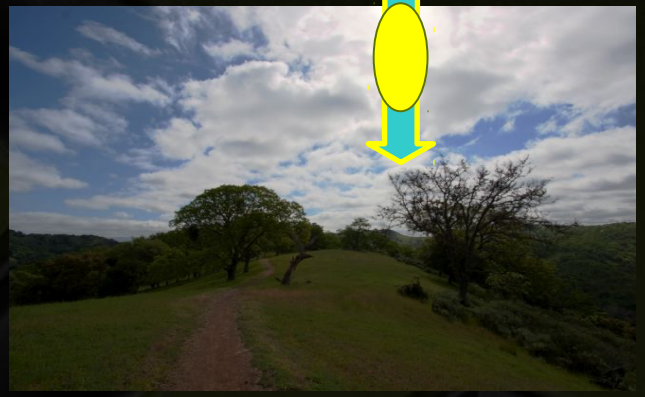
Point-and-Swipe User Interface

- Swipe left (right) decreases (increases) the editing values
  - each edit is independent
  - no complex multi-layer framework



Point-and-Swipe User Interface

- Swipe down (up) decreases (increases) the size of selected region
  - simply adjust the constraint values at the end points

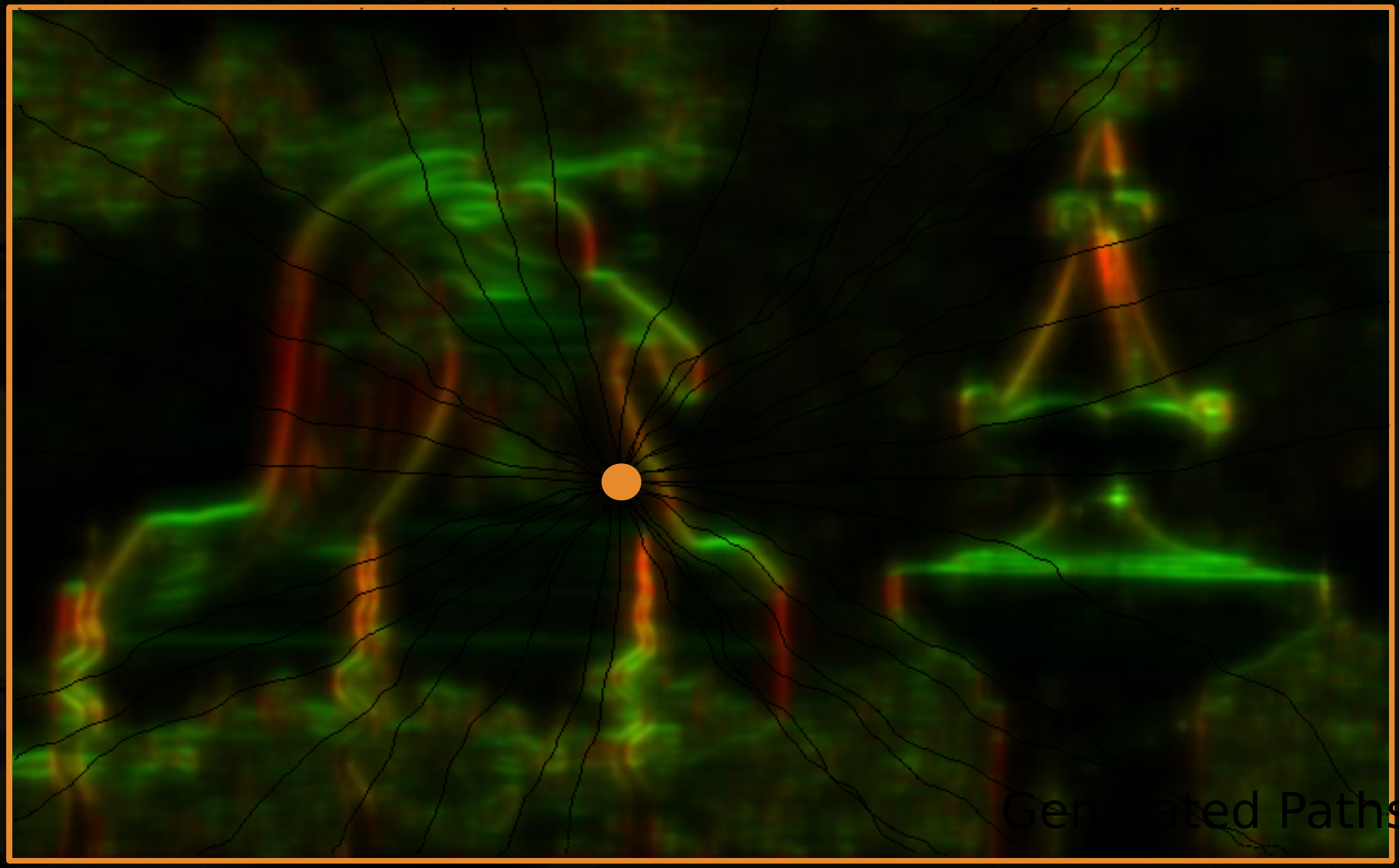


Calculating the weights

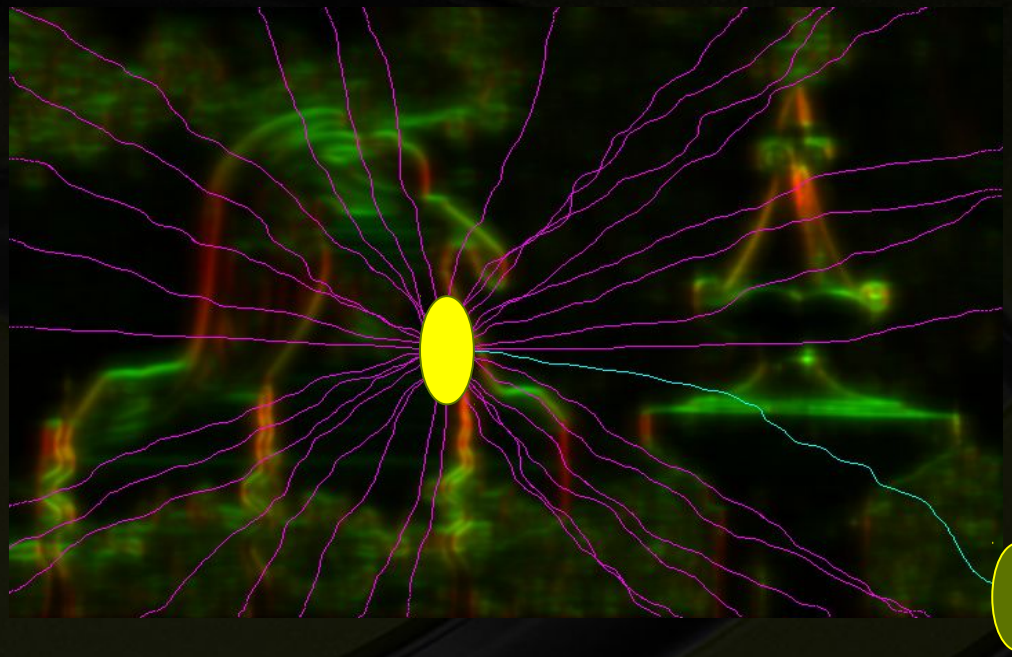
Preprocess: Gradients

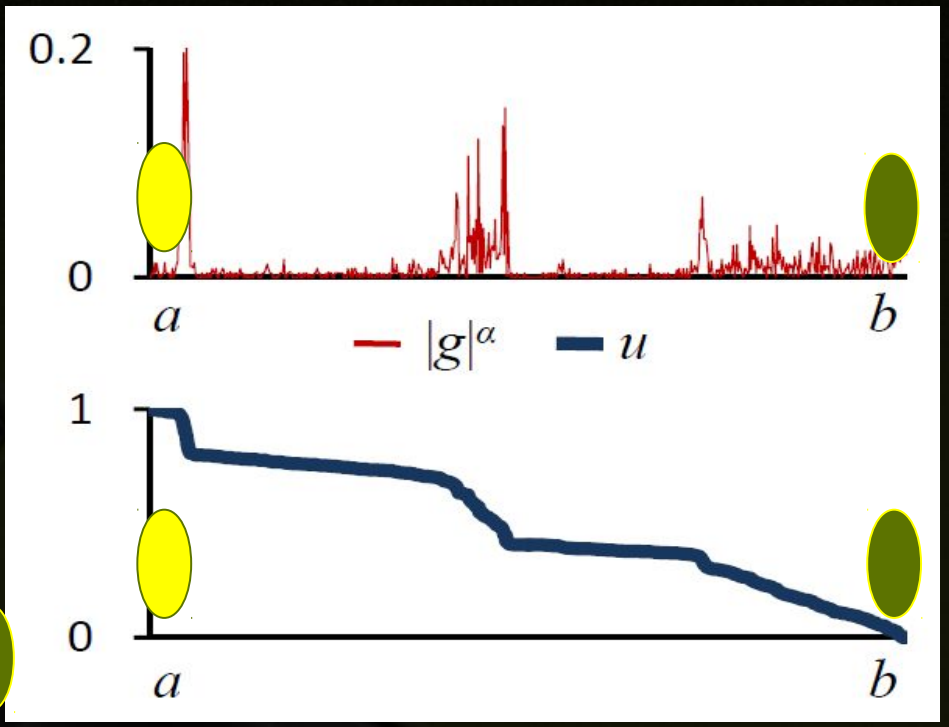
Preprocess: Gradients

Path Generation

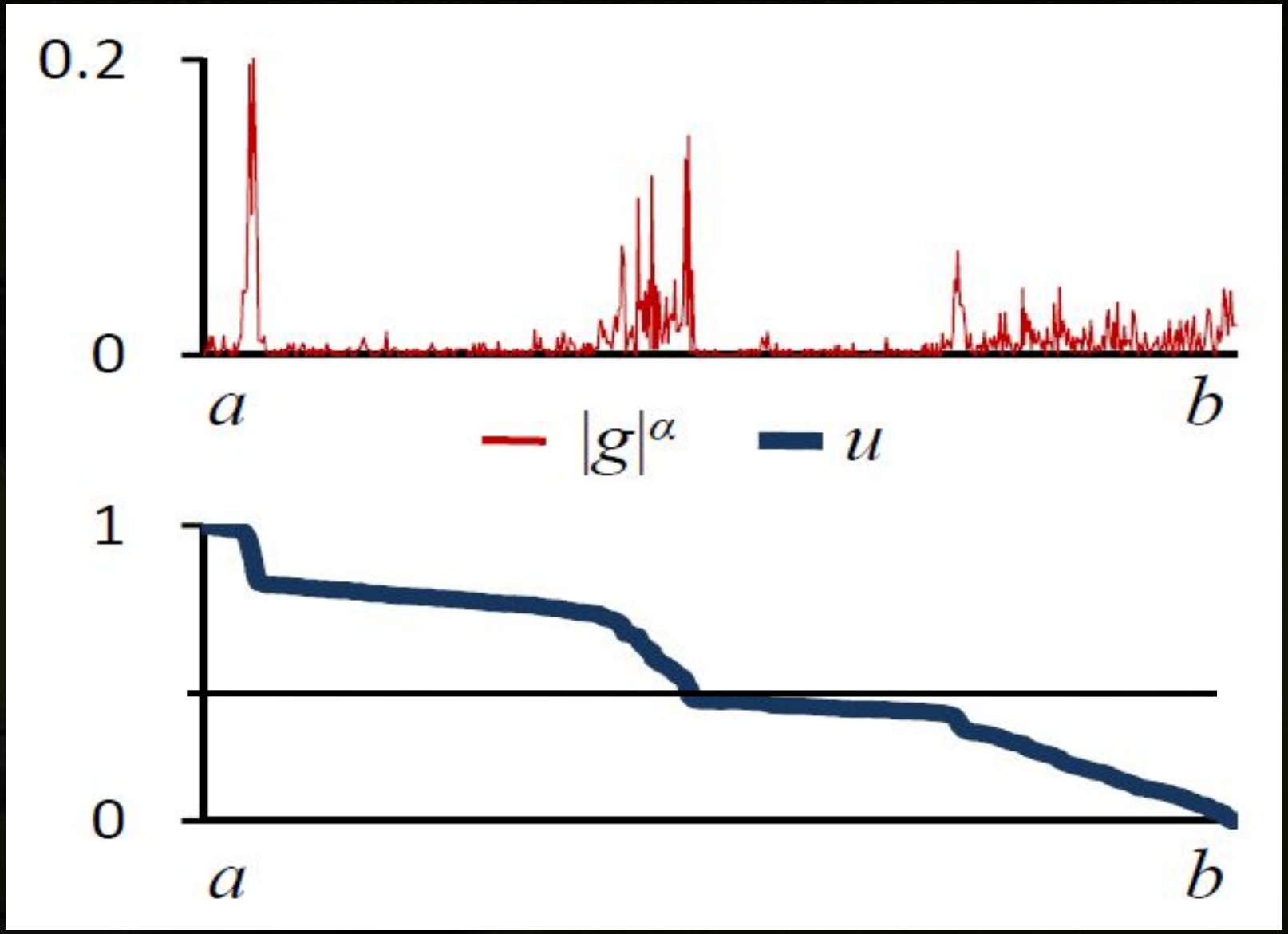


Reduce weights at strong gradients

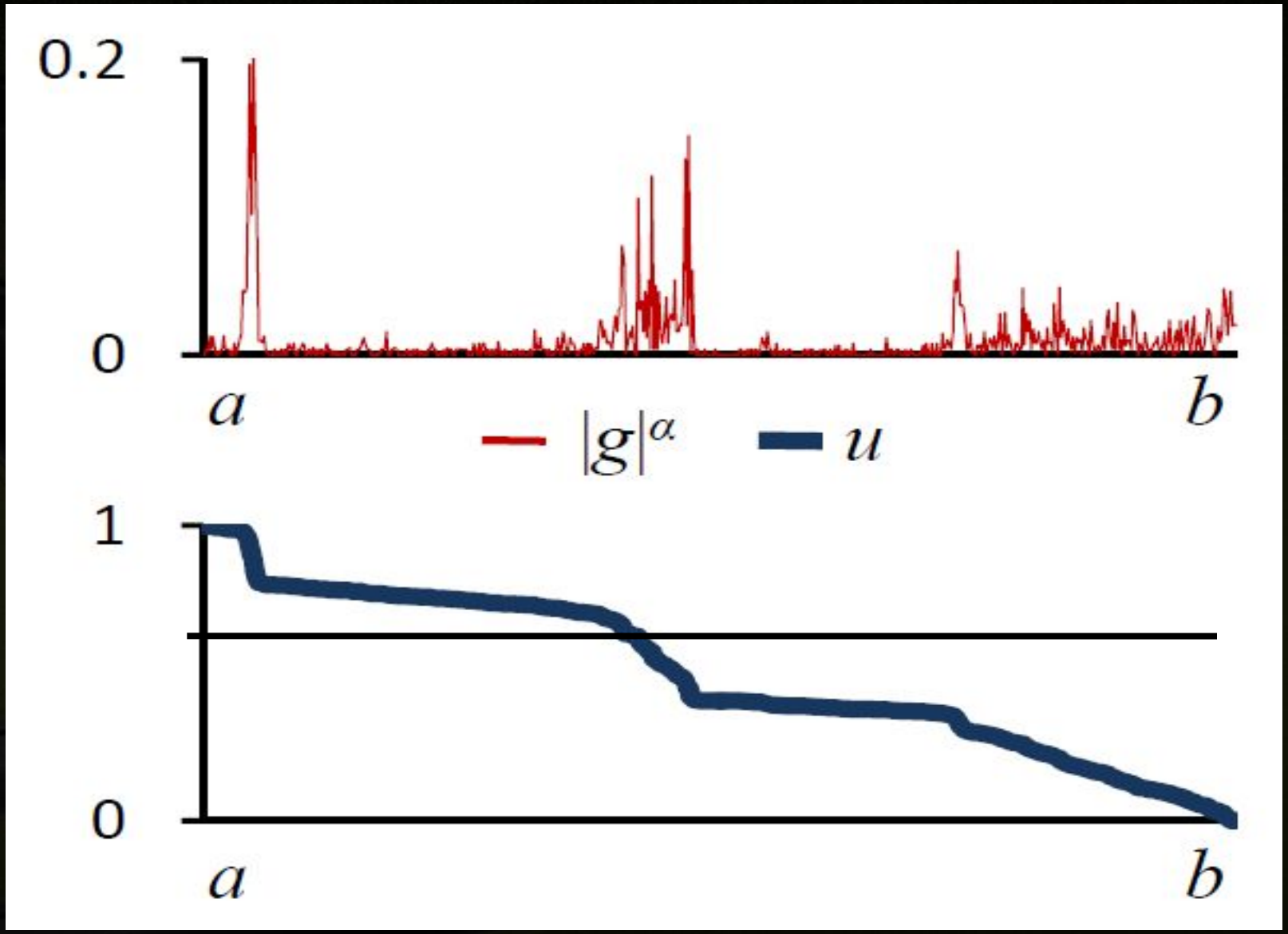




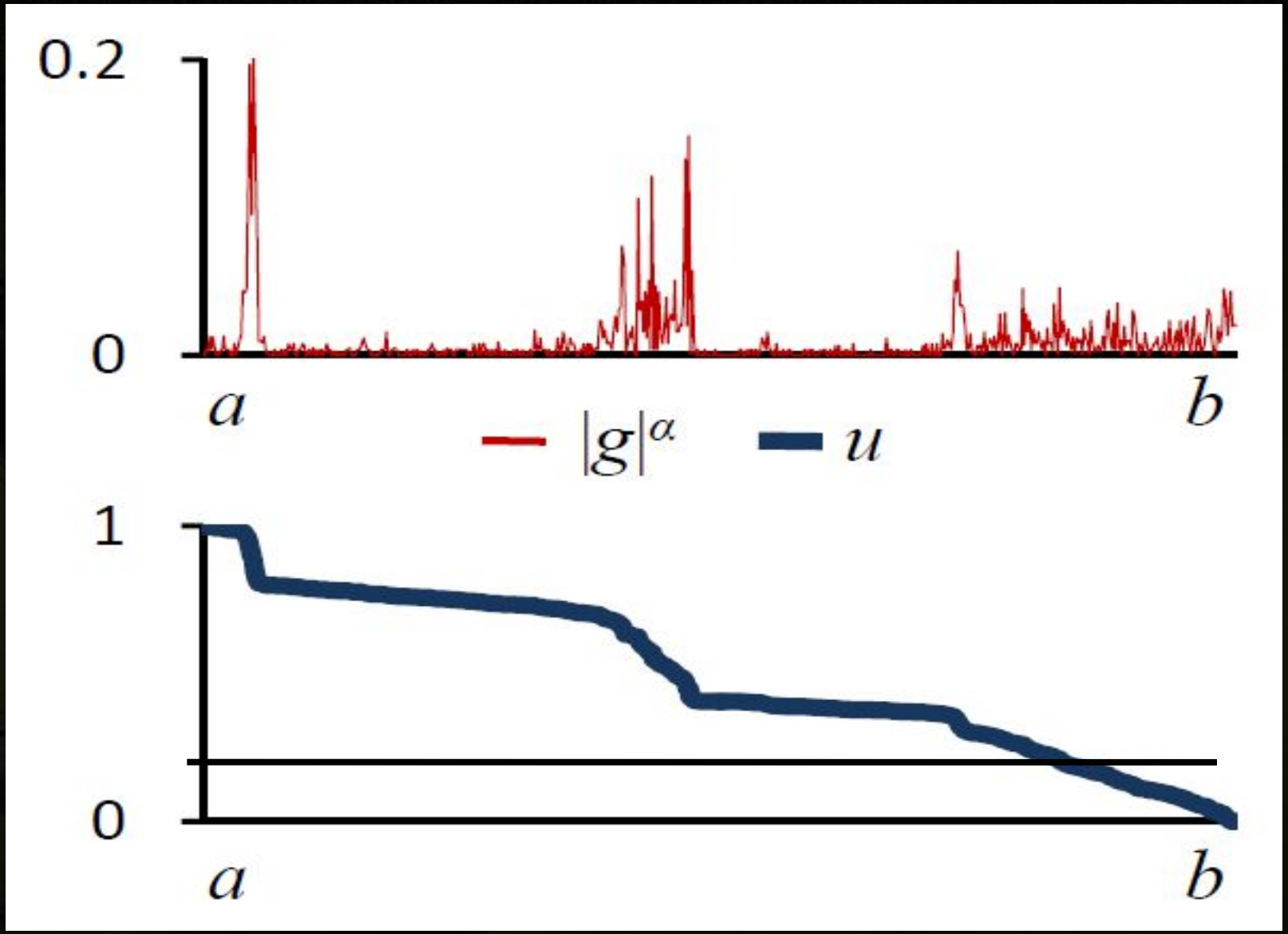
Default extent



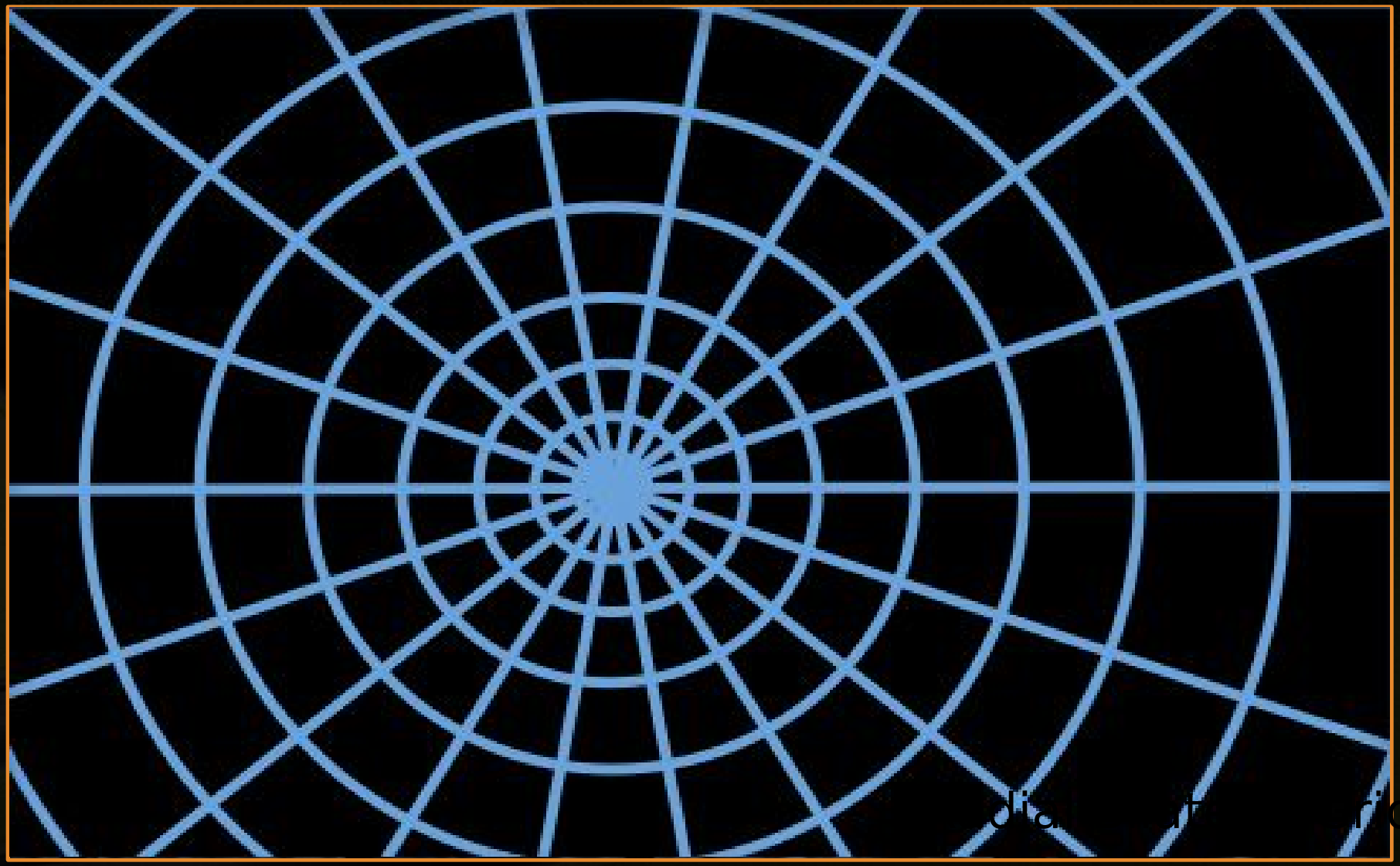
Reduce



Expand

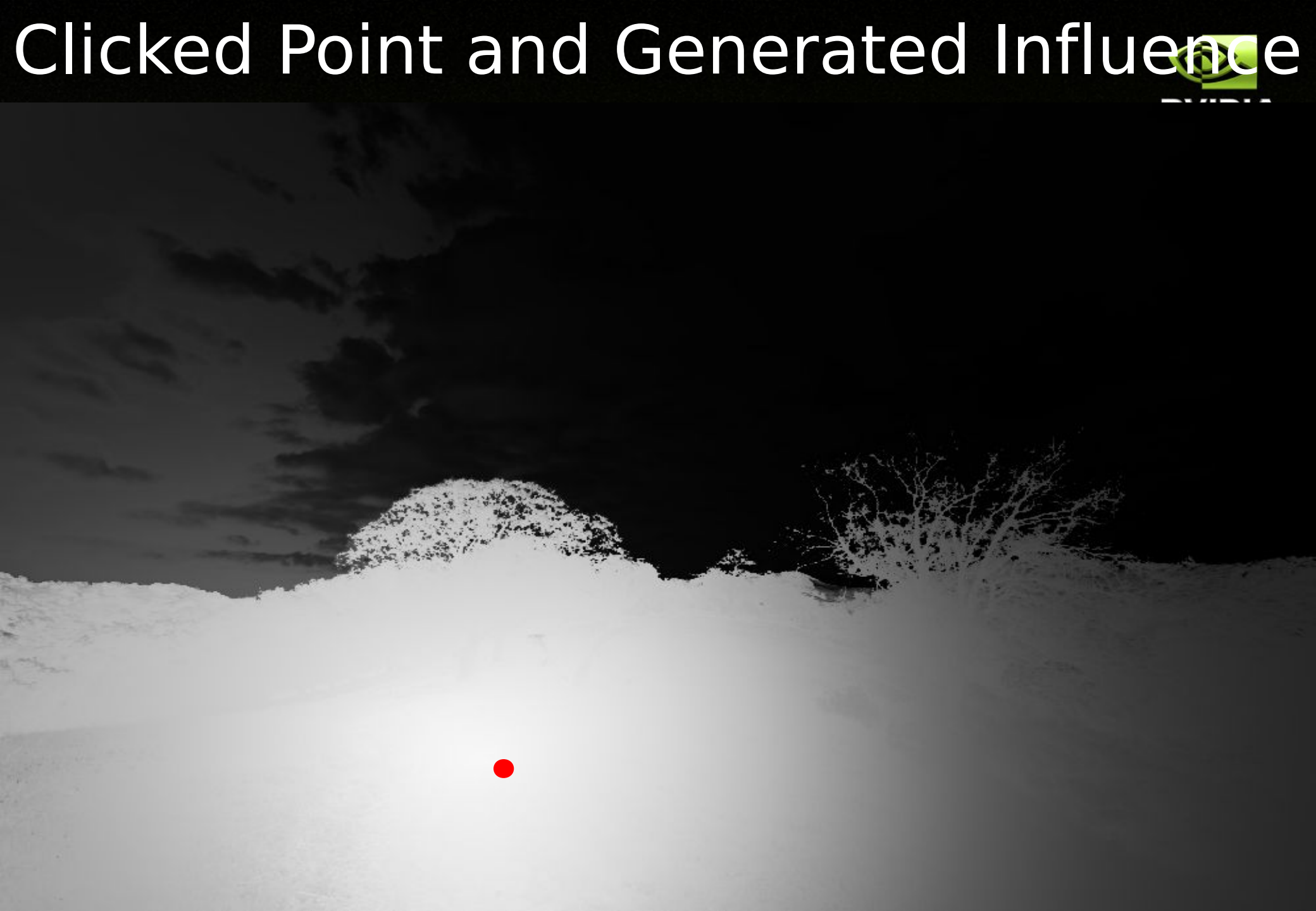


Scattered Bilateral Interpolation



Scattered Bilateral Interpolation

Tone-Mapped HDR image


Clicked Point and Generated Influence

After Brightness Adjustment

Original

After Brightness Adjustment

Clicked Point and Generated Influence

After Contrast Adjustment

Before

After Contrast Adjustment


Clicked Point and Generated Influence

After Color Adjustment

Before

After Color Adjustment

Comparison