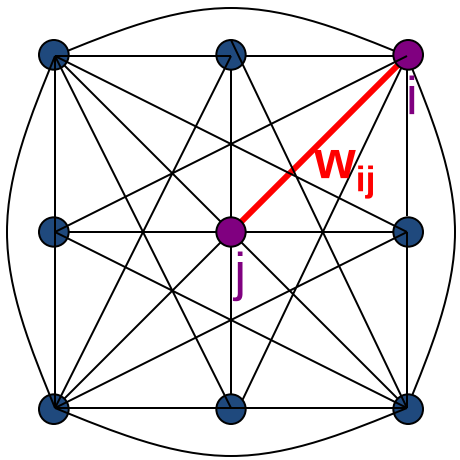
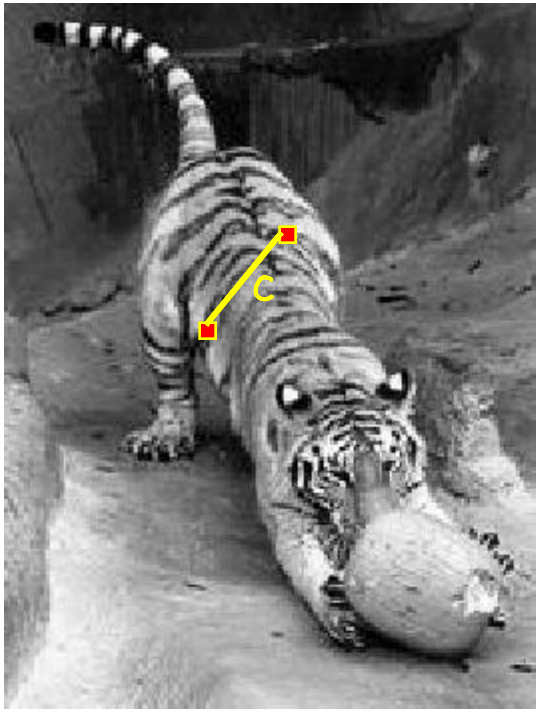
Images as graphs

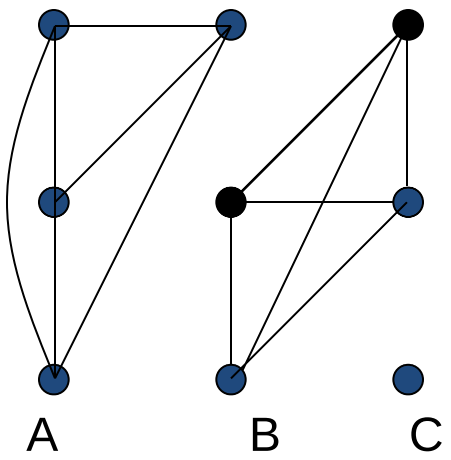




- *Fully-connected* graph
  - node for every pixel
  - link between *every* pair of pixels, **p**,**q**
  - similarity $\mathbf{w_{ij}}$ for each link

Source: Seitz

Segmentation by Graph Cuts



- Break Graph into Segments
  - Delete links that cross between segments
  - Easiest to break links that have low cost (low similarity)
    - similar pixels should be in the same segments
    - dissimilar pixels should be in different segments

Source: Seitz

Measuring Affinity

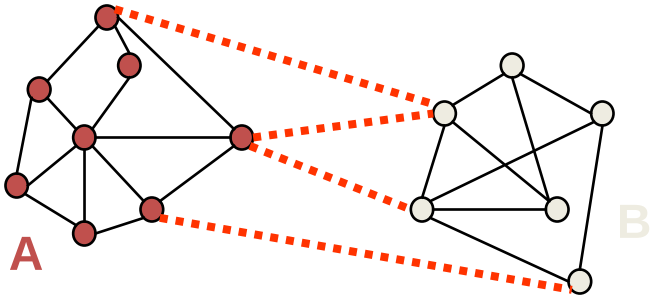
- Distance $$aff(x,y) = \exp\left\{-\frac{1}{2\sigma_d^2}\left\|x-y\right\|^2\right\}$$

- Intensity $$aff(x,y) = \exp\left\{-\frac{1}{2\sigma_d^2}\left\|I(x)-I(y)\right\|^2\right\}$$

- Color $$aff(x,y) = \exp\left\{-\frac{1}{2\sigma_d^2}\underbrace{dist\left(c(x),c(y)\right)}_{\text{(some suitable color space distance)}}{}^2\right\}$$

- Texture $$aff(x,y) = \exp\left\{-\frac{1}{2\sigma_d^2}\underbrace{\left\|f(x)-f(y)\right\|}_{\text{(vectors of filter outputs)}}{}^2\right\}$$

Cuts in a graph



- Link Cut
  - set of links whose removal makes a graph disconnected
  - cost of a cut:

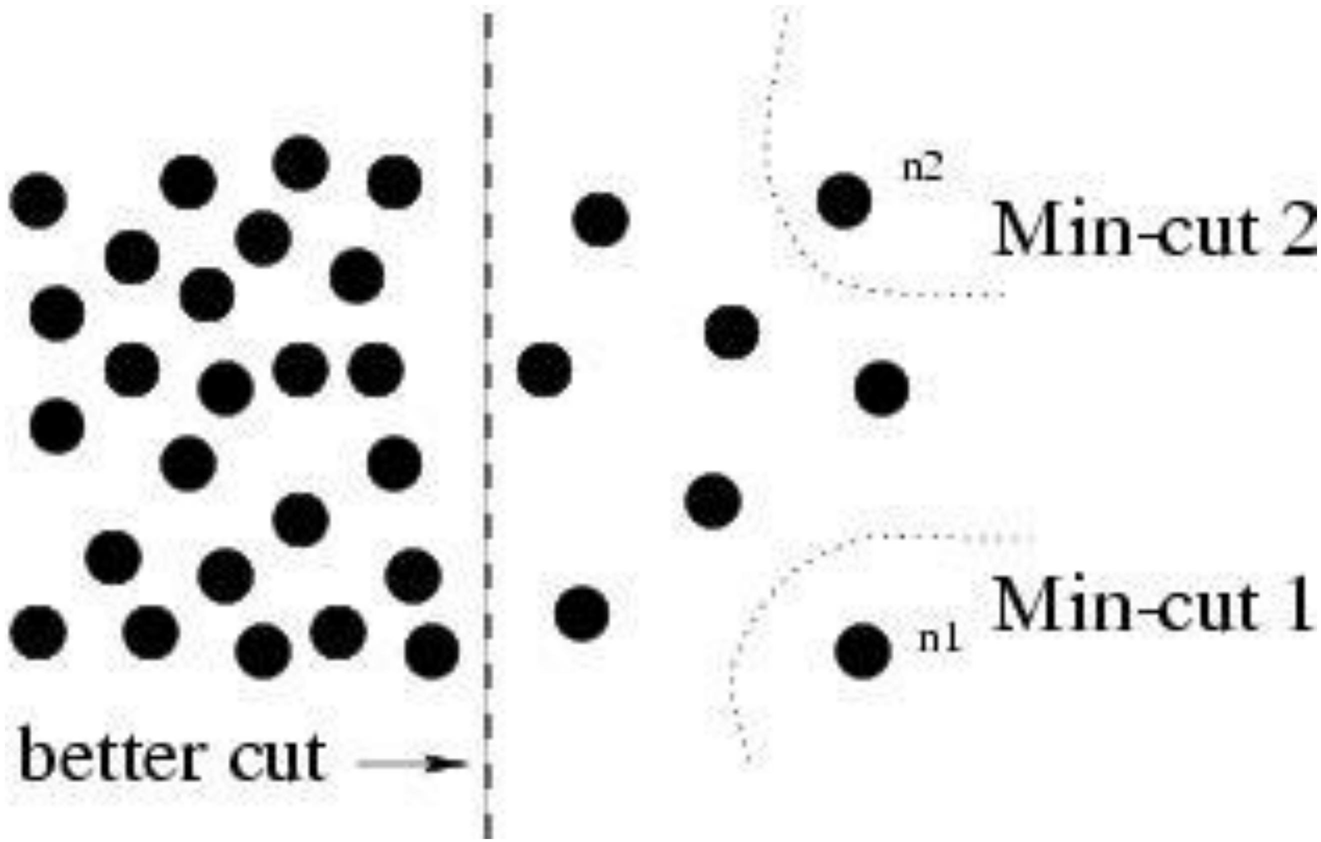
$$cut(A,B) = \sum_{p \in A, q \in B} c_{p,q}$$

One idea: Find minimum cut

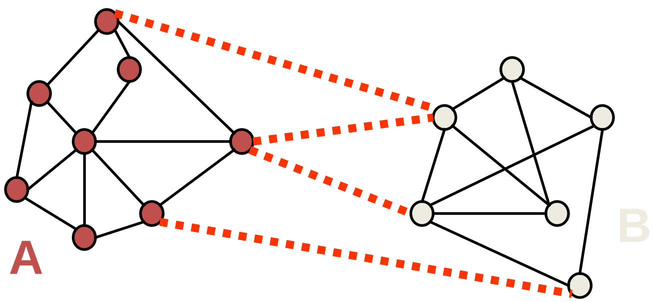
- gives you a segmentation
- fast algorithms exist for doing this

Source: Seitz

But min cut is not always the best cut...



Cuts in a graph



Normalized Cut

- a cut penalizes large segments
- fix by normalizing for size of segments

$$Ncut(A,B) = \frac{cut(A,B)}{volume(A)} + \frac{cut(A,B)}{volume(B)}$$

- volume(A) = sum of costs of all edges that touch A

Source: Seitz

Finding Minimum Normalized-Cut

- Finding the Minimum Normalized-Cut is NP-Hard.
- Polynomial Approximations are generally used for segmentation

Finding Minimum Normalized-Cut

$W = N \times N$ symmetric matrix, where

$$W(i,j) = \begin{cases} e^{-\|F_i - F_j\|/\sigma_F^2} \times e^{-\|X_i - X_j\|/\sigma_X^2} & \text{if } j \in N(i) \\ 0 & \text{otherwise} \end{cases}$$

$\|F_i - F_j\|$ = Image feature similarity

$\|X_i - X_j\|$ = Spatial Proximity

$D = N \times N$ diagonal matrix, where $D(i,i) = \sum_j W(i,j)$

* From Khurram Hassan-Shafique CAP5415 Computer Vision 2003

Finding Minimum Normalized-Cut

- It can be shown that

$$\min N_{cut} = \min_{\mathbf{y}} \frac{\mathbf{y}^{\mathbf{T}}(\mathbf{D}-\mathbf{W})\mathbf{y}}{\mathbf{y}^{\mathbf{T}}\mathbf{D}\mathbf{y}}$$

such that

$$y(i) \in \{1, -b\},\ 0 < b \le 1, \text{ and } \mathbf{y}^{T}\mathbf{D}\mathbf{1} = 0$$

- If *y* is allowed to take real values then the minimization can be done by solving the generalized eigenvalue system

$$(\mathbf{D}-\mathbf{W})\mathbf{y} = \lambda\mathbf{D}\mathbf{y}$$

* Slide from Khurram Hassan-Shafique CAP5415 Computer Vision 2003

Algorithm

- Compute matrices W & D
- Solve $(\mathbf{D}-\mathbf{W})\mathbf{y}=\lambda\mathbf{D}\mathbf{y}$ for eigen vectors with the smallest eigen values
- Use the eigen vector with second smallest eigen value to bipartition the graph
- Recursively partition the segmented parts if necessary.

* Slide from Khurram Hassan-Shafique CAP5415 Computer Vision 2003

Recursive normalized cuts

1. Given an image or image sequence, set up a weighted graph: G=(V, E)
   - Vertex for each pixel
   - Edge weight for nearby pairs of pixels

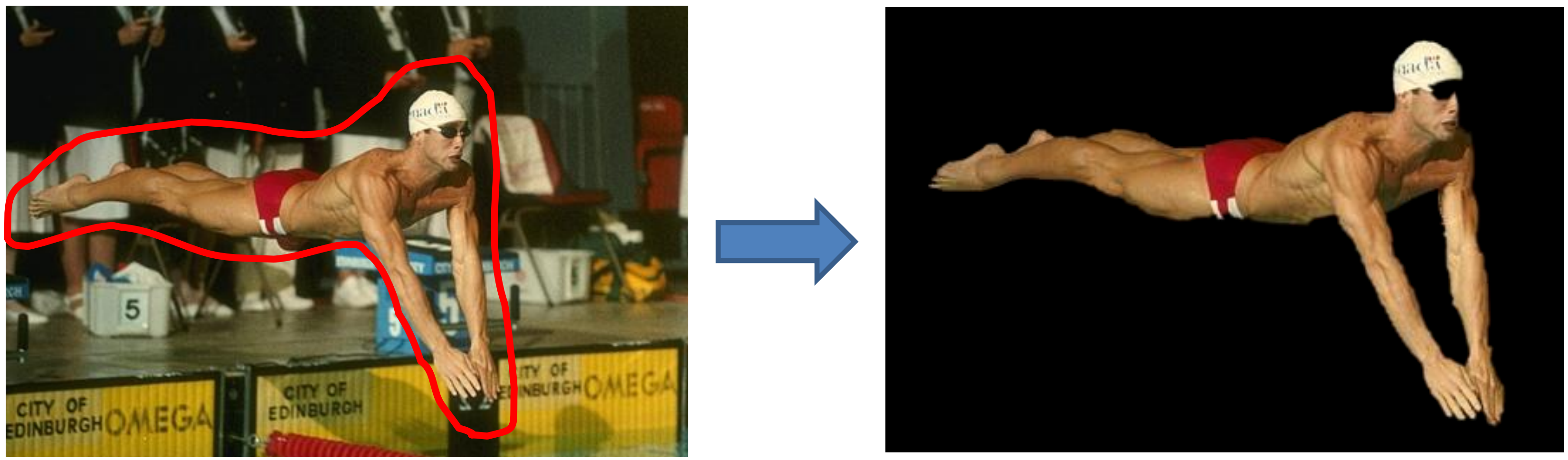
2. Solve for eigenvectors with the smallest eigenvalues: $(D - W)y = \lambda Dy$
   - Use the eigenvector with the second smallest eigenvalue to bipartition the graph
   - Note: this is an approximation

4. Recursively repartition the segmented parts if necessary

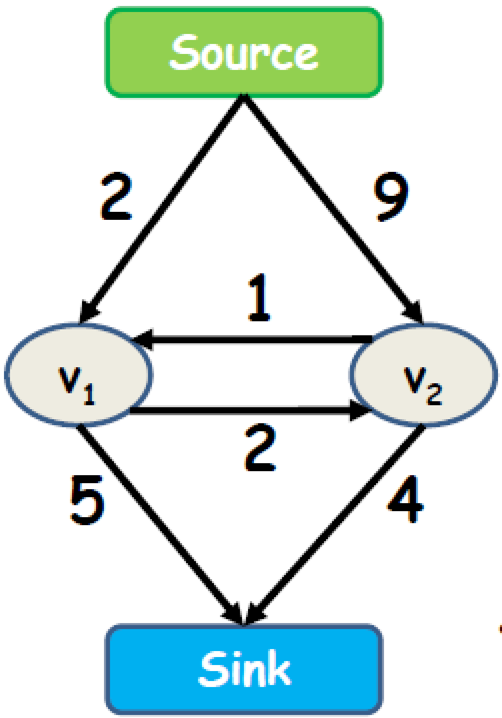
Details: *[http://www.cs.berkeley.edu/~malik/papers/SM-n**cut**.pdf](http://www.cs.berkeley.edu/~malik/papers/SM-ncut.pdf)*

Normalized cuts results

Graph cuts segmentation



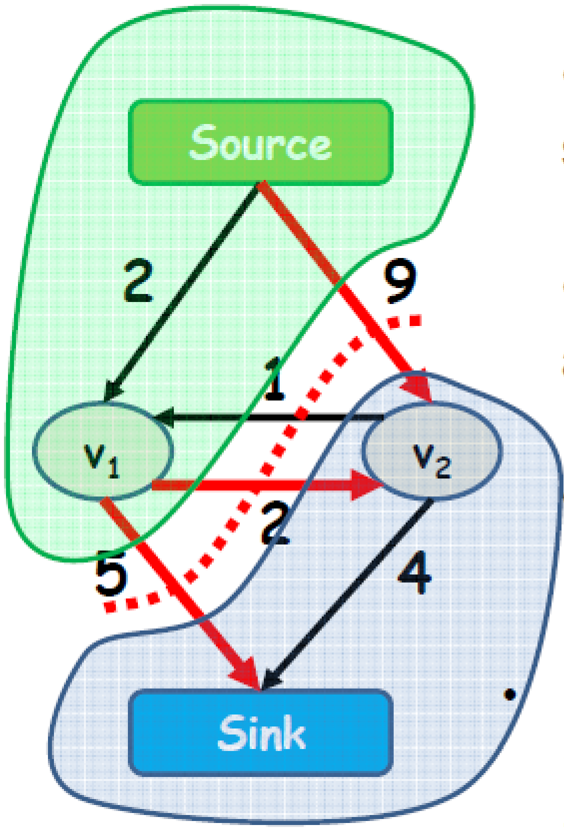
The st-Mincut Problem



- An st-cut (**S**,**T**) divides the nodes between source and sink
- The cost of the cut is the sum of costs of all edges going from S to T
- The st-min-cut is the cut with lowest cost

- Each node is either assigned to the source S or sink T
- The cost of the edge (i, j) is taken if ($i \in S$) and ($j \in T$)

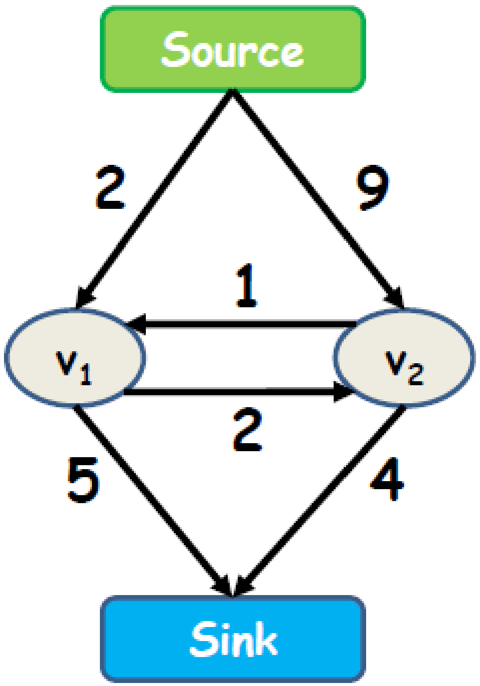
The st-Mincut Problem



5 + 2 + 9 = 16

- An st-cut (**S**,**T**) divides the nodes between source and sink
- The cost of the cut is the sum of costs of all edges going from S to T
- The st-min-cut is the cut with lowest cost
- Each node is either assigned to the source S or sink T
- The cost of the edge (i, j) is taken if ($i \in S$) and ($j \in T$)

Min-cut\Max-flow Theorem



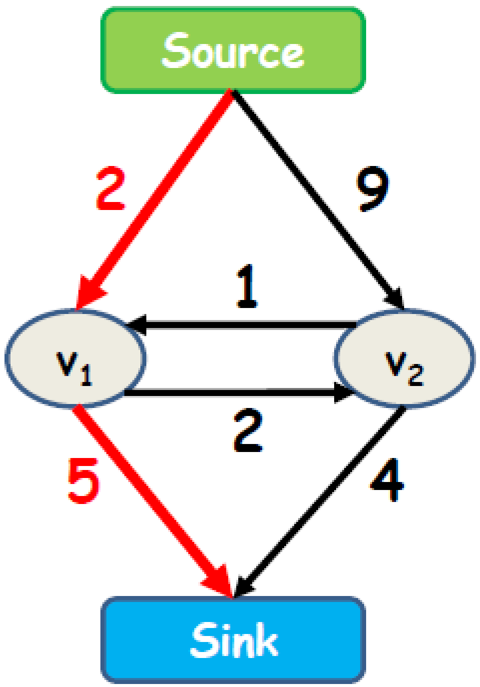
In every network, the maximum flow equals the cost of the st-mincut

Max flow = min cut = 7

Next: the augmented path algorithm for computing the max-flow/min-cut

Maxflow Algorithms

Flow = 0

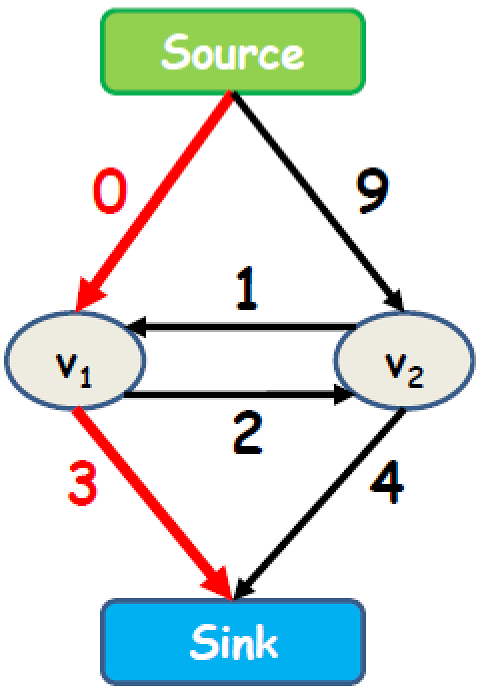


Augmenting Path Based Algorithms

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

Maxflow Algorithms

Flow = 2

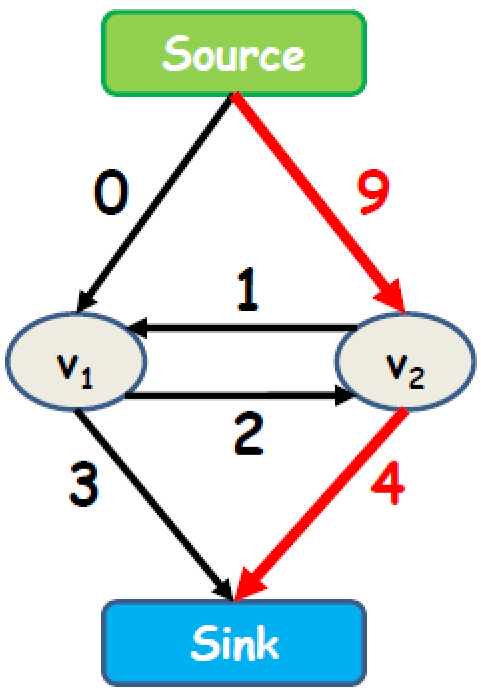


Augmenting Path Based Algorithms

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

Maxflow Algorithms

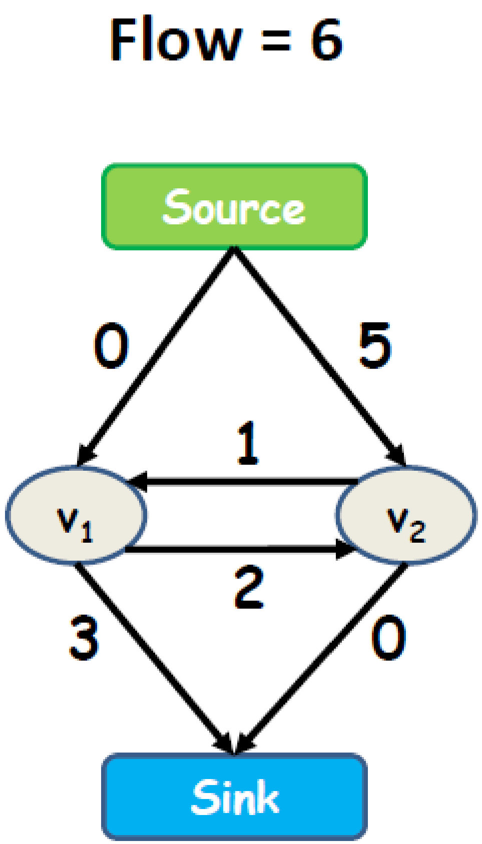
Flow = 2



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

Maxflow Algorithms

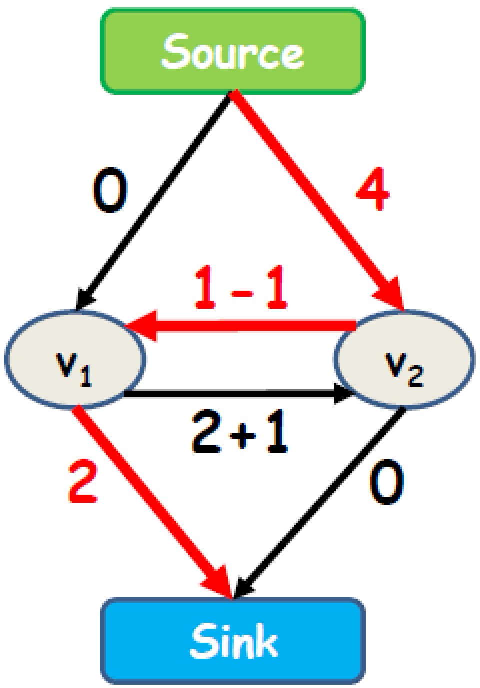


Augmenting Path Based Algorithms

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

Maxflow Algorithms

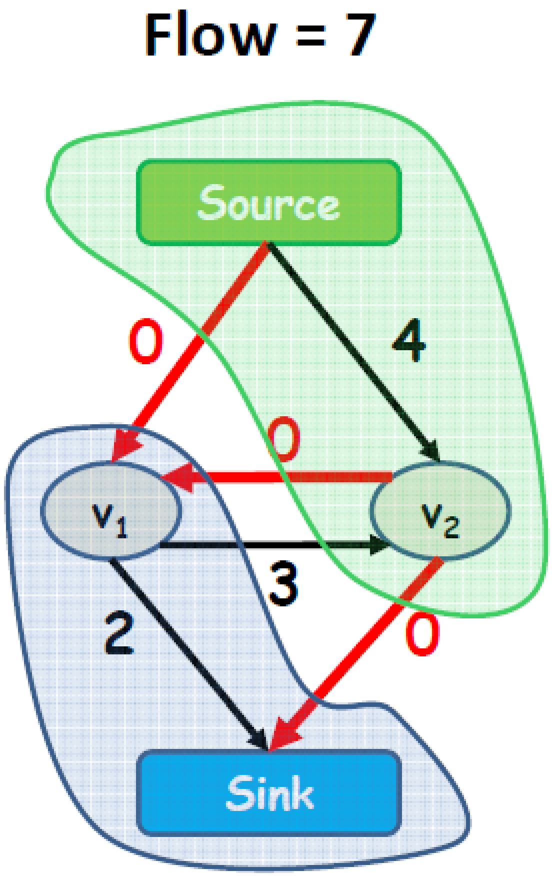
Flow = 6 + 1

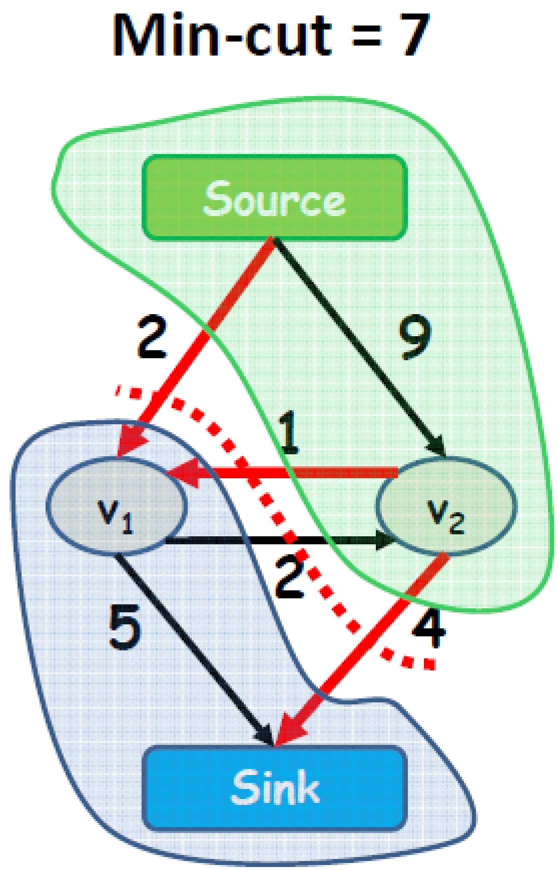


Augmenting Path Based Algorithms

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

Maxflow Algorithms



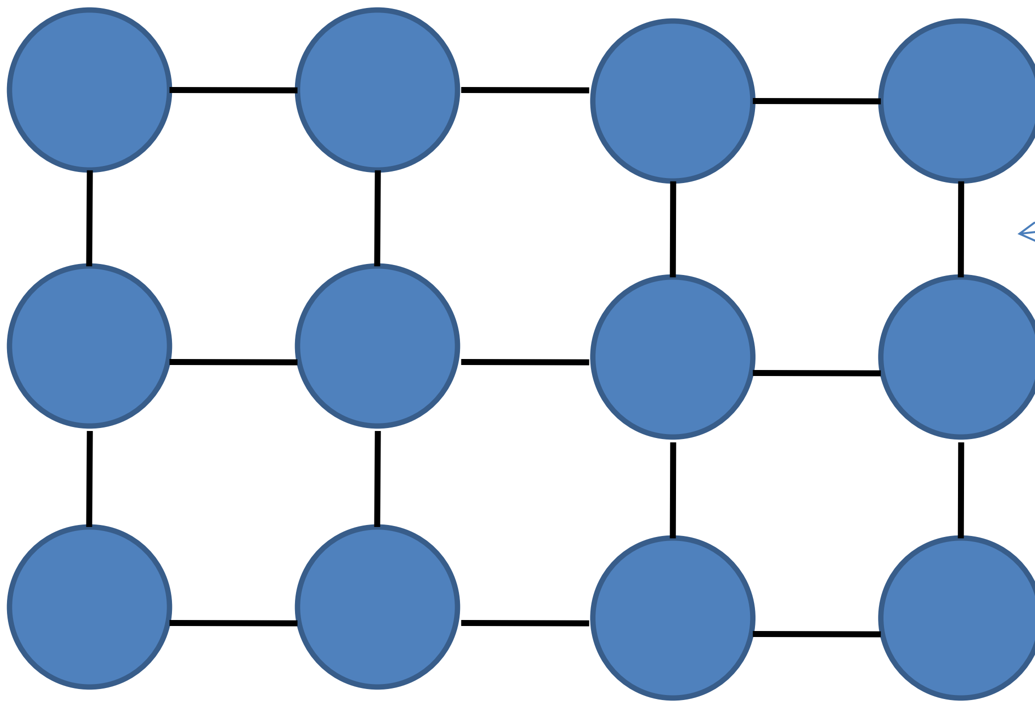


2 + 1 + 4 = 7

Markov Random Fields

Node y_i: pixel label

Edge: constrained pairs

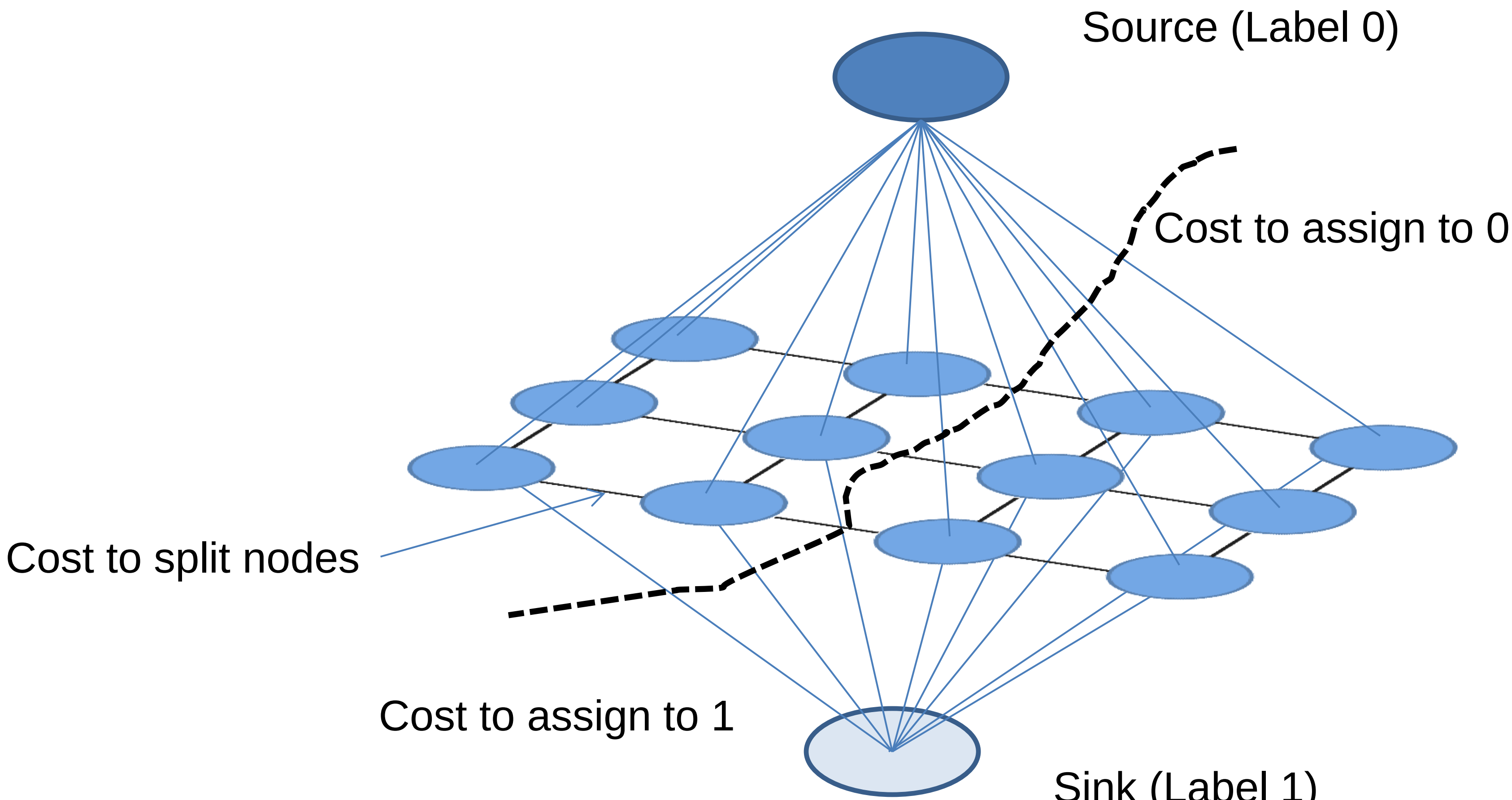


Cost to assign a label to each pixel

Cost to assign a pair of labels to connected pixels

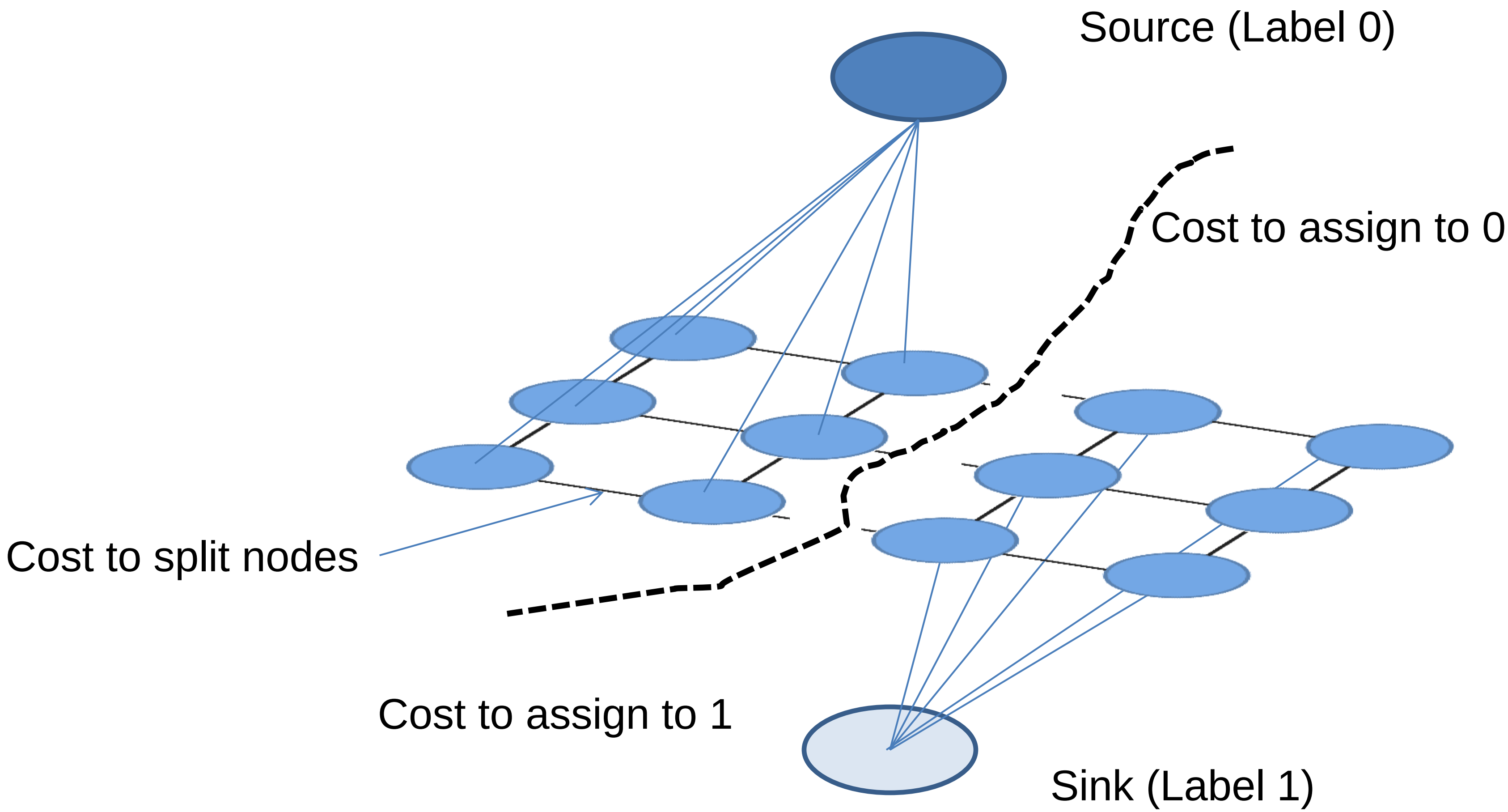
$$Energy(\mathbf{y};\theta,data)=\sum_i \psi_1(y_i;\theta,data)\ \sum_{i,j\in edges}\psi_2(y_i,y_j;\theta,data)$$

Solving MRFs with graph cuts



$$Energy(\mathbf{y};\theta,data)=\sum_i \psi_1(y_i;\theta,data)\ \sum_{i,j\in edges}\psi_2(y_i,y_j;\theta,data)$$

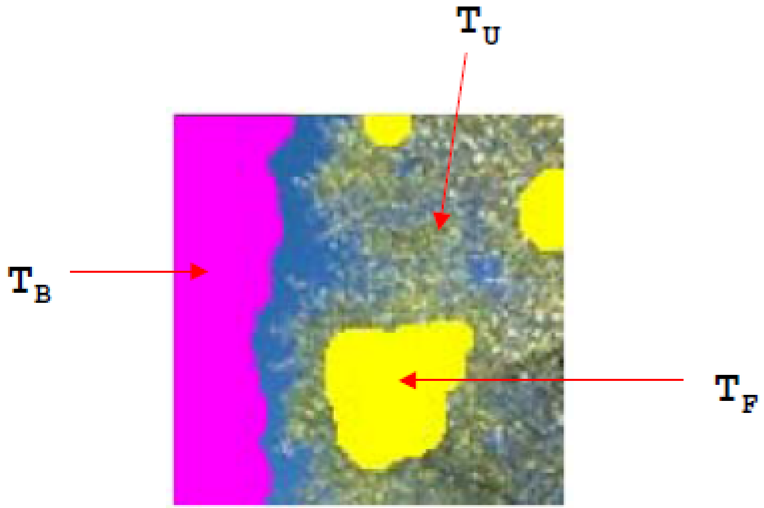
Solving MRFs with graph cuts

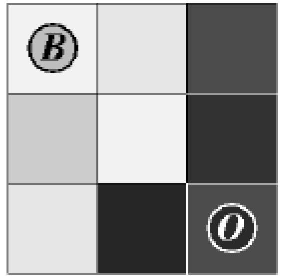


$$Energy(\mathbf{y};\theta,data) = \sum_i \psi_1(y_i;\theta,data) \sum_{i,j \in edges} \psi_2(y_i,y_j;\theta,data)$$

GraphCut for a Monochrome Image

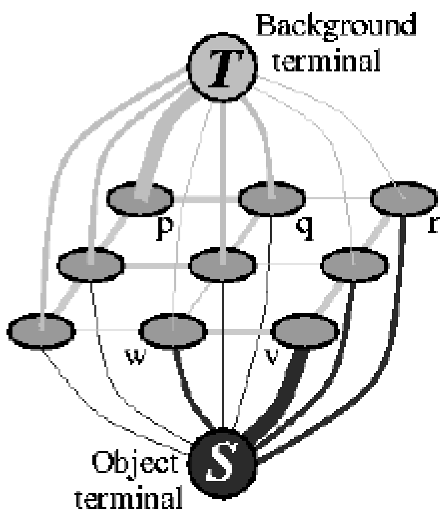
- User provides a trimap $T = \{ T_F, T_B, T_U \}$ which partitions the image into 3 regions: foreground, background, unknown.



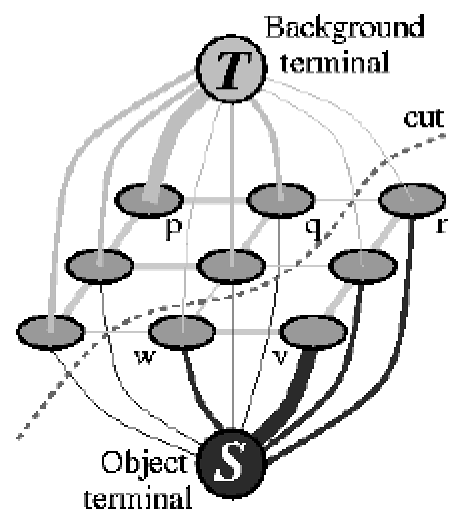


(a) Image with seeds.

(d) Segmentation results.



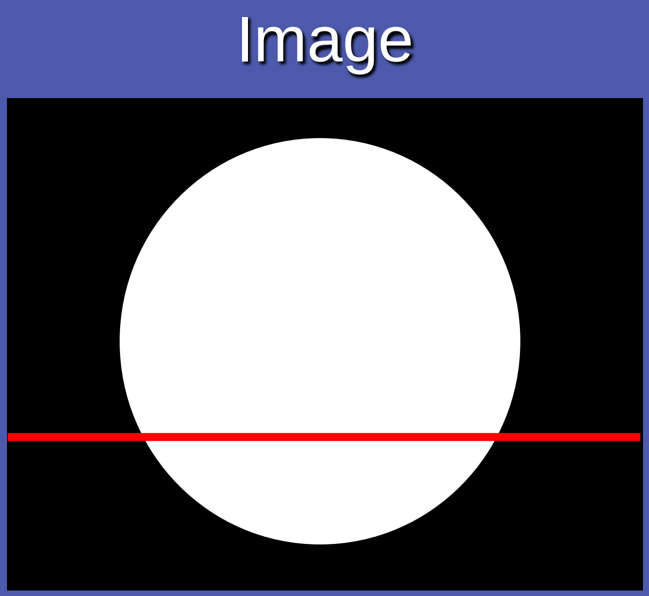
(b) Graph.

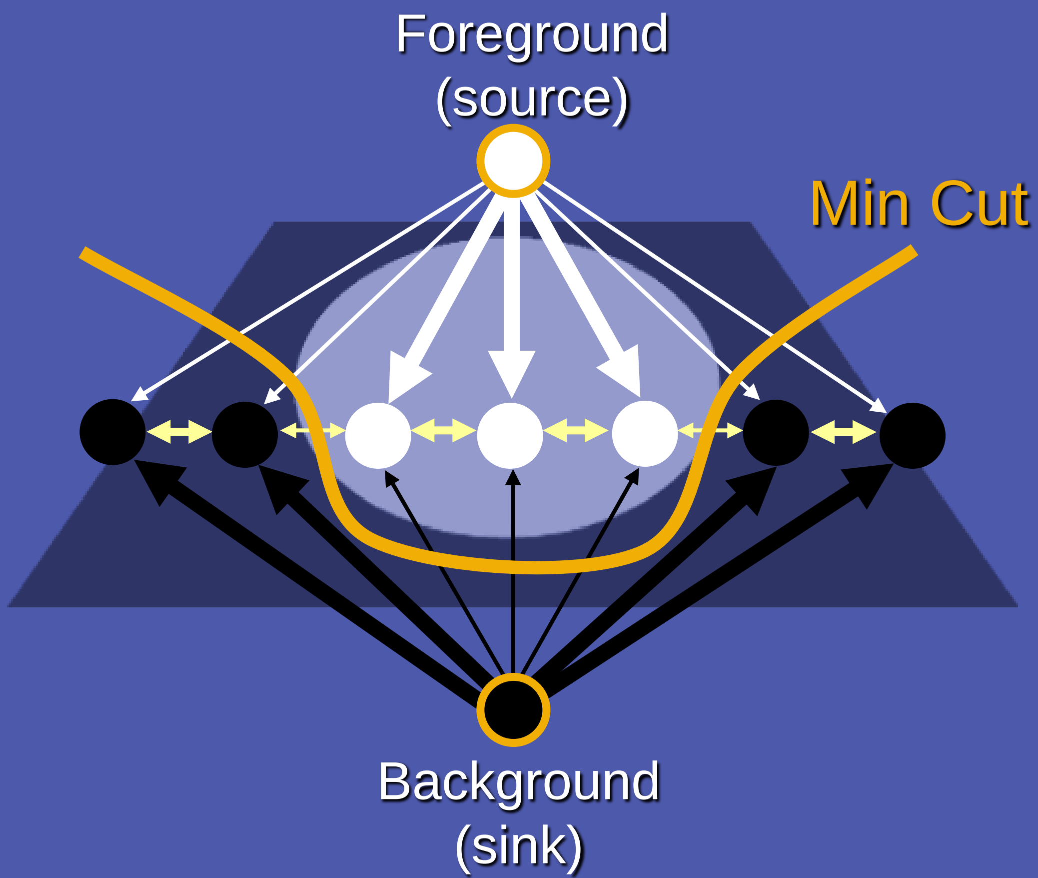


(c) Cut.

Graph cuts

Boykov and Jolly (2001)





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

Min Cut: Global minimal enegry in polynomial time

Source: Rother

Optimization Formulation - Boykov & Jolly '01

$$E(A) = \lambda \cdot R(A) + B(A)$$

where

$$R(A) = \sum_{p \in \mathcal{P}} R_p(A_p)$$

$$B(A) = \sum_{\{p,q\} \in \mathcal{N}} B_{\{p,q\}} \cdot \delta(A_p, A_q)$$

and

$$\delta(A_p, A_q) = \begin{cases} 1 & \text{if } A_p \neq A_q \\ 0 & \text{otherwise.} \end{cases}$$

- Goal: Find Segmentation, A, which minimizes E(A)

- A – Proposed Segmentation
- E(A) – Overall Energy
- R(A) – Degree to which pixels fits model
- B(A) – Degree to λ which the cuts breaks up similar pixels
- - Balance A() and B()

Link Weights

edge	weight (cost)	for
$\{p,q\}$	$B_{\{p,q\}}$	$\{p,q\} \in \mathcal{N}$
	$\lambda \cdot R_p(\text{“bkg”})$	$p \in \mathcal{P},\ p \notin \mathcal{O} \cup \mathcal{B}$
$\{p,S\}$	K	$p \in \mathcal{O}$
	0	$p \in \mathcal{B}$
	$\lambda \cdot R_p(\text{“obj”})$	$p \in \mathcal{P},\ p \notin \mathcal{O} \cup \mathcal{B}$
$\{p,T\}$	0	$p \in \mathcal{O}$
	K	$p \in \mathcal{B}$

$$K = 1 + \max_{p \in \mathcal{P}} \sum_{q:\ \{p,q\} \in \mathcal{N}} B_{\{p,q\}}$$

- Pixel links based on color/intensity similarities
- Source/Target links based on histogram models of fore/background

$$
\begin{aligned}
R_p(\text{“obj”}) &= -\ln \Pr(I_p|\mathcal{O}) \\
R_p(\text{“bkg”}) &= -\ln \Pr(I_p|\mathcal{B}).
\end{aligned}
$$

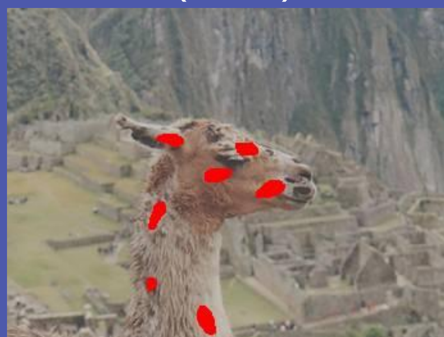
$$
B_{\{p,q\}} \propto exp\left(-\frac{(I_p - I_q)^2}{2\sigma^2}\right) \cdot \frac{1}{dist(p,q)}.
$$

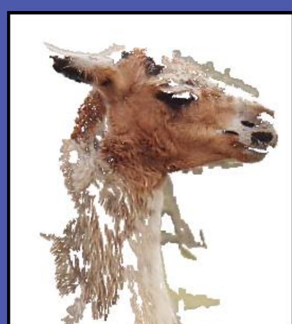
Grab cuts and graph cuts

User Input

Result

Magic Wand
(198?)



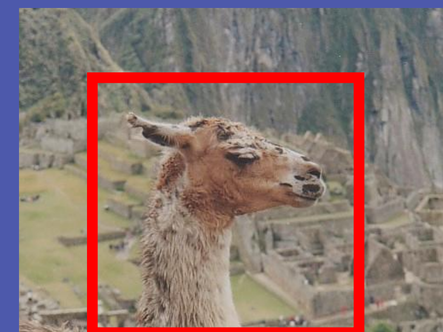


Regions

Intelligent Scissors
Mortensen and Barrett (1995)

Boundary

GrabCut

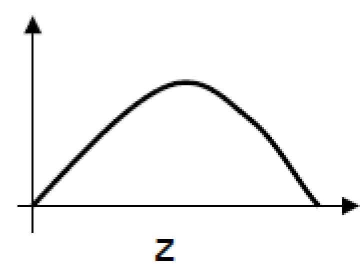


Regions & Boundary

Source: Rother

- The image is an array $\boldsymbol{z} = (z_1, \ldots z_N)$ of grey values indexed by the single index n.
- The segmentation of the image is an alpha-channel, or, a series of opacity values $\boldsymbol{\alpha}=(\alpha_1,\ldots, \alpha_N)$ at each pixel with $0 \le \alpha_n \le 1$.
- The parameter $\boldsymbol{\theta}$ describes the foreground/background grey-level distributions. i.e. a pair of histogram of gray values:

$$\theta = \{h(z;\alpha), \alpha = 0,1\}$$



Segmentation by Energy Minimization

- An energy function **E** is defined so that its minimum corresponds to a good segmentation.
- This is captured by a "Gibbs" energy of the form:

$$\mathrm{E}(\boldsymbol{\alpha},\boldsymbol{\theta},\mathbf{z}) = \mathbf{U}(\boldsymbol{\alpha},\boldsymbol{\theta},\mathbf{z}) + \mathbf{V}(\boldsymbol{\alpha},\mathbf{z})$$

$$\mathbf{E}(\boldsymbol{\alpha},\boldsymbol{\theta},\mathbf{z}) = \mathbf{U}(\boldsymbol{\alpha},\boldsymbol{\theta},\mathbf{z}) + \mathbf{V}(\boldsymbol{\alpha},\mathbf{z})$$

- U evaluates the fit of the opacity α to the data $\mathbf{z}$
  - i.e. it gives a good score (low score) if α looks like it's consistent with the histogram.

$$U(\alpha,\theta,z) = \sum_n -\log h(z_n;\alpha_n)$$

- V is a smoothness term which penalizes if there is too much disparity between neighboring pixel values.

$$V(\underline{\alpha},\mathbf{z}) = \gamma \sum_{(m,n)\in\mathbf{C}} dis(m,n)^{-1} \left[\alpha_n \neq \alpha_m\right] \exp -\beta (z_m - z_n)^2,$$

$$\beta = \left(2\left\langle (z_m - z_n)^2 \right\rangle\right)^{-1}$$

$$\mathrm{E}(\boldsymbol{\alpha},\boldsymbol{\theta},\mathbf{z}) = \mathbf{U}(\boldsymbol{\alpha},\boldsymbol{\theta},\mathbf{z}) + \mathbf{V}(\boldsymbol{\alpha},\mathbf{z})$$

- Given the energy model we can obtain a segmentation by finding

$$\hat{\alpha} = \arg\min_{\alpha} E(\alpha,\theta)$$

- Which can be solved using a minimum cut algorithm which gives you a hard segmentation, $\boldsymbol{\alpha} = \{0,1\}$, of the object.

How GrabCut adds to Graph Cut

- The monochrome image model is replaced for color by a Gaussian Mixture Model (GMM) in place of histograms.
- One shot min-cut solution is replaced by an iterative procedure that alternates between estimation and parameter learning
- Allow for incomplete labeling, i.e. the user need only specify the background trimap T_B (and implicitly the unknown map T_U)

- This amounts to one less user interaction step that was required in previous versions.

From this …

[Specifying foreground and background]

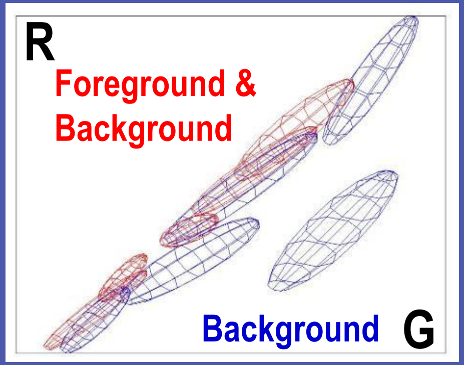
To this …

[Specifying background only]

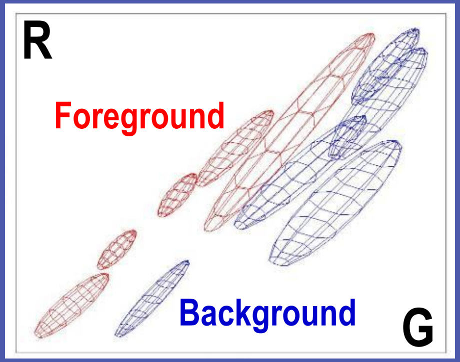
Adding the Color Model

- Each pixel z_n is now in RGB color space
- Color space histograms are impractical so we use a Gaussian Mixture Model (GMM)
  - 2 Full-covariance Gaussian mixtures with K components (K ~ 5).
  - One for foreground, one for background.
- Add to our model a vector $\boldsymbol{k} = \{ k_1 \ldots k_N \}$, with k_i in $\{1 \ldots K\}$
- k_i assigns the pixel z_i to a unique GMM component (Either from F.G. or B.G. as α dictates)

Colour Model



Iterated
graph cut



Gaussian Mixture Model (typically 5-8 components)

Source: Rother

New Energy Model

- Must incorporate ***k*** into our model:

$$\mathbf{E}(\boldsymbol{\alpha},\mathbf{k},\boldsymbol{\theta},\mathbf{z}) = \mathbf{U}(\boldsymbol{\alpha},\mathbf{k},\boldsymbol{\theta},\mathbf{z}) + \mathbf{V}(\boldsymbol{\alpha},\mathbf{z})$$

where

$$\mathbf{U}(\boldsymbol{\alpha},\mathbf{k},\boldsymbol{\theta},\mathbf{z}) = \sum_{n} D(\alpha_n,k_n,\theta,z_n)$$

- $D(\alpha_n,k_n,\theta_n,z_n) = -\log p(z_n \mid \alpha_n,k_n,\theta) - \log \pi(\alpha_n,k_n)$
- Where $\pi(\cdot)$ is a set of mixture weights which satisfy the constraint:

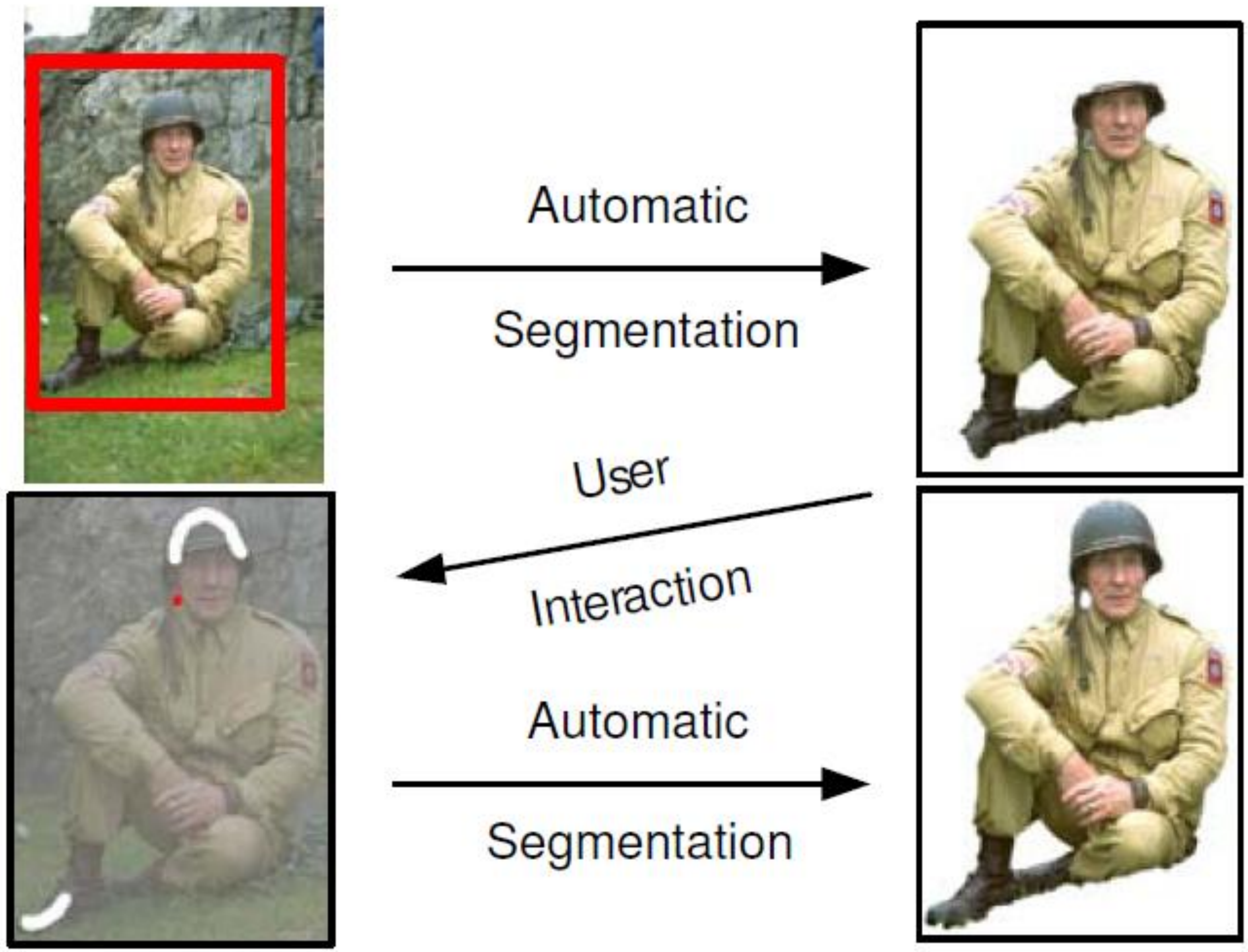
$$D(\alpha_n,k_n,\underline{\theta},z_n) = -\log \pi(\alpha_n,k_n) + \frac{1}{2}\log\det\Sigma(\alpha_n,k_n) + \frac{1}{2}[z_n - \mu(\alpha_n,k_n)]^\top \Sigma(\alpha_n,k_n)^{-1}[z_n - \mu(\alpha_n,k_n)].$$

New Energy Model

- Our θ becomes

$$\theta=\{\underset{\text{weights}}{\underset{\uparrow}{\pi(\alpha,k)}},\ \underset{\text{means}}{\underset{\uparrow}{\mu(\alpha,k)}},\ \underset{\text{cov.}}{\underset{\uparrow}{\Sigma(\alpha,k)}},\ \underset{\text{fg/bg.}}{\underset{\uparrow}{\alpha=0,1}},\ \underset{\text{mixture component}}{\underset{\uparrow}{k=1\ldots K}}\}$$

- Total of 2K Gaussian components


Automatic
Segmentation
User
Interaction
Automatic
Segmentation

Initialisation

- User initialises trimap T by supplying only T_B. The foreground is set to $T_F = \emptyset$; $T_U = \overline{T}_B$, complement of the background.
- Initialise $\alpha_n = 0$ for $n \in T_B$ and $\alpha_n = 1$ for $n \in T_U$.
- Background and foreground GMMs initialised from sets $\alpha_n = 0$ and $\alpha_n = 1$ respectively.

Iterative minimisation

1. *Assign GMM components to pixels:* for each n in T_U,
$$k_n := \arg\min_{k_n} D_n(\alpha_n, k_n, \theta, z_n).$$
2. *Learn GMM parameters* from data $\mathbf{z}$:
$$\underline{\theta} := \arg\min_{\underline{\theta}} U(\underline{\alpha}, \mathbf{k}, \underline{\theta}, \mathbf{z})$$
3. *Estimate segmentation:* use min cut to solve:
$$\min_{\{\alpha_n:\, n \in T_U\}} \min_{\mathbf{k}} \mathbf{E}(\underline{\alpha}, \mathbf{k}, \underline{\theta}, \mathbf{z}).$$
4. Repeat from step 1, until convergence.
5. Apply border matting (section 4).

User editing

- *Edit:* fix some pixels either to $\alpha_n = 0$ (background brush) or $\alpha_n = 1$ (foreground brush); update trimap T accordingly. Perform step 3 above, just once.
- *Refine operation:* [optional] perform entire iterative minimisation algorithm.

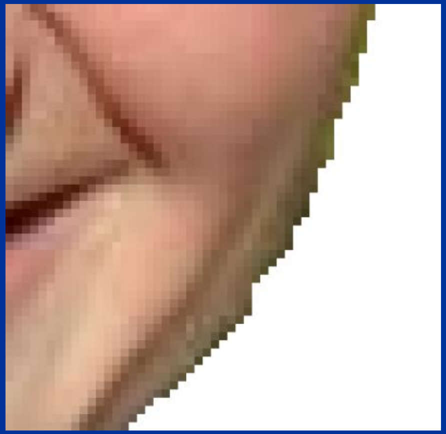
Moderately straightforward examples

… GrabCut completes automatically

Difficult Examples

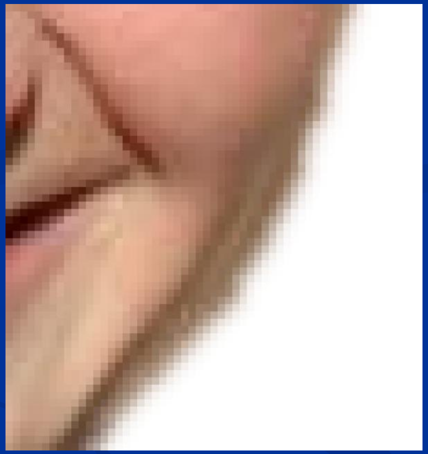
	Camouflage & Low Contrast	Fine structure	Harder Case
Initial Rectangle			
Initial Result			

Border Matting



Hard Segmentation

Automatic Trimap

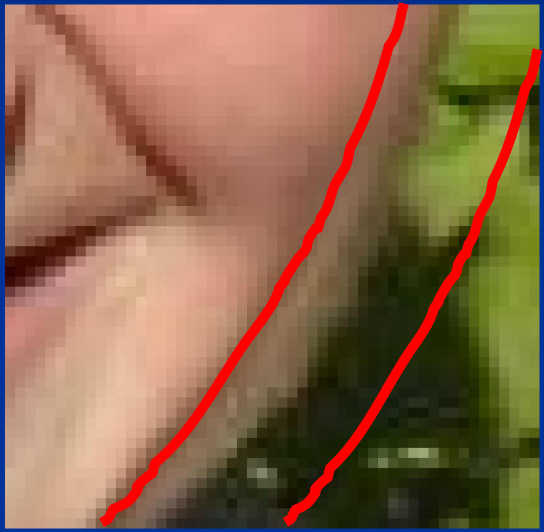


Soft Segmentation

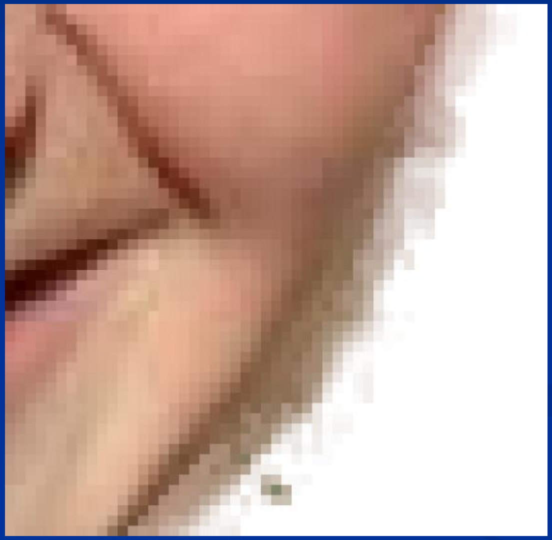
to

Comparison

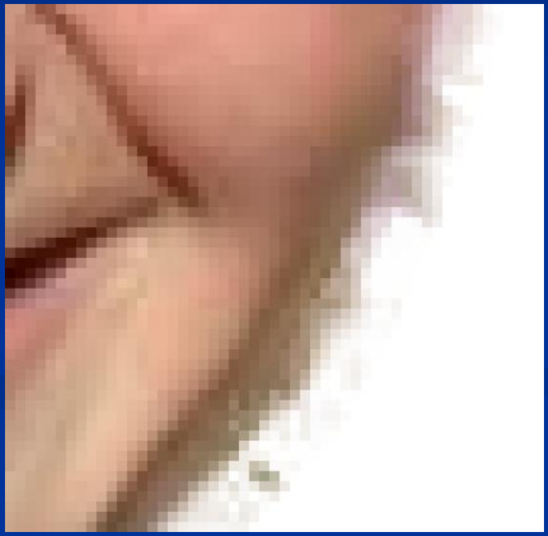
With no regularisation over alpha



Input



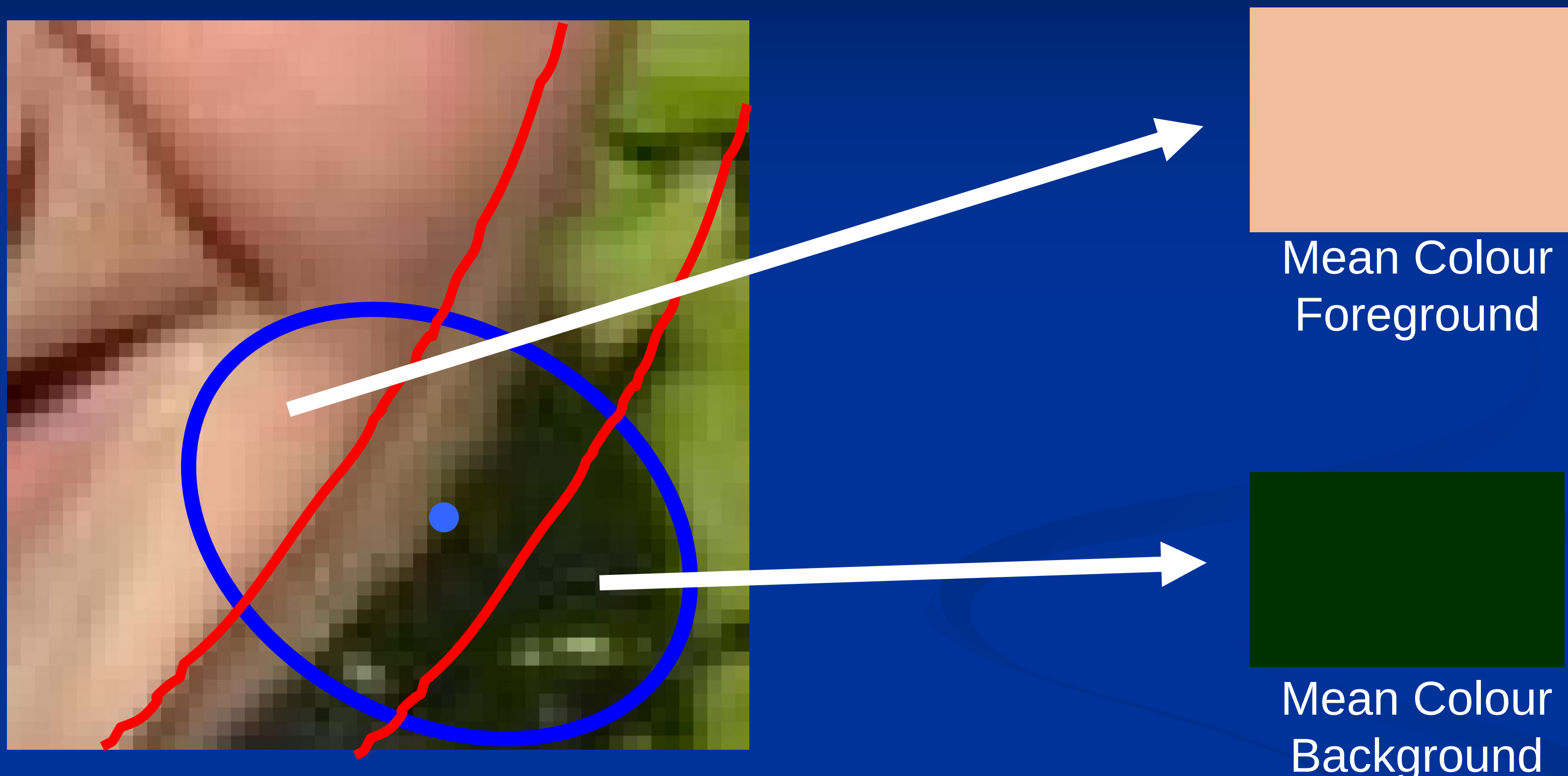
Bayes Matting
Chuang et. al. (2001)



Knockout 2
Photoshop Plug-In

Shum et. al. (2004): Coherence matting in "Pop-up light fields"

Natural Image Matting



Solve

Ruzon and Tomasi (2000): Alpha estimation in natural images

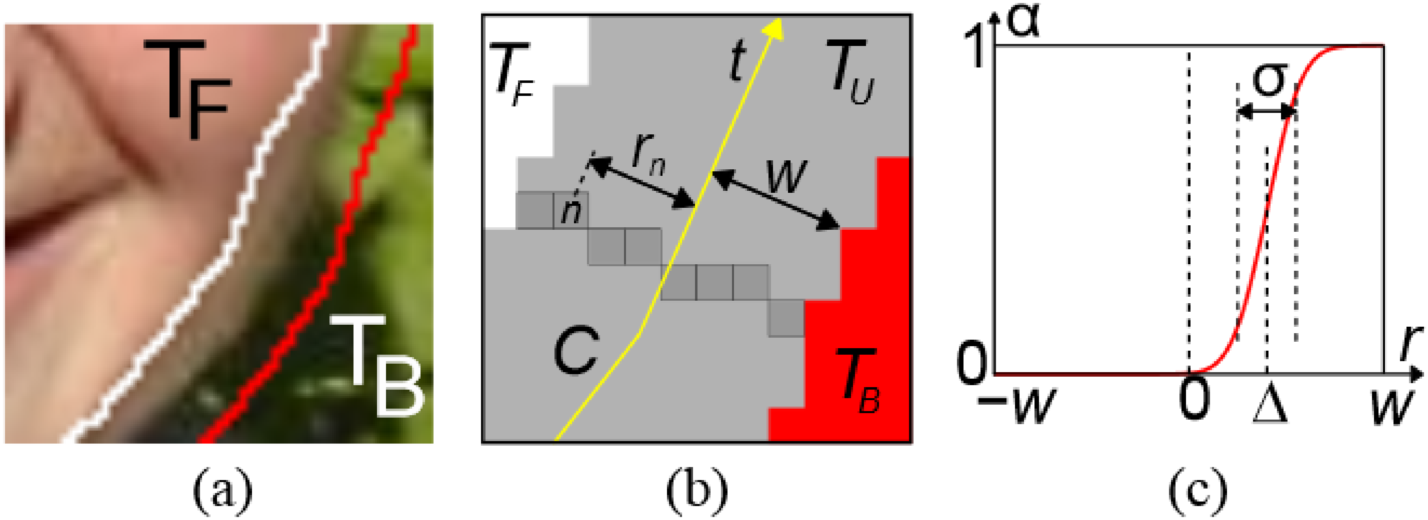
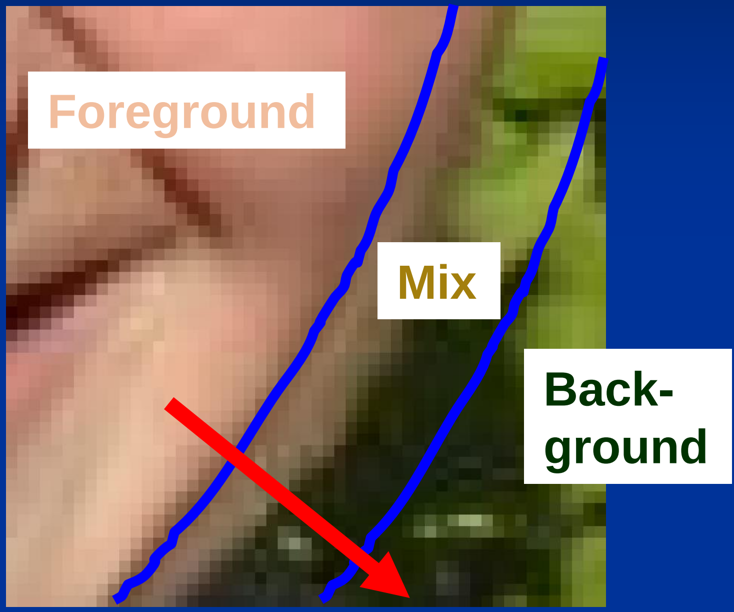
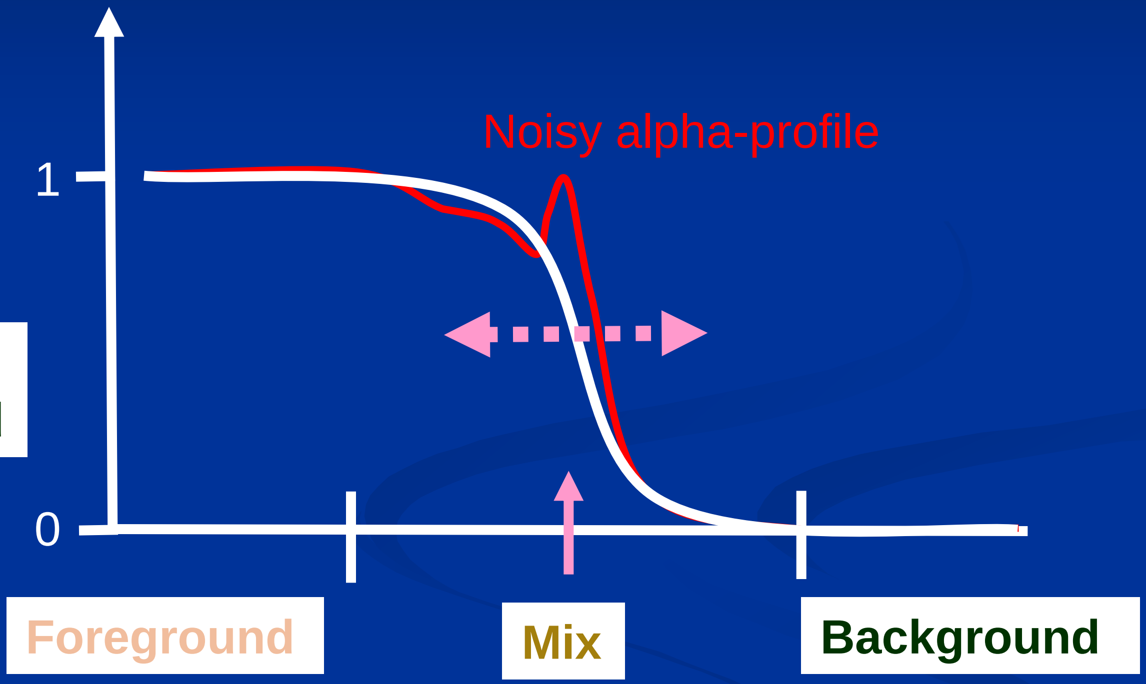


Figure 6: **Border matting.** (a) Original image with trimap overlaid. (b) Notation for contour parameterisation and distance map. Contour C (yellow) is obtained from hard segmentation. Each pixel in T_U is assigned values (integer) of contour parameter t and distance r_n from C. Pixels shown share the same value of t. (c) Soft step-function for α-profile g, with centre Δ and width σ.

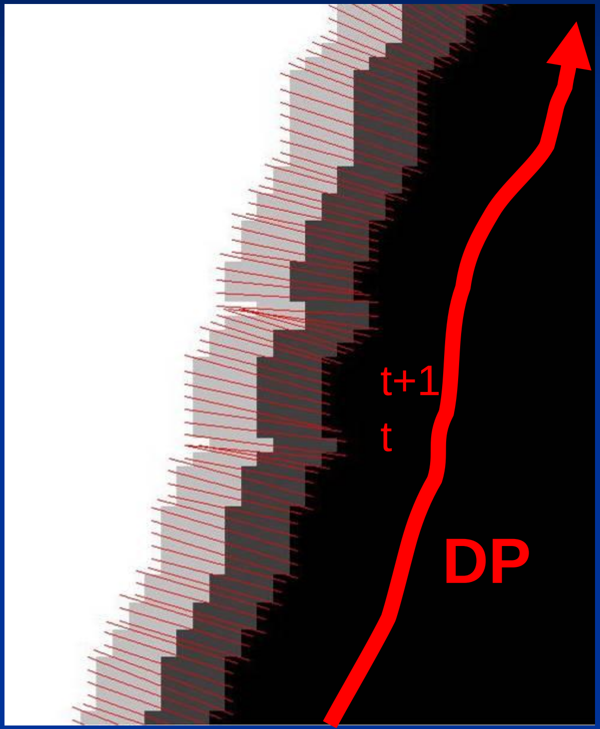
Border Matting

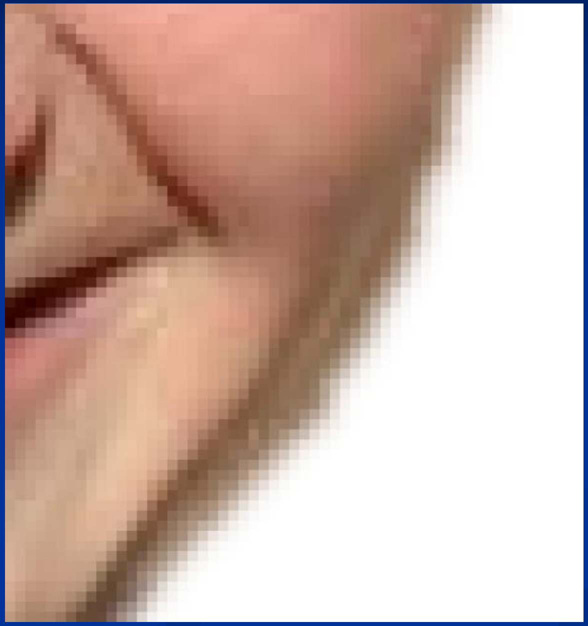




Fit a smooth alpha-profile with parameters

Dynamic Programming



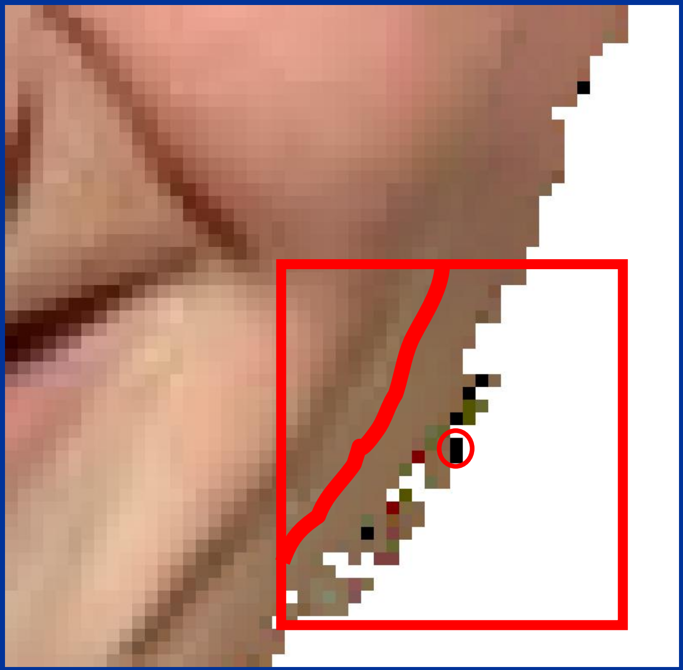


Result using DP Border Matting

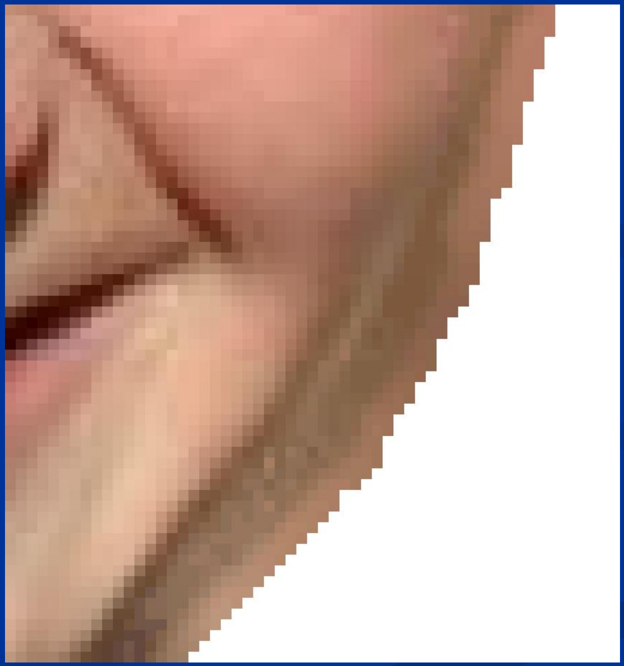
$$E = \underbrace{\sum_{n \in T_U} \tilde{D}_n(\alpha_n)}_{\text{Noisy alpha-profile}} + \underbrace{\sum_{t=1}^{T} \tilde{V}(\Delta_t, \sigma_t, \Delta_{t+1}, \sigma_{t+1})}_{\text{Regularisation}}$$

GrabCut Border Matting - Colour

- Compute MAP of p(F|C,alpha) (marginalize over B)
- To avoid colour bleeding use colour stealing
  ("exemplar based inpainting" – Patches do not work)



[Chuang et al. '01]



Grabcut Border Matting

Results