# 泊松图像编辑与交互式数字蒙太奇

**章国锋**

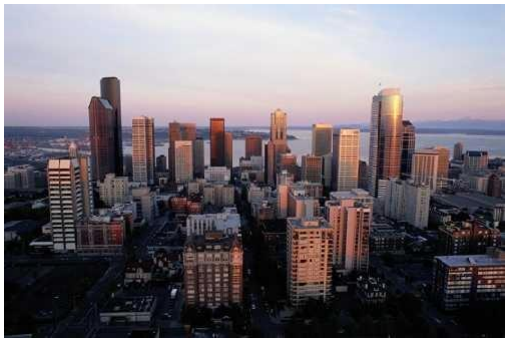
**浙江大学CAD&CG国家重点实验室**

# Image Blending

# Image Composition

# Compositing Procedure

1. Extract Sprites (e.g using *Intelligent Scissors* in Photoshop)



2. Blend them into the composite (in the right order)

Composite by
David Dewey

# 两种方法

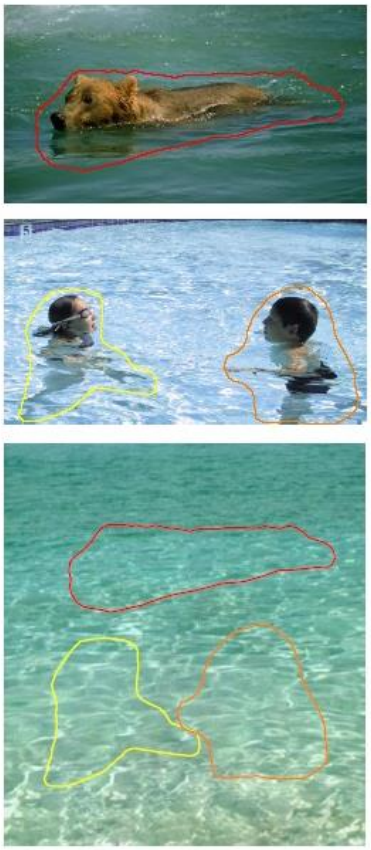
- 泊松图像编辑
- 交互式数字蒙太奇

# 泊松图像编辑

# 泊松图像编辑

sources　destinations　cloning　seamless cloning

# 泊松图像编辑

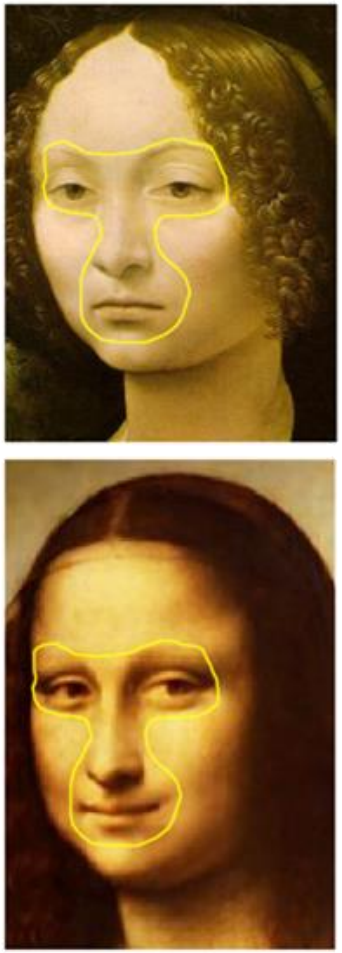


sources/destinations

cloning

seamless cloning

# 泊松图像编辑



source/destination

cloning

seamless cloning

# 泊松图像编辑

- Poisson image editing
  - Patrick Pérez, Michel Gangnet, Andrew Blake
  - SIGGRAPH 2003

Using generic interpolation machinery based on solving **Poisson equations**, a variety of novel tools are introduced **for seamless editing of image regions**.

# 知识点概括

- 偏微分方程PDE
- 泊松方程与拉普拉斯方程
- 边界条件
- 简单插值
- 导向插值

# 最常用的PDE(Partial Differential Equation)

- Wave Equation

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$$

$$\frac{\partial^2 u}{\partial t^2} = c^2 \nabla^2 u + f(t, x, y, z)$$

- Heat Equation

$$\frac{\partial u}{\partial t} = c^2 \nabla^2 u + f(t, x, y, z)$$

# 续

- Poisson Equation, Steady State of Wave Equation and Heat Equation

$$\nabla^2 u = f(x, y, z)$$

- Laplace's Equation

$$\nabla^2 u = 0$$

# 边界条件Boundary Condition

- Dirichlet Boundary Conditions
  - Specify the value of the function on a surface
- Neumann Boundary Condition
  - Specify the normal derivative of the function on a surface

# 符号说明

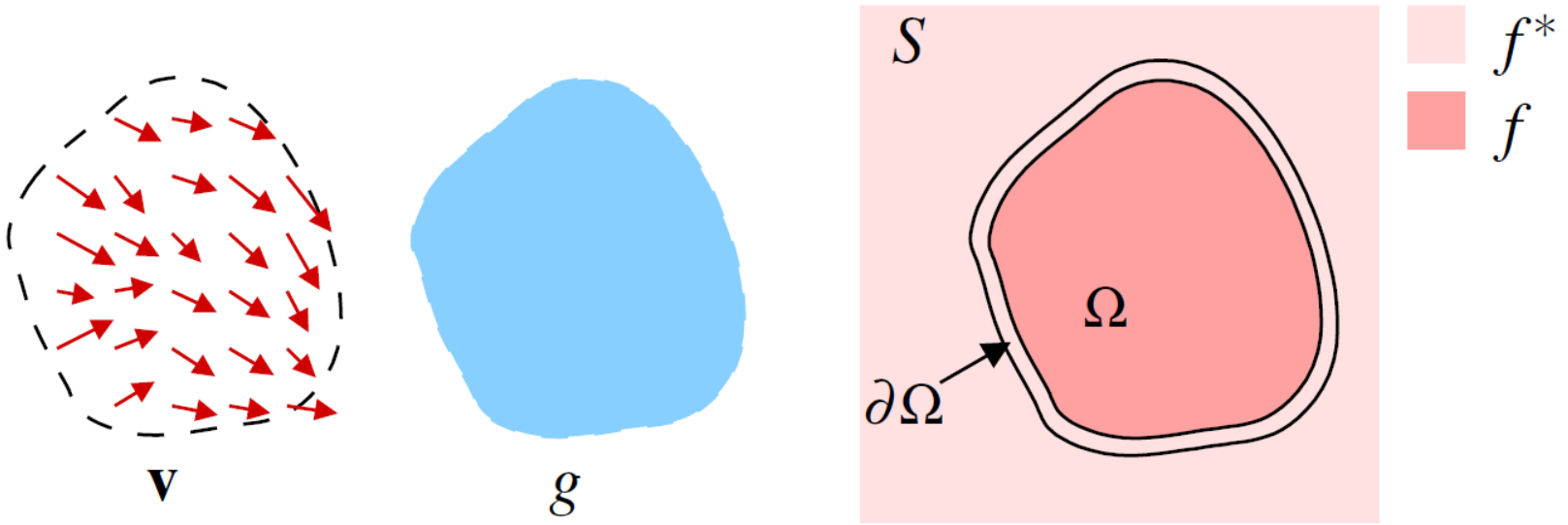



Figure 1: **Guided interpolation notations**. Unknown function $f$ interpolates in domain $\Omega$ the destination function $f^*$, under guidance of vector field $\mathbf{v}$, which might be or not the gradient field of a source function $g$.

# 简单插值Simple Interpolation

- Maximize the Smoothness

$$\begin{cases} \min_f \iint_\Omega \|\nabla f\|^2 \\ f|_{\partial\Omega} = f^*|_{\partial\Omega} \end{cases}$$

- Solution: Laplace Equation with Dirichlet Boundary Conditions

$$\begin{cases} \nabla^2 f = 0 \\ f|_{\partial\Omega} = f^*|_{\partial\Omega} \end{cases}$$

# 导向插值Guided Interpolation

- Interpolation-> minimization

$$\begin{cases} \min_f \iint_\Omega \|\nabla f - \mathbf{v}\|^2 \\ f|_{\partial\Omega} = f^*|_{\partial\Omega} \end{cases}$$

- Solution: Poisson Equation with Dirichlet Boundary Conditions

$$\begin{cases} \nabla^2 f = \nabla \cdot \mathbf{v} \\ f|_{\partial\Omega} = f^*|_{\partial\Omega} \end{cases}$$

- Relationship with Laplace case?

# 求解上述问题

- Discretize the Minimization Directly

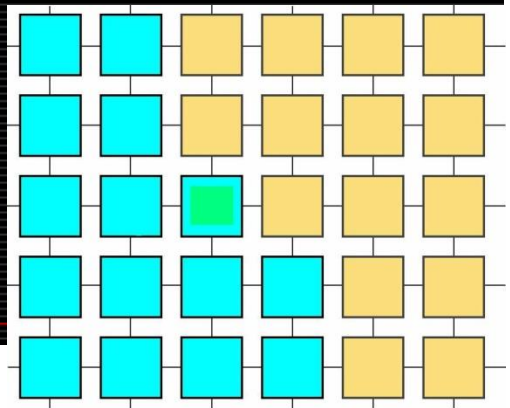
$$\min_{f|\Omega} \sum_{<p,q>\cap\Omega\neq\emptyset} (f_p - f_q - v_{pq})^2 \text{ with } f_p = f_p^* \text{ for } \forall p \in \partial\Omega$$

- Partial Derivative

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

- Partial Derivative for Interior Points

$$|N_p| f_p - \sum_{q\in N_p} f_q = + \sum_{q\in N_p} v_{pq}$$



# 求解上述问题

- Linear System of Equations
- Gauss-Seidel Method with Successive Overrelaxation
- V-cycle Multigrid
- Discretize Laplacian with Discrete Laplacian of Gaussian
- http://www.tau.ac.il/~stoledo/taucs/ Taucs

# 应用：图像的无缝拼接 (Seamless Cloning)

- Importing Gradients from a Source Image

$$g$$

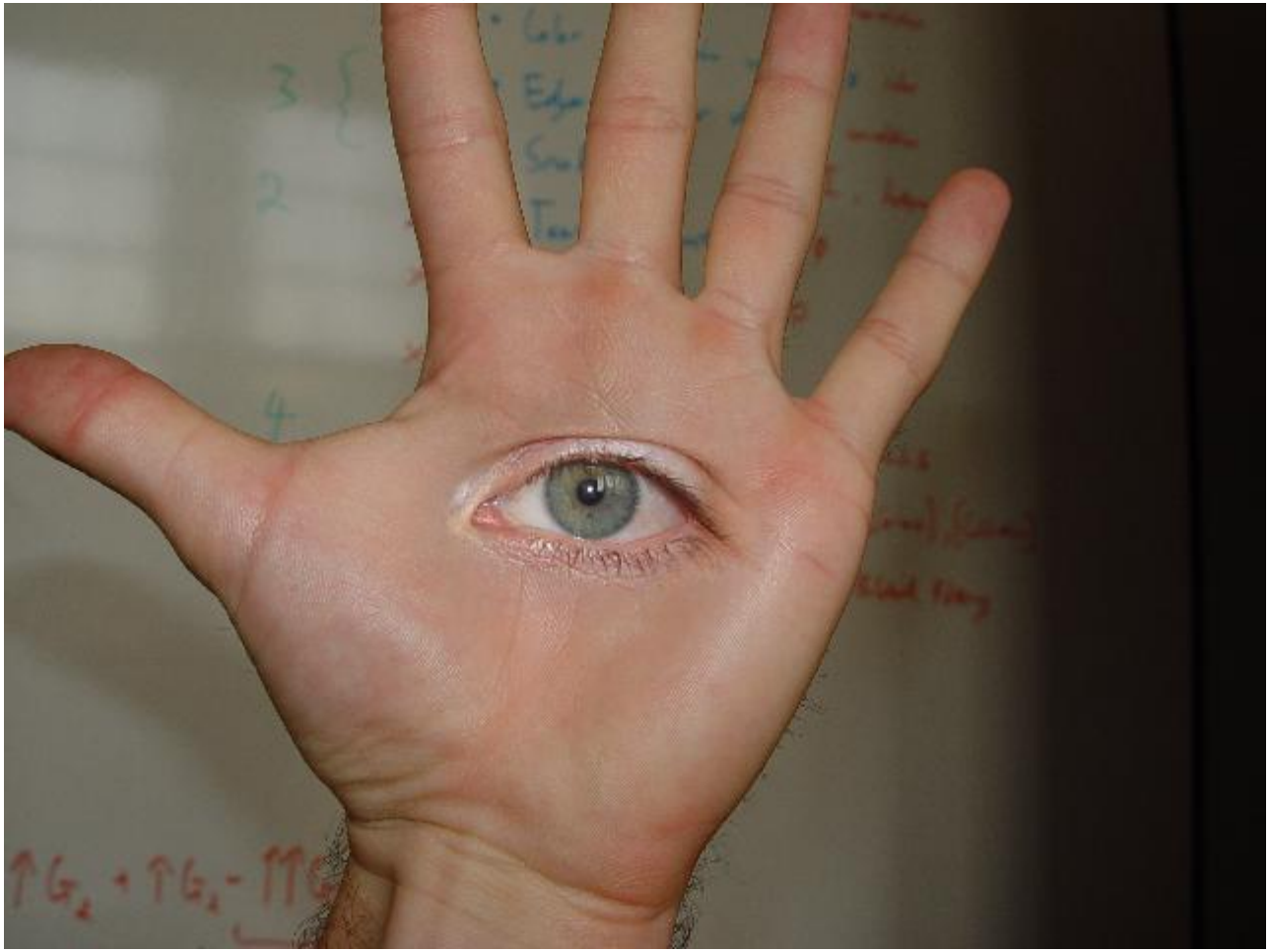
$$\mathbf{v} = \nabla g$$

- Discretize

$$\text{for all } \langle p, q \rangle, v_{pq} = g_p - g_q$$

# Seamless Cloning Results

# Horror Photo



© Chris Cameron

# Texture Transfer

swapped textures

# Seamless Cloning: Mixing Gradients

- Two Proposals
  - Define **v** as Linear Combination of Source and Destination Gradients
  - Select Stronger one from Source and Destination Gradients (not conservative!)

$$\text{for all } \mathbf{x} \in \Omega,\ \mathbf{v}(\mathbf{x}) = \begin{cases} \nabla f^*(\mathbf{x}) & \text{if } |\nabla f^*(\mathbf{x})| > |\nabla g(\mathbf{x})| \\ \nabla g(\mathbf{x}) & \text{otherwise.} \end{cases}$$

  - Discretization

$$v_{pq} = \begin{cases} f_p^* - f_q^* & \text{if } |f_p^* - f_q^*| > |g_p - g_q| \\ g_p - g_q & \text{otherwise,} \end{cases}$$

# Mixing Gradients Results

(a) color-based cutout and paste

(b) seamless cloning

(c) seamless cloning and destination averaged

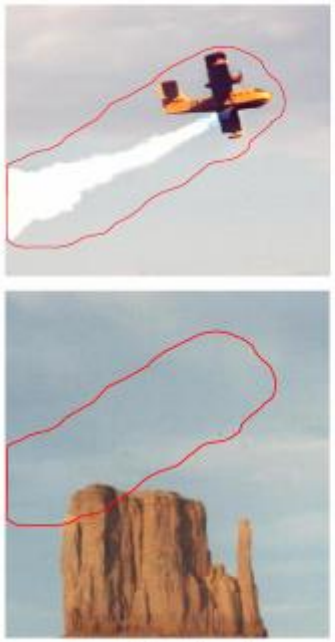
(d) mixed seamless cloning

# Mixing Gradients Results

source

destination

# Mixing Gradients Results



source/destination

seamless cloning

mixed seamless cloning

# Texture Flattening

- Remain Only Salient Gradients

$$\text{for all } \mathbf{x} \in \Omega,\ \mathbf{v}(\mathbf{x}) = M(\mathbf{x}) \nabla f^*(\mathbf{x})$$

- Discretization

$$v_{pq} = \begin{cases} f_p - f_q & \text{if an edge lies between } p \text{ and } q \\ 0 & \text{otherwise,} \end{cases}$$

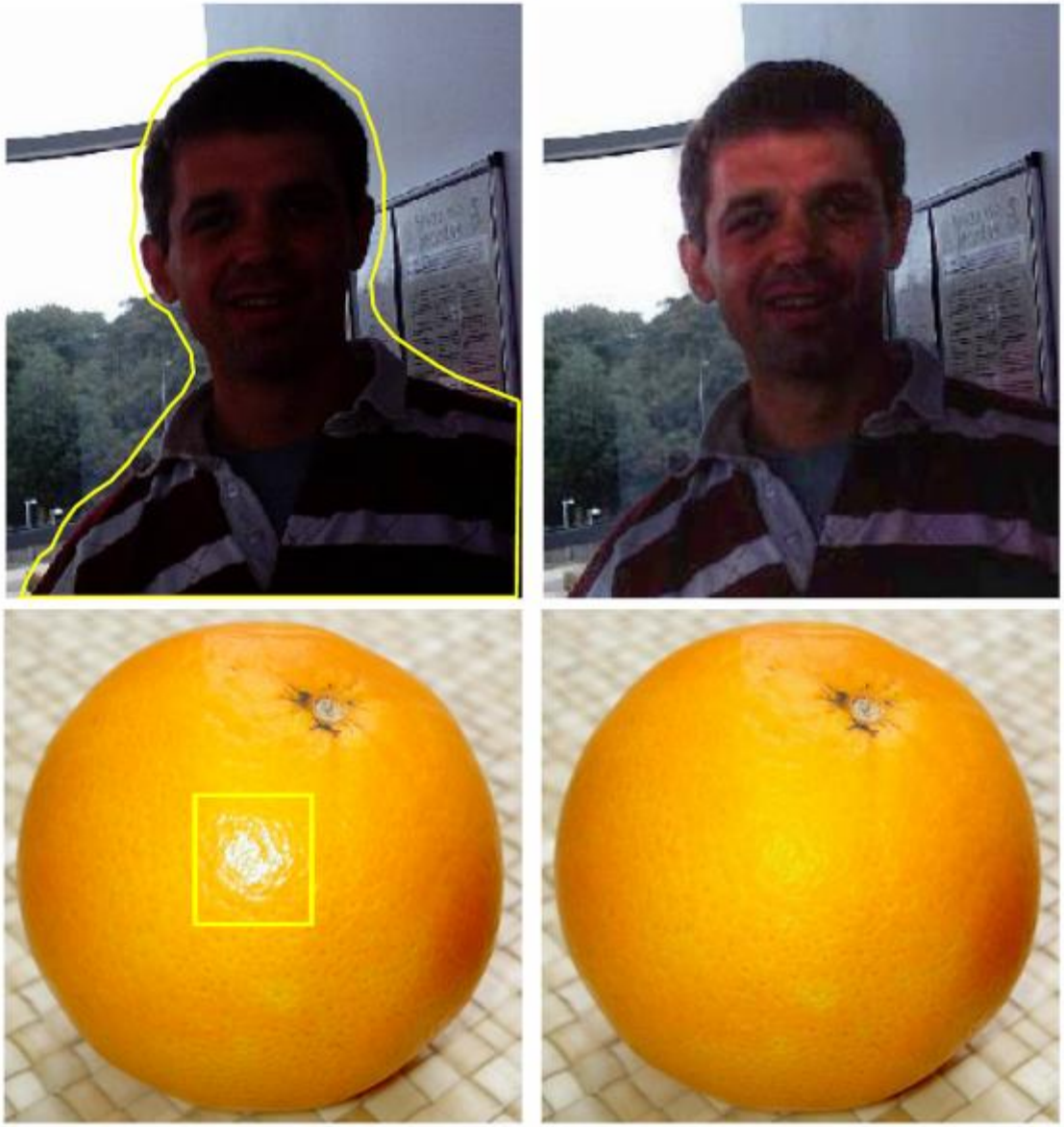
# Texture Flattening

Edge mask

# Local Illumination Changes

- Fattal Transformation

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



# Local Color Changes

- Mix two different colored version of original image
  - One provide $f^*$ outside
  - One provide g inside

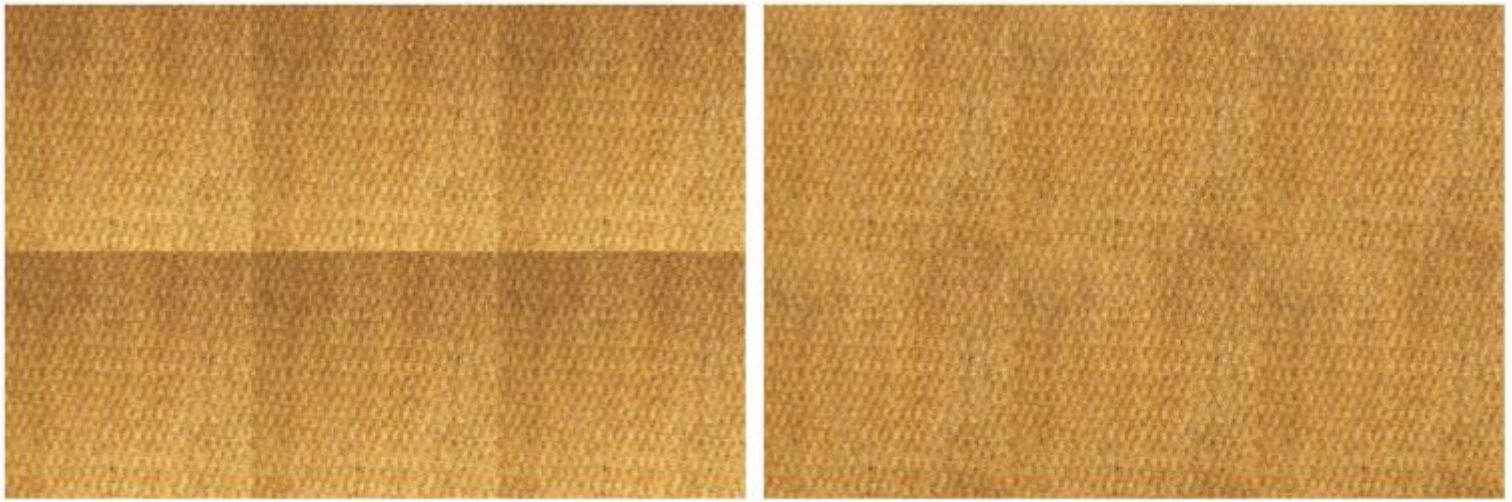
# Local Color Changes

Figure 11: **Local color changes**. Left: original image showing selection $\Omega$ surrounding loosely an object of interest; center: background decolorization done by setting $g$ to the original color image and $f^*$ to the luminance of $g$; right: recoloring the object of interest by multiplying the RGB channels of the original image by 1.5, 0.5, and 0.5 respectively to form the source image.

# Seamless Tiling

- Select original image as g
- Boundary condition:
  - $f^*_{north}=f^*_{south}=0.5(g_{north}+g_{south})$
  - Similarly for the east and west

# Seamless Tiling



# 交互式数字蒙太奇

# Interactive Digital Photomontage

Aseem Agarwala, Mira Dontcheva
Maneesh Agrawala, Steven Drucker, Alex Colburn
Brian Curless, David Salesin, Michael Cohen

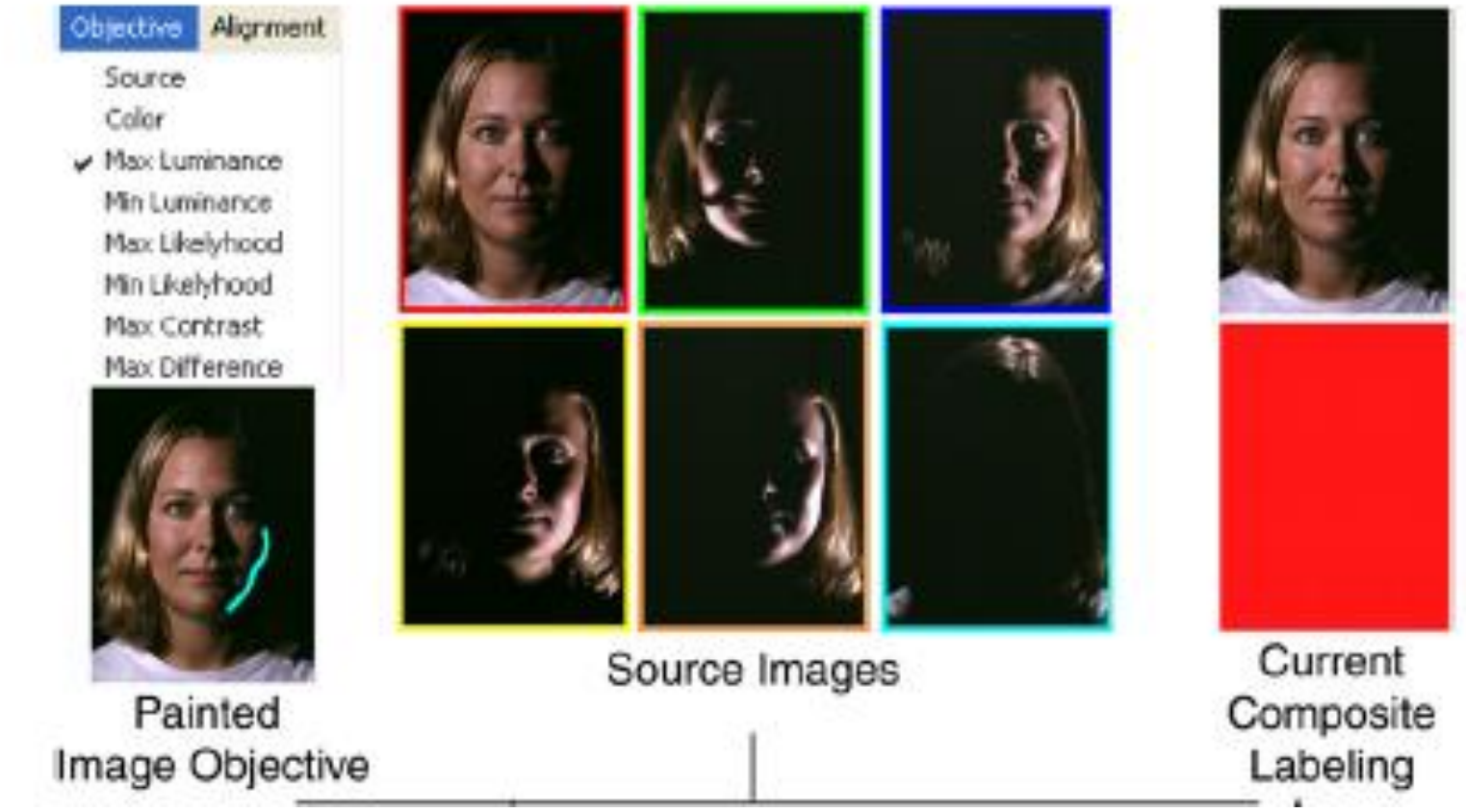





# The Photomontage Framework

- begins with a set of source images( ***image stack*** ).

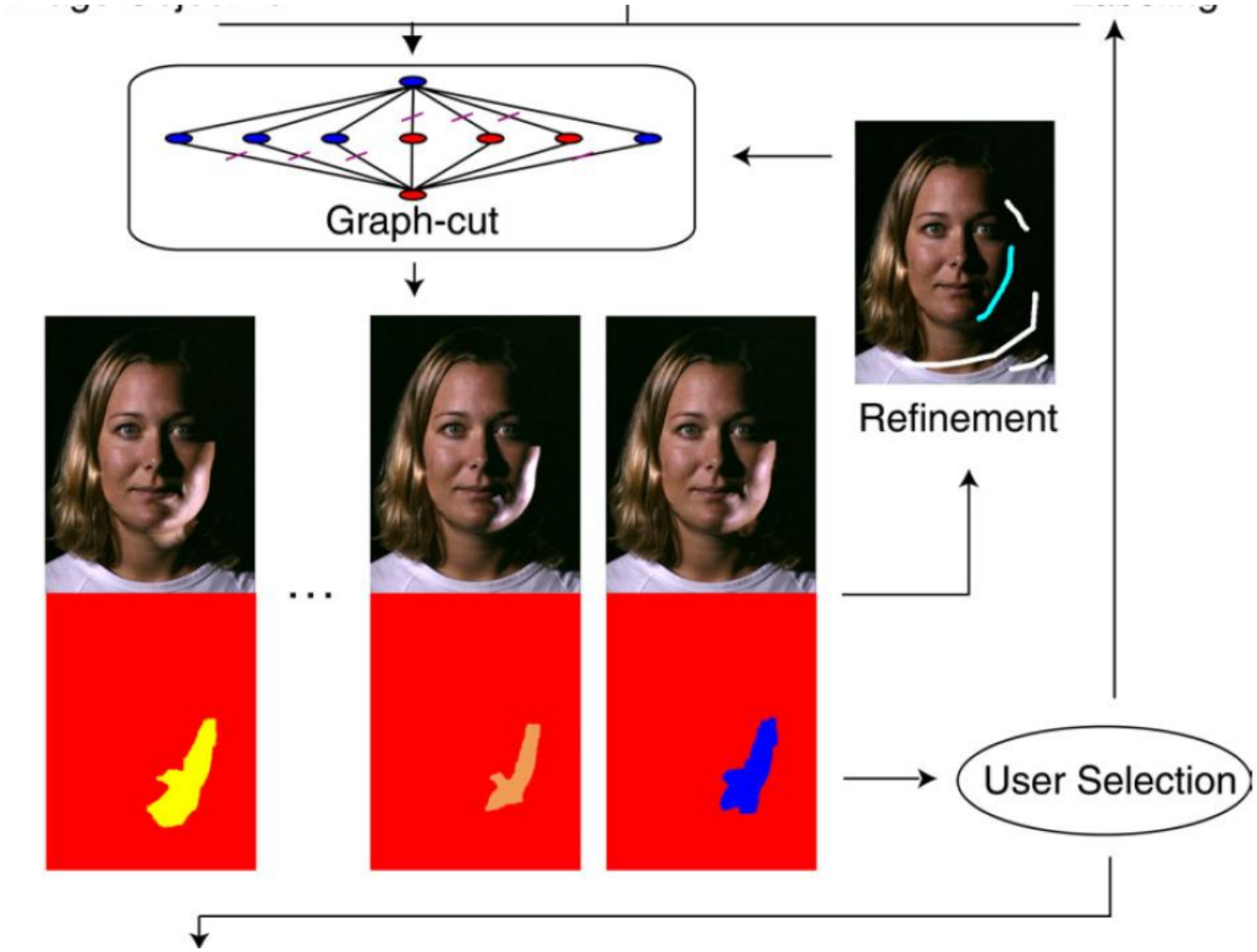
# Image stack


Objective
Alignment
Source
Color
Max Luminance
Min Luminance
Max Likelyhood
Min Likelyhood
Max Contrast
Max Difference
Painted
Image Objective
Source Images
Current
Composite
Labeling

# Brush

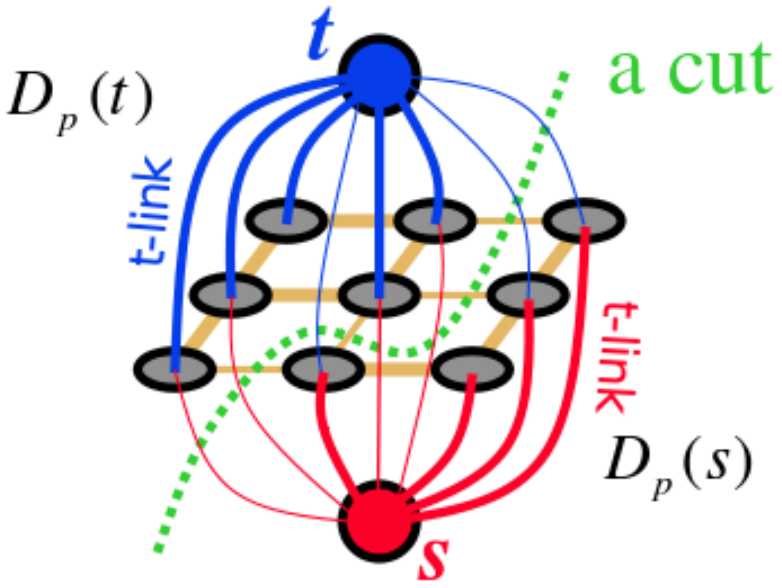
- User brushes a specific label onto each image.
- The user goes through an iterative refinement process to create a composite. Associated with the composite is a ***labeling***.

# Brush and Refine



# Graph Cuts

- Definition & Notation
  - 对于一个图 $G = (V, E)$，其中 $V$ 为节点集合，包括源点$s$和终点$t$、以及其他诸多中间节点集合$V'$，$E$ 为连接这些节点的边，每条边附有容量$c(u, v)$ 代表节点$u$通过这条边流向节点$v$所能承受的最大流量。
  - Graph cuts的目的在于找到图的Min-cut，Cut将 $V'$ 分割为两个部分，去掉这些边将使舍得图中的任意一个节点只与$s$或$t$相连通，而Min-cut是所有cut中边的能量值总和最小的一个。




$$E(f) = \sum_{p \in P} D_p(f_p) + \sum_{p,q \in N} V_{p,q}(f_p, f_q)$$

# Graph Cuts

- Recommended Paper
  - Yuri Boykov, Olga Veksler, Ramin Zabih. Fast Approximate Energy Minimization via Graph Cuts. IEEE Trans. Pattern Anal. Mach. Intell. 23(11): 1222-1239, 2001.
- Graph Cuts Home Page
  - http://www.cs.cornell.edu/~rdz/graphcuts.html
- Source code:

  http://www.cs.ucl.ac.uk/staff/V.Kolmogorov/software.html

# An Introduction to Graph-Cut

- Graph-cut is an algorithm that finds a globally optimal
- Segmentation solution.
- Also know as Min-cut.
- Equivalent to Max-flow. [1]

  For any network having a single origin and single destination node, the maximum possible flow from origin to destination equals the minimum cut value for all cuts in the network.

[1] Wu and Leahy: An Optimal Graph Theoretic Approach to Data Clustering:…

# What is a “cut”?

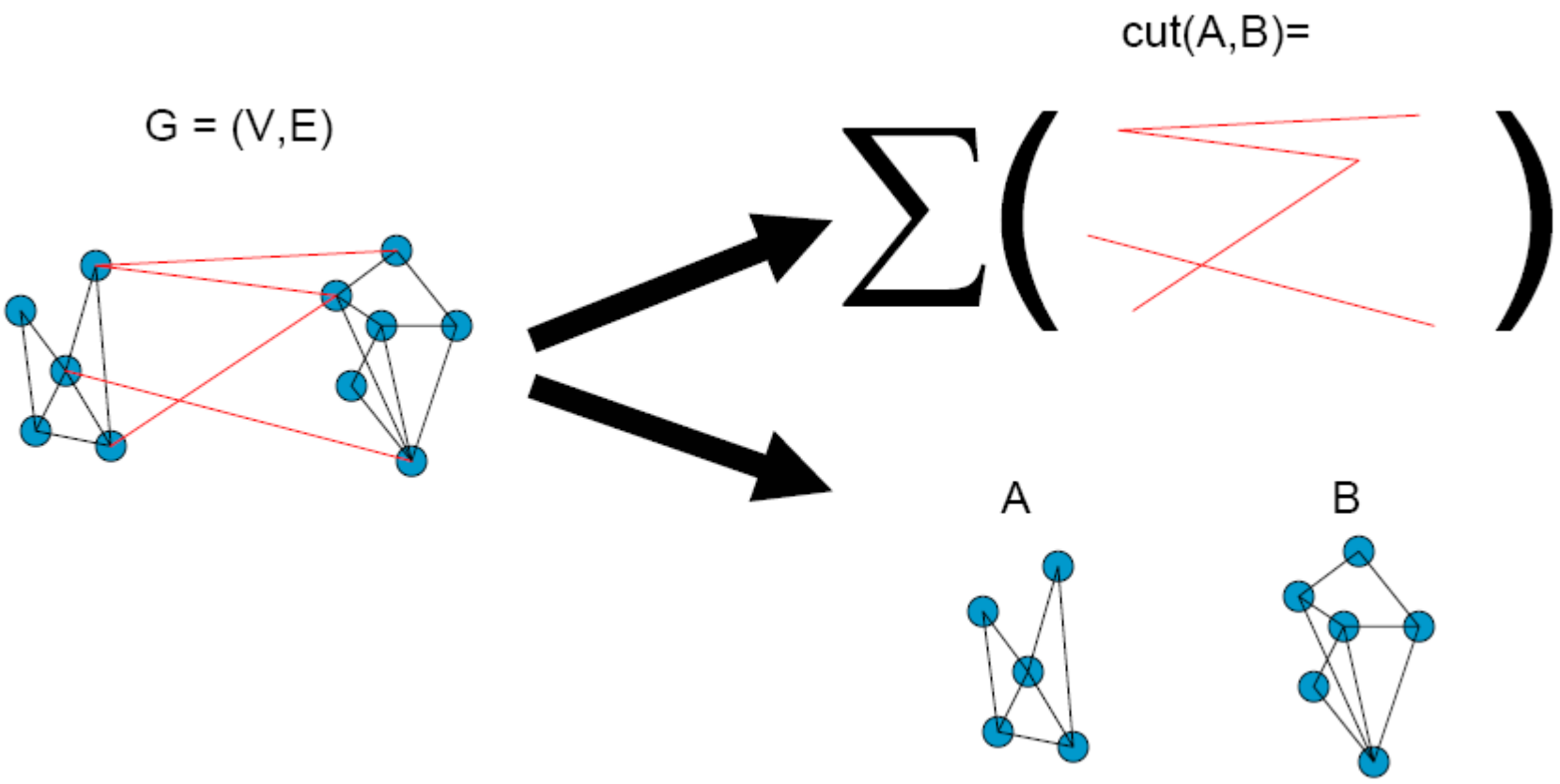
A graph G = (V,E) can be partitioned into two disjoint sets, $A, B, A \cup B \models V, A \cap B = 0$
by simply removing edges connecting the two parts.

The degree of dissimilarity between these two pieces can be computed as total weight of the edges that have been removed. In graph theoretic language it is called the *cut*:

$$cut(A,B) = \sum_{u \in A, v \in B} w(u,v)$$ [2].

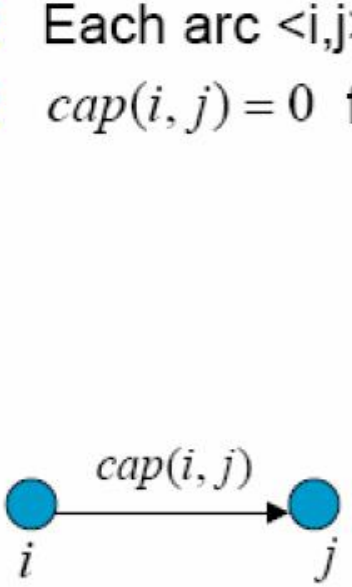
[2] Shi and Malik: Normalized cuts and image segmentation.

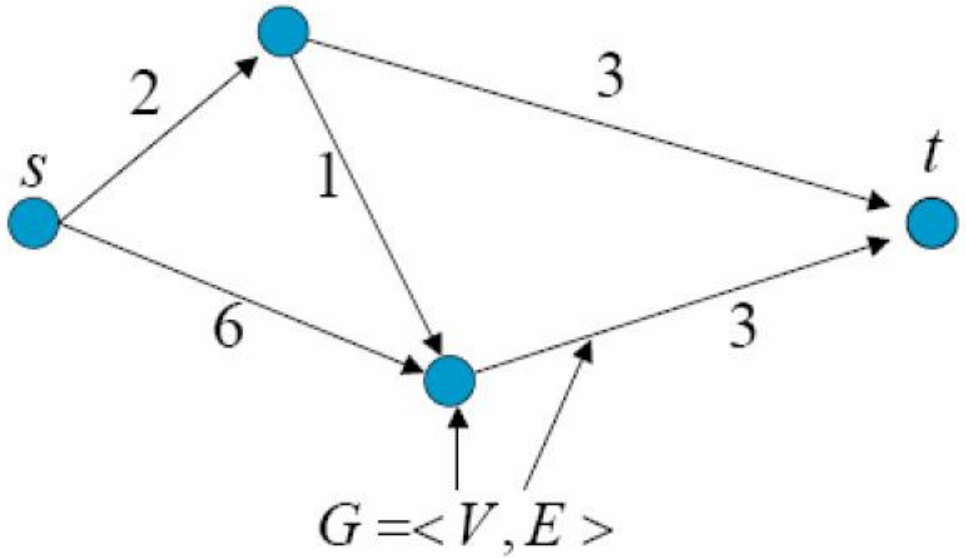
# Example cut



# Finding the Minimum-cut

- A source node and a sink node
- Directed edge (Arc) <i,j> from node i to node j
- Each arc <i,j> has a nonnegative capacity cap(i,j)
- $cap(i, j) = 0$ for non-exist arcs



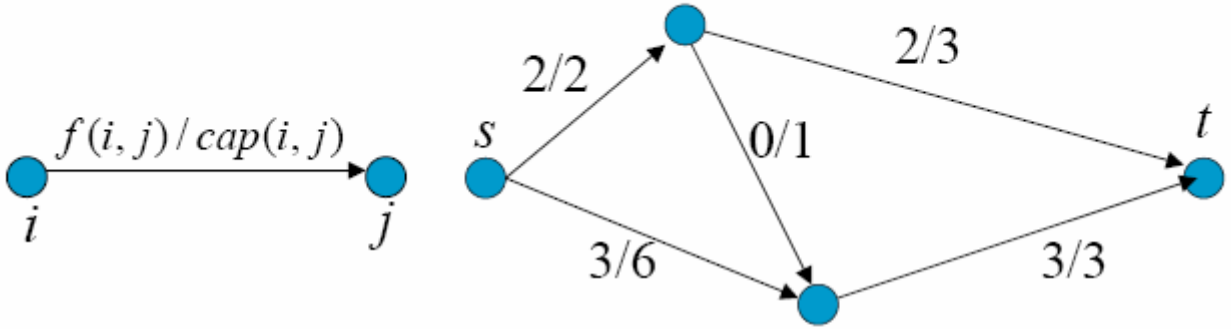
# Finding the Minimum-cut

- Flow is a real value function $f$ that assign a real value $f(i,j)$ to each arc <i,j> under :
  - Capacity constraint : $f(i,j) \le cap(i,j)$
  - Mass balance constraint:

$$\sum_{<i,j>\in E} f(i,j) - \sum_{<k,i>\in E} f(k,i) = \begin{cases} 0 & i \in V - \{s,t\} \\ |f| & i = s \\ -|f| & i = t \end{cases}$$

$|f|$ is the value of flow $f$

# Finding the Minimum-cut

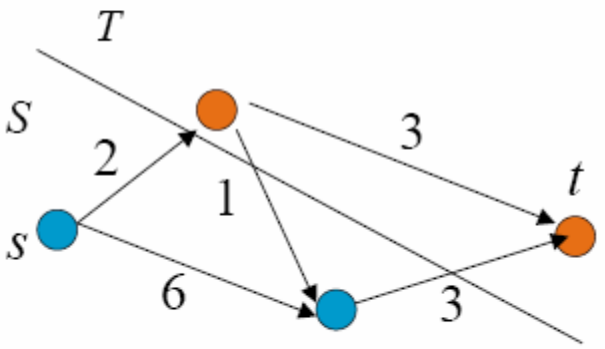



An example of flow

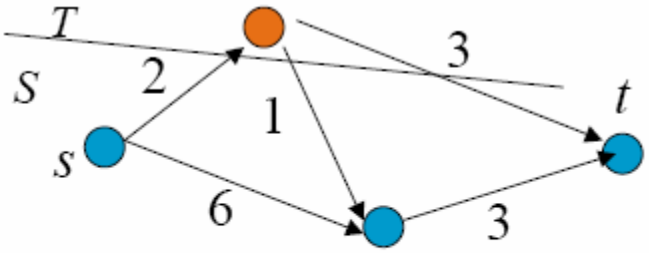
- maximum flow is the flow has maximum value among all possible flow functions

# Finding the Minimum-cut

- A cut is a partition of node set V which has two subsets S and T
- A cut is a s-t cut iif $s \in S, t \in T$



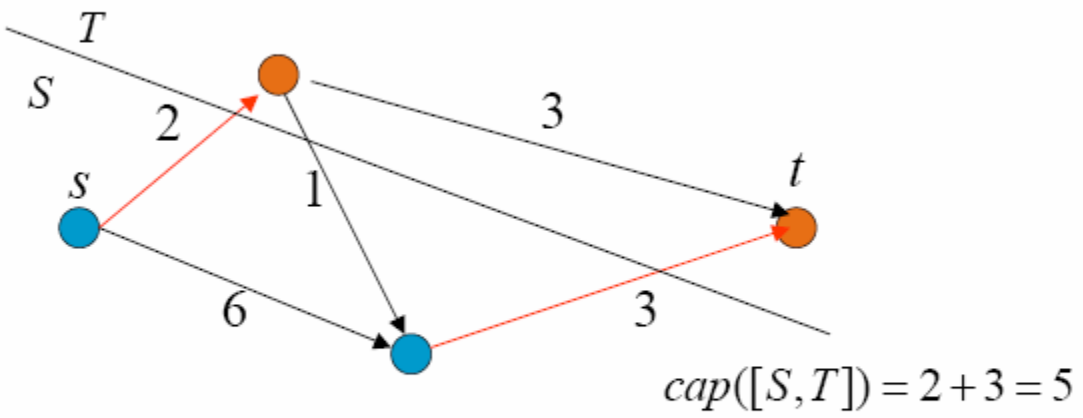

s-t cut




not a s-t cut

# Finding the Minimum-cut

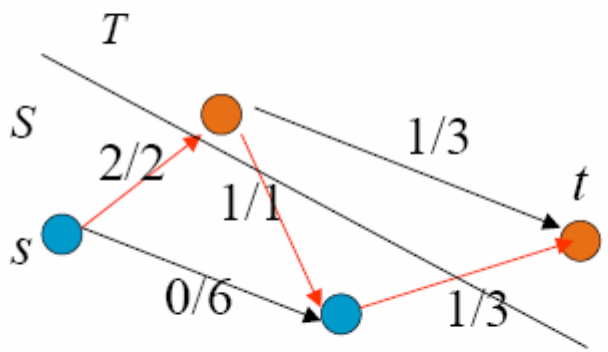
- $cap([S,T]) = \sum_{<i,j>\in E, i\in S, j\in T} cap(i,j)$




- Minimum cut is the s-t cut whose capacity is minimum among all possible s-t cuts

# Finding the Minimum-cut

- $$f([S,T]) = \sum_{<i,j>\in E, v_i \in S, v_j \in T} f(i,j) - \sum_{<j,i>\in E, v_i \in S, v_j \in T} f(j,i)$$

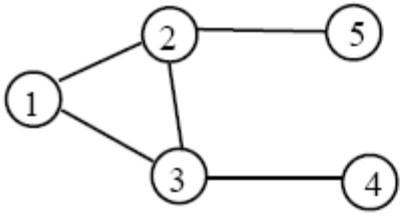



$$f([S,T]) = 2+1-1 = 2$$

# Pixel labeling problem

## Given

$\mathcal{S} = \{1, \cdots, n\}$ $\mathcal{N} \subseteq \mathcal{S} \times \mathcal{S}$



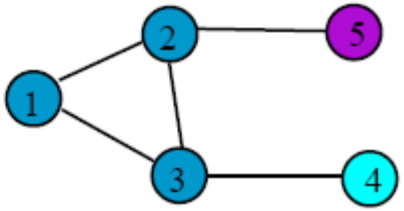

$\mathcal{L} = \{l_1, \ldots, l_m\}$ 

*Assignment cost* for giving a particular label to a particular node. Written as D.

*Separation cost* for assigning a particular pair of labels to neighboring nodes. Written as V.

## Find

Labeling f = ($f_1$,…,$f_n$)




Such that the sum of the assignment costs and separation costs (the energy E) is small

# Energy Minimization

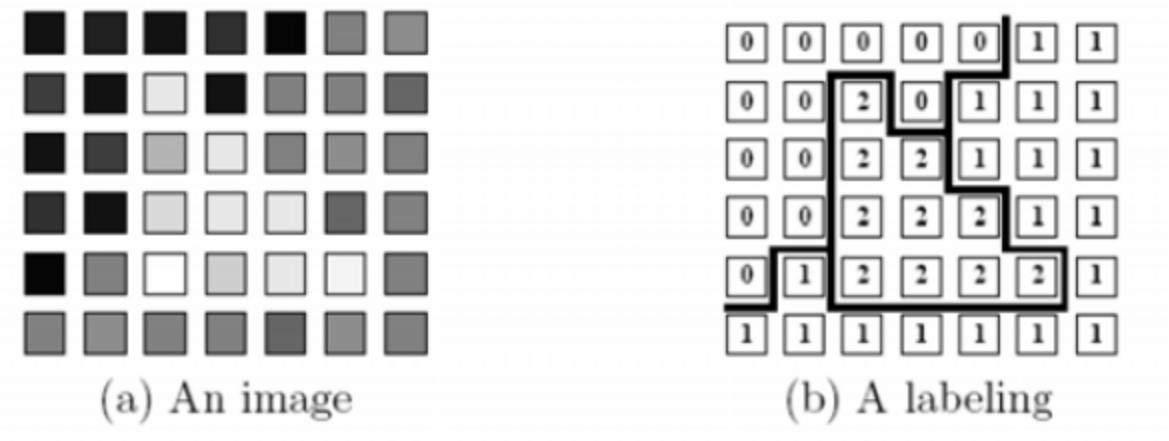
Optimizing the labeling problem can be thought of as minimizing some energy function.

$$E(f) = \sum_{p \in P} D_p(f_p) + \sum_{p,q \in N} V_{p,q}(f_p, f_q)$$

$D_p(f_p)$ measure of image discrepancy

$\sum_{p,q \in N} V_{p,q}(f_p, f_q)$ measure of smoothness or other visual constraints

# The Labeling Problem




(a) An image (b) A labeling

Common idea behind many Computer Vision problems

Assign labels to pixels based on noisy measurements (input images)

In the presence of uncertainties, find the best Labeling !

(Stereo, 3D Reconstruction, Segmentation, Image Restoration)

# Multi-Label Graph-Cuts

- $\alpha - \beta$ Swap
  - Semi-metric

$$V(\alpha, \beta) = V(\beta, \alpha) \geq 0 \quad \text{and} \quad V(\alpha, \beta) = 0 \Leftrightarrow \alpha = \beta.$$

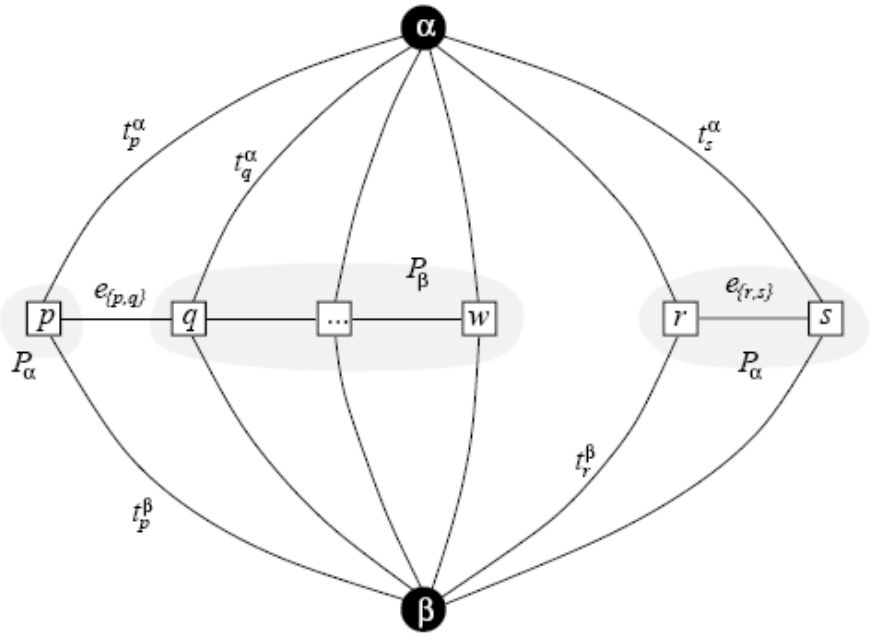
- $\alpha -$ expansion
  - Metric

If $V$ also satisfies the triangle inequality

$$V(\alpha, \beta) \leq V(\alpha, \gamma) + V(\gamma, \beta)$$

# $\alpha - \beta$ Swap

1. Start with an arbitrary labeling $f$
2. Set success := 0
3. For each pair of labels $\{\alpha, \beta\} \subset \mathcal{L}$
   - 3.1. Find $\hat{f} = \arg\min E(f')$ among $f'$ within one $\alpha$-$\beta$ swap of $f$
   - 3.2. If $E(\hat{f}) < E(f)$, set $f := \hat{f}$ and success := 1
4. If success = 1 goto 2
5. Return $f$



# Minimize the objective

- For the task of image composition. The goal of the refinement is to minimize a penalty function.

In our case, we define the *cost function* $C$ of a pixel labeling $L$ as the sum of two terms: a *data penalty* $C_d$ over all pixels $p$ and an *interaction penalty* $C_i$ over all pairs of neighboring pixels $p, q$:

$$C(L) = \sum_{p} C_d(p, L(p)) + \sum_{p,q} C_i(p, q, L(p), L(q)) \quad (1)$$

# Data Penalty

- For the task of image composition, the data penalty is:

$$C_d\big(p, L(p)\big) = \begin{cases} 0 & L(p) = u \\ large\ penalty & otherwise \end{cases}$$

# Interactional Penalty

$$C_i(p, q, L(p), L(q)) = \begin{cases} X & \text{if matching “colors”} \\ Y & \text{if matching “gradients”} \\ X+Y & \text{if matching “colors \& gradients”} \\ X/Z & \text{if matching “colors \& edges”} \end{cases}$$

where

$$\begin{aligned} X &= \|S_{L(p)}(p) - S_{L(q)}(p)\| + \|S_{L(p)}(q) - S_{L(q)}(q)\| \\ Y &= \|\nabla S_{L(p)}(p) - \nabla S_{L(q)}(p)\| + \|\nabla S_{L(p)}(q) - \nabla S_{L(q)}(q)\| \\ Z &= E_{L(p)}(p,q) + E_{L(q)}(p,q)) \end{aligned}$$

and $\nabla S_z(p)$ is a 6-component color gradient (in $R$, $G$, and $B$) of image $z$ at pixel $p$, and $E_z(p,q)$ is the scalar edge potential between two neighboring pixels $p$ and $q$ of image $z$, computed using a Sobel filter.

- Intuitively, this is equvalent to finding a best **seam** between the neighboring pixels $p,q$ if $L(p) \neq L(q)$

# Minimize the penalty? Graph Cut Optimization

- To minimize this penalty, we use Graph Cut.

$$C(L) = \sum_{p} C_d(p, L(p)) + \sum_{p,q} C_i(p, q, L(p), L(q)) \quad (1)$$

- Boykov et al. [2001] have developed graph-cut techniques to optimize pixel labeling problems

# Graph Cut

- Designed to solve labeling problem in MRF.

A standard form of the energy function is

$$E(f) = \sum_{p \in \mathcal{P}} D_p(f_p) + \sum_{p,q \in \mathcal{N}} V_{p,q}(f_p, f_q),$$

- $f$ is the label of a Markov Random Field. $D_p$ is the data term of pixel $p$. $V_{p,q}$ is the smoothness term.

# Graph Cut

- "Binary" problems can be solved exactly using this approach; problems where pixels can be labeled with more than two different labels cannot be solved exactly, but solutions produced are usually near the global optimum.

- Alpha-expansion, alpha-beta swap

# Graph Cut

- The interested readers can refer to these papers:
  - Kolmogorov V, Zabin R. What energy functions can be minimized via graph cuts?[J]. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2004, 26(2): 147-159.
  - Boykov Y, Veksler O, Zabih R. Fast approximate energy minimization via graph cuts[J]. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2001, 23(11): 1222-1239.

# Graph Cut Library

- Libraries implemented by the authors of the papers:
  - maxflow algorithm by Vladimir Kolmogorov: http://pub.ist.ac.at/~vnk/software.html
  - GCO by Yuri Boykov: http://vision.csd.uwo.ca/code/

# Gradient-domain fusion

- For many applications the source images are too dissimilar for a graph-cut alone to result in visually seamless composites
- In these cases, it’s useful to fuse in the gradient domain.

# Gradient-domain fusion

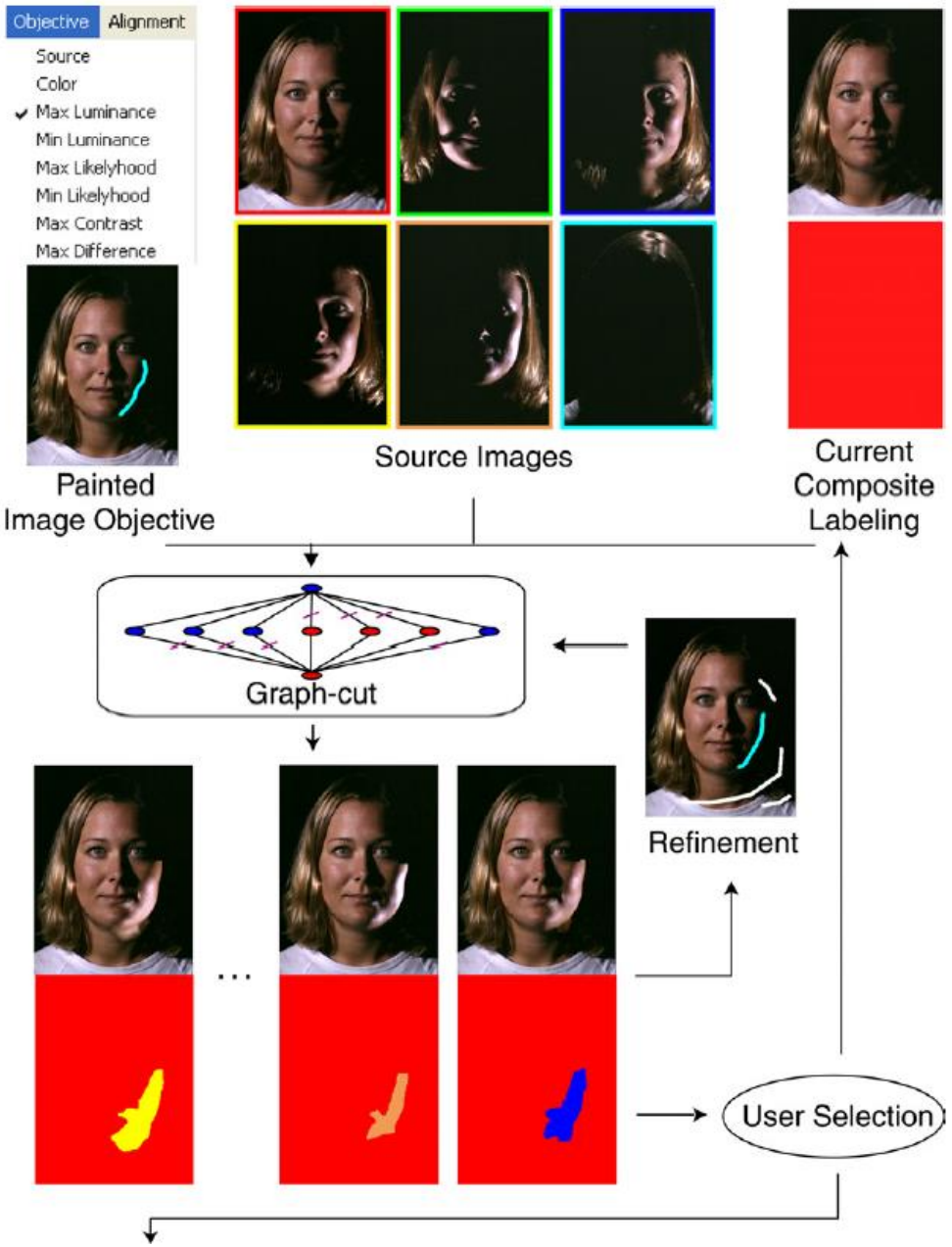
For a single color channel, we seek to solve for the pixel values $I(x,y)$. We re-order these values into a vector $v$, but, for convenience here, we still refer to each element $v_{x.y}$ based on its corresponding $(x,y)$ pixel coordinates. An input gradient $\nabla I(x,y)$ specifies two linear equations, each involving two variables:

$$v_{x+1,y} - v_{x,y} = \nabla I_x(x,y) \quad (2)$$

$$v_{x,y+1} - v_{x,y} = \nabla I_y(x,y) \quad (3)$$

- This is in some way similar to poisson image composition.

Iterative manner

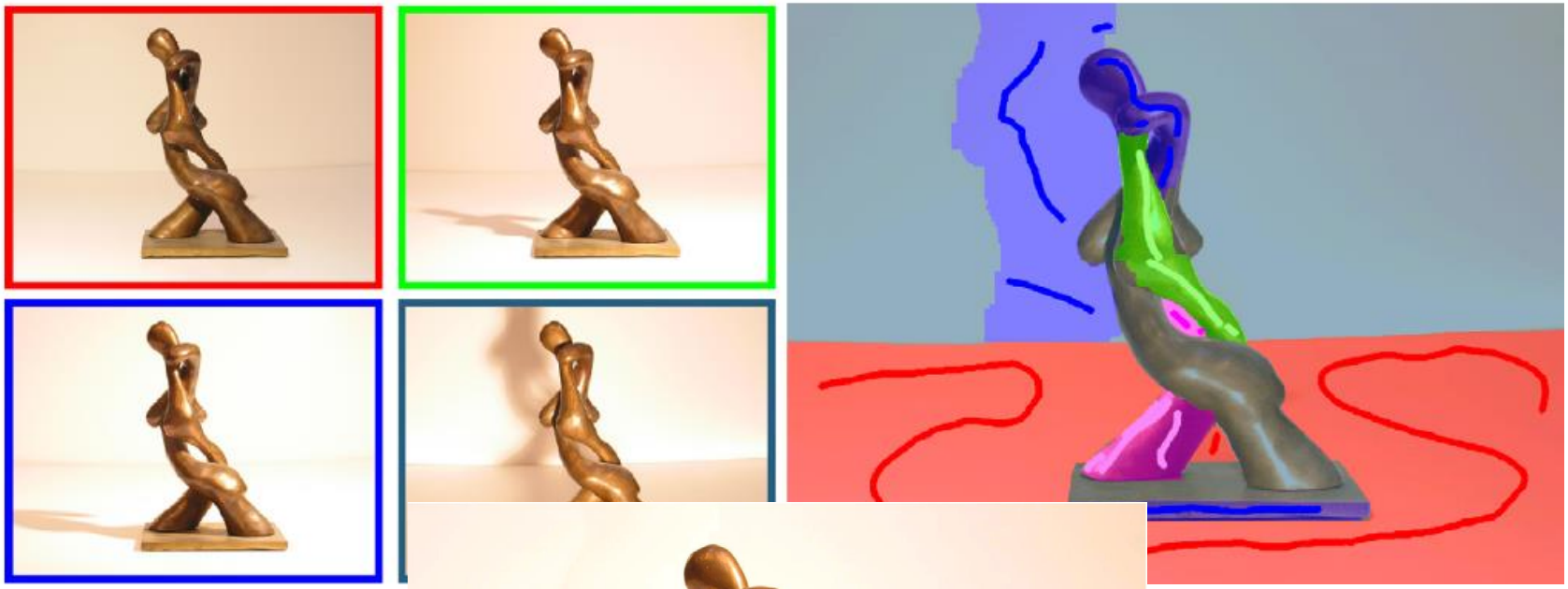


# Application besides Image Composition

- For other applications, the data penalty can be modified accordingly.
- Application includes:
  - ***Extended depth of field***
  - ***Relighting***
  - ***Stroboscopic visualization of movement***
  - ***Time-lapse mosaics***
  - ***Panoramic stitching***
  - ***Clean-plate production***

# 应用：扩展景深

# 应用： Relighting



# 应用： Relighting

# 应用：Stroboscopic visualization of movement

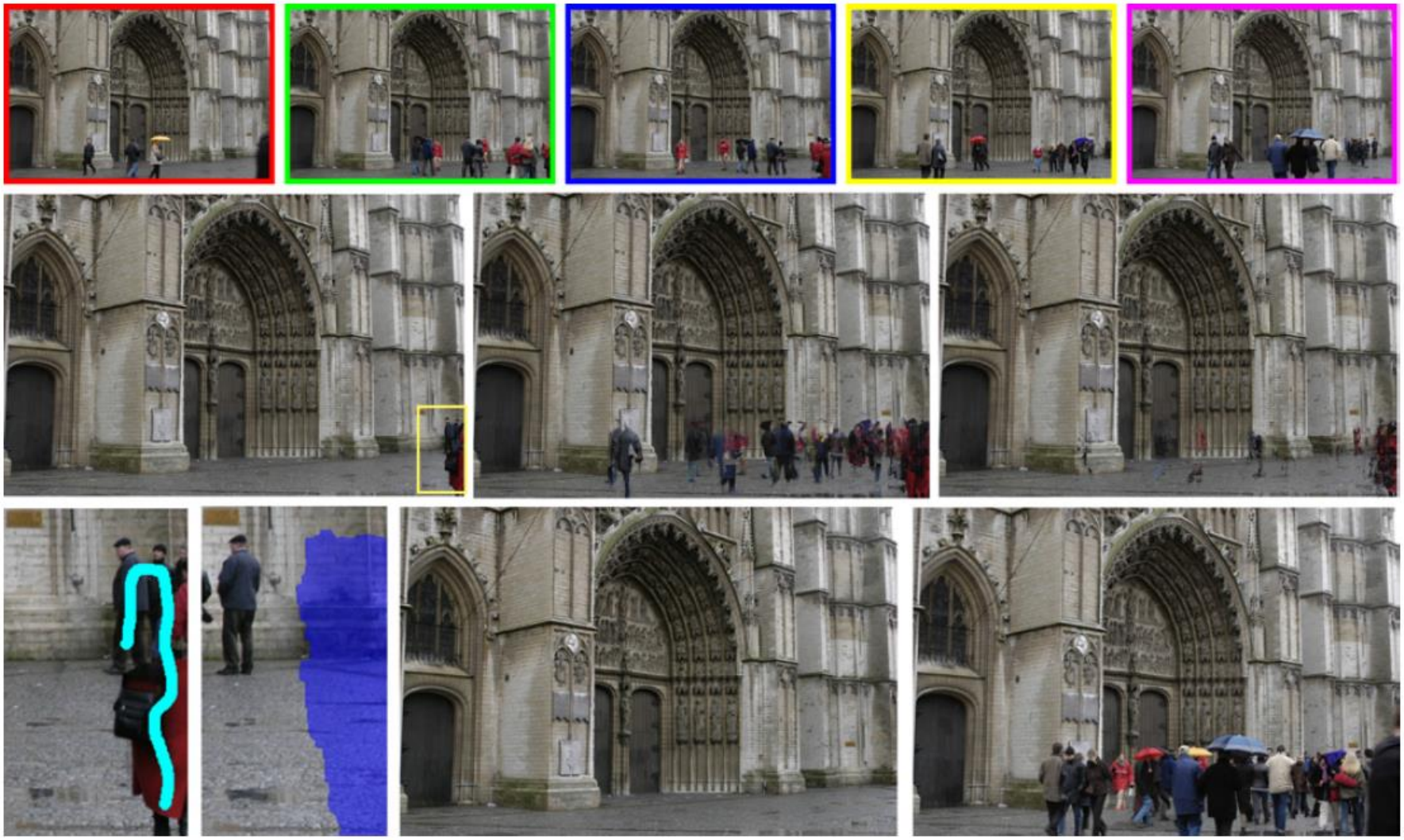
# 应用：Selective composites

# 应用：Selective composites

# 应用：Clean-plate production

**Figure 9** Three of a series of nine images of a scene that were captured by moving the camera to the left, right, up, and down in increments of a few feet. The images were registered manually to align the background mountains. A *minimum contrast* image objective was then used globally to remove the wires.

# 应用：Clean-plate production



# Comparison

- What are the pros and cons of the gradient domain fusion scheme and the graph cut scheme?

# 稀疏矩阵

# Sparse Matrix

- A sparse matrix is a matrix in which most of the elements are zero
- Example of sparse matrix

$$\begin{pmatrix} 10 & 20 & 0 & 0 & 0 & 0 \\ 0 & 30 & 0 & 40 & 0 & 0 \\ 0 & 0 & 50 & 60 & 70 & 0 \\ 0 & 0 & 0 & 0 & 0 & 80 \end{pmatrix}$$

# Sparse Matrix

- Many of the above problems require solving a large sparse linear system( for example, the poisson equation in the image setting is a large sparse system)

# How to represent Sparse Matrix in Computer?

- List of lists (LIL)
  - LIL stores one list per row, with each entry containing the column index and the value. Typically, these entries are kept sorted by column index for faster lookup.
- Coordinate list (COO)
  - stores a list of (row, column, value) tuples.

# How to represent Sparse Matrix in Computer?

- Compressed Row Storage (CRS or CSR)
  - three vectors: one for floating point numbers (val) and the other two for integers (col_ind, row_ptr).
  - As an example:

$$A = \begin{bmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{bmatrix}.$$

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| val | 10 | −2 | 3 | 9 | 3 | 7 | 8 | 7 | 3 ··· | 9 | 13 | 4 | 2 | −1 | | |
| col_ind | 1 | 5 | 1 | 2 | 6 | 2 | 3 | 4 | 1 ··· | 5 | 6 | 2 | 5 | 6 | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| row_ptr | 1 | 3 | 6 | 9 | 13 | 17 | 20 |

# How to represent Sparse Matrix in Computer?

- Other representations include:
  - Compressed column Storage
  - Yale
  - Please refer to Wikipedia for more detailed description.

# Libraries that supports Sparse Matrix

- Eigen Library
- Taucs

# 线性方程组的求解

# Solving linear system

- Direct Methods
  - Gaussian Elimination
  - Matrix factorization, LU
  - LDLT, Cholesky
- Iterative Methods
  - Gauss-Seidel
  - Jacobi
  - Conjugate Gradient

# Gauss-Seidel

- The Gauss–Seidel method is an iterative technique for solving a square system of n linear equations with unknown x:

$$Ax = b$$

# Gauss-Seidel

- It's defined by the following iteration:

$$L_* x^{(k+1)} = b - U x^{(k)}$$

where the matrix A is decomposed into a lower triangular component $L_*$, and a strictly upper triangular component $U$:

$$A = L_* + U$$

# Gauss-Seidel

- In more detail:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

- The decomposition of A into $L_*$ and $U$:

$A = L_* + U$, where

$$L_* = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \quad U = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}.$$

# Gauss-Seidel

- The iteration $L_* x^{(k+1)} = b - U x^{(k)}$ can be rewritten as:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left( b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)} \right), \quad i, j = 1, 2, \ldots, n.$$

# Gauss-Seidel

- Convergence:

The procedure is known to converge if either:

- $A$ is symmetric positive-definite, or
- $A$ is strictly or irreducibly diagonally dominant.

The Gauss–Seidel method sometimes converges even if these conditions are not satisfied

# Gauss-Seidel Algorithm

Inputs: $A$, $b$
Output: $\phi$

Choose an initial guess $\phi$ to the solution
**repeat** until convergence
    **for** $i$ **from** 1 **until** $n$ **do**
        $\sigma \leftarrow 0$
        **for** $j$ **from** 1 **until** $n$ **do**
            **if** $j \neq i$ **then**
                $\sigma \leftarrow \sigma + a_{ij}\phi_j$
            **end if**
        **end** ($j$-loop)
        $\phi_i \leftarrow \frac{1}{a_{ii}}(b_i - \sigma)$
    **end** ($i$-loop)
    check if convergence is reached
**end** (repeat)

# Examples

- Suppose:

$$A = \begin{bmatrix} 16 & 3 \\ 7 & -11 \end{bmatrix} \text{ and } b = \begin{bmatrix} 11 \\ 13 \end{bmatrix}$$

- We want to use the equation $L_* x^{(k+1)} = b - U x^{(k)}$ in the form

$$x^{(k+1)} = T x^{(k)} + C,$$

where

$$T = {L_*}^{-1} U, C = {L_*}^{-1} b$$

# Examples

- By decomposing A, we have:

$$L_* = \begin{bmatrix} 16 & 0 \\ 7 & -11 \end{bmatrix} \text{ and } U = \begin{bmatrix} 0 & 3 \\ 0 & 0 \end{bmatrix}$$

- The inverse of $L_*$ is:

$$L_*^{-1} = \begin{bmatrix} 16 & 0 \\ 7 & -11 \end{bmatrix}^{-1} = \begin{bmatrix} 0.0625 & 0.0000 \\ 0.0398 & -0.0909 \end{bmatrix}.$$

# Examples

- Now we have:

$$T = -\begin{bmatrix} 0.0625 & 0.0000 \\ 0.0398 & -0.0909 \end{bmatrix} \times \begin{bmatrix} 0 & 3 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0.000 & -0.1875 \\ 0.000 & -0.1193 \end{bmatrix},$$

$$C = \begin{bmatrix} 0.0625 & 0.0000 \\ 0.0398 & -0.0909 \end{bmatrix} \times \begin{bmatrix} 11 \\ 13 \end{bmatrix} = \begin{bmatrix} 0.6875 \\ -0.7443 \end{bmatrix}.$$

- Then we choose a initial value $x^{(0)}$:

$$x^{(0)} = \begin{bmatrix} 1.0 \\ 1.0 \end{bmatrix}$$

# Examples

- $x^{(k+1)} = Tx^{(k)} + C,$

$$x^{(1)} = \begin{bmatrix} 0.000 & -0.1875 \\ 0.000 & -0.1193 \end{bmatrix} \times \begin{bmatrix} 1.0 \\ 1.0 \end{bmatrix} + \begin{bmatrix} 0.6875 \\ -0.7443 \end{bmatrix} = \begin{bmatrix} 0.5000 \\ -0.8636 \end{bmatrix}.$$

$$x^{(2)} = \begin{bmatrix} 0.000 & -0.1875 \\ 0.000 & -0.1193 \end{bmatrix} \times \begin{bmatrix} 0.5000 \\ -0.8636 \end{bmatrix} + \begin{bmatrix} 0.6875 \\ -0.7443 \end{bmatrix} = \begin{bmatrix} 0.8494 \\ -0.6413 \end{bmatrix}.$$

$$x^{(3)} = \begin{bmatrix} 0.000 & -0.1875 \\ 0.000 & -0.1193 \end{bmatrix} \times \begin{bmatrix} 0.8494 \\ -0.6413 \end{bmatrix} + \begin{bmatrix} 0.6875 \\ -0.7443 \end{bmatrix} = \begin{bmatrix} 0.8077 \\ -0.6678 \end{bmatrix}.$$

$$x^{(4)} = \begin{bmatrix} 0.000 & -0.1875 \\ 0.000 & -0.1193 \end{bmatrix} \times \begin{bmatrix} 0.8077 \\ -0.6678 \end{bmatrix} + \begin{bmatrix} 0.6875 \\ -0.7443 \end{bmatrix} = \begin{bmatrix} 0.8127 \\ -0.6646 \end{bmatrix}.$$

$$x^{(5)} = \begin{bmatrix} 0.000 & -0.1875 \\ 0.000 & -0.1193 \end{bmatrix} \times \begin{bmatrix} 0.8127 \\ -0.6646 \end{bmatrix} + \begin{bmatrix} 0.6875 \\ -0.7443 \end{bmatrix} = \begin{bmatrix} 0.8121 \\ -0.6650 \end{bmatrix}.$$

$$x^{(6)} = \begin{bmatrix} 0.000 & -0.1875 \\ 0.000 & -0.1193 \end{bmatrix} \times \begin{bmatrix} 0.8121 \\ -0.6650 \end{bmatrix} + \begin{bmatrix} 0.6875 \\ -0.7443 \end{bmatrix} = \begin{bmatrix} 0.8122 \\ -0.6650 \end{bmatrix}.$$

$$x^{(7)} = \begin{bmatrix} 0.000 & -0.1875 \\ 0.000 & -0.1193 \end{bmatrix} \times \begin{bmatrix} 0.8122 \\ -0.6650 \end{bmatrix} + \begin{bmatrix} 0.6875 \\ -0.7443 \end{bmatrix} = \begin{bmatrix} 0.8122 \\ -0.6650 \end{bmatrix}.$$

# Jacobi

- Given a square system of *n* linear equations $Ax = b$

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

- $A$ can be decomposed into a diagonal component $D$, and the remainder $R$:

# Jacobi

- $$A = D + R$$

$$D = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix} \text{ and } R = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ a_{21} & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{bmatrix}$$

- The solution is then obtained iteratively via:

$$x^{(k+1)} = D^{-1}(b - Rx^{(k)})$$

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left( b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \ldots, n.$$

# Jacobi Algorithm

Choose an initial guess $x^{(0)}$ to the solution

$k = 0$

while convergence not reached do

  for i := 1 step until n do

    $\sigma = 0$

    for j := 1 step until n do

      if j $\neq$ i then

        $\sigma = \sigma + a_{ij} x_j^{(k)}$

      end if

    end (j-loop)

    $x_i^{(k+1)} = \frac{(b_i - \sigma)}{a_{ii}}$

  end (i-loop)

  check if convergence is reached

  $k = k + 1$

loop (while convergence condition not reached)

# Jacobi

- Convergence:
  - when the ***spectral radius*** of the iteration matrix is less than 1:

$$\rho(D^{-1}R) < 1$$

  - The method is guaranteed to converge if the matrix $A$ is strictly or irreducibly diagonally dominant.

# Examples

- Suppose:

$$A = \begin{bmatrix} 2 & 1 \\ 5 & 7 \end{bmatrix}, \; b = \begin{bmatrix} 11 \\ 13 \end{bmatrix} \quad \text{and} \quad x^{(0)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

- we rewrite the equation in a more convenient form:

$$x^{(k+1)} = D^{-1}\left(b - Rx^{(k)}\right) = Tx^{(k)} + C$$

# Examples

- We can easily see that:

$$T = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/7 \end{bmatrix} \left\{ \begin{bmatrix} 0 & 0 \\ -5 & 0 \end{bmatrix} + \begin{bmatrix} 0 & -1 \\ 0 & 0 \end{bmatrix} \right\} = \begin{bmatrix} 0 & -1/2 \\ -5/7 & 0 \end{bmatrix}.$$

$$C = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/7 \end{bmatrix} \begin{bmatrix} 11 \\ 13 \end{bmatrix} = \begin{bmatrix} 11/2 \\ 13/7 \end{bmatrix}.$$

- We have:

$$x^{(1)} = \begin{bmatrix} 0 & -1/2 \\ -5/7 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 11/2 \\ 13/7 \end{bmatrix} = \begin{bmatrix} 5.0 \\ 8/7 \end{bmatrix} \approx \begin{bmatrix} 5 \\ 1.143 \end{bmatrix}.$$

# Examples

$$x^{(2)} = \begin{bmatrix} 0 & -1/2 \\ -5/7 & 0 \end{bmatrix} \begin{bmatrix} 5.0 \\ 8/7 \end{bmatrix} + \begin{bmatrix} 11/2 \\ 13/7 \end{bmatrix} = \begin{bmatrix} 69/14 \\ -12/7 \end{bmatrix} \approx \begin{bmatrix} 4.929 \\ -1.714 \end{bmatrix}$$

- This process is repeated until convergence( the 25$^{th}$ iteration):

$$x = \begin{bmatrix} 7.111 \\ -3.222 \end{bmatrix}$$

# Conjugate Gradient

- The conjugate gradient method is an algorithm for the numerical solution of particular systems of linear equations whose matrix is **symmetric** and **positive-definite.**

# Conjugate Gradient

- Two vectors $u, v$ are said to be conjugate (with respect to $A$) if:

$$u^T A v = 0$$

- Suppose $P = \{p_k\}$ is a basis of $R^n$. Within $P$, we can expand the solution $x_*$ of $Ax = b$:

$$x_* = \sum_i^n \alpha_i p_i$$

# Conjugate Gradient

- And we can see that:

$$b = Ax_* = \sum_i^n \alpha_i A p_i$$

- For any $p_k \in P$, we have:

$$\mathbf{p}_k^{\mathrm{T}}\mathbf{b} = \mathbf{p}_k^{\mathrm{T}}\mathbf{A}\mathbf{x}_* = \sum_{i=1}^{n} \alpha_i \mathbf{p}_k^{\mathrm{T}}\mathbf{A}\mathbf{p}_i = \alpha_k \mathbf{p}_k^{\mathrm{T}}\mathbf{A}\mathbf{p}_k.$$

(because $\forall i \neq k, p_i, p_k$ are mutually conjugate)

$$\alpha_k = \frac{\mathbf{p}_k^{\mathrm{T}}\mathbf{b}}{\mathbf{p}_k^{\mathrm{T}}\mathbf{A}\mathbf{p}_k} = \frac{\langle \mathbf{p}_k, \mathbf{b} \rangle}{\langle \mathbf{p}_k, \mathbf{p}_k \rangle_{\mathbf{A}}} = \frac{\langle \mathbf{p}_k, \mathbf{b} \rangle}{\|\mathbf{p}_k\|_{\mathbf{A}}^2}.$$

# Conjugate Gradient as an iterative method

- If we choose the conjugate vectors $\boldsymbol{p}_k$ carefully, then we may not need all of them to obtain a good approximation to the solution $x_*$
- So, we want to regard the conjugate gradient method as an **iterative** method
- in each iteration we need a metric to tell us whether we are closer to the solution $x_*$

# Conjugate Gradient

- The metric is

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} - \mathbf{x}^{\mathrm{T}}\mathbf{b}, \quad \mathbf{x} \in \mathbf{R}^n.$$

- The residue $r_k$ at step $k$, which is also the negative of the gradient of $f(x_k)$, is:

$$r_k = b - Ax_k$$

- So the gradient descent method would be to move in the direction of $r_k$

# Conjugate Gradient

- But we also insist that the direction $p_k$ be conjugate to each other.
- This gives the following expression:

$$p_k = r_k - \sum_{i<k} \frac{p_i^T A r_k}{p_i^T A p_i} p_i$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

with

$$\alpha_k = \frac{\mathbf{p}_k^{\mathrm{T}} \mathbf{b}}{\mathbf{p}_k^{\mathrm{T}} \mathbf{A} \mathbf{p}_k} = \frac{\mathbf{p}_k^{\mathrm{T}} (\mathbf{r}_{k-1} + \mathbf{A}\mathbf{x}_{k-1})}{\mathbf{p}_k^{\mathrm{T}} \mathbf{A} \mathbf{p}_k} = \frac{\mathbf{p}_k^{\mathrm{T}} \mathbf{r}_{k-1}}{\mathbf{p}_k^{\mathrm{T}} \mathbf{A} \mathbf{p}_k},$$

# Conjugate Gradient Algorithm

$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$

$\mathbf{p}_0 := \mathbf{r}_0$

$k := 0$

repeat

$\alpha_k := \dfrac{\mathbf{r}_k^{\mathrm{T}}\mathbf{r}_k}{\mathbf{p}_k^{\mathrm{T}}\mathbf{A}\mathbf{p}_k}$

$\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k\mathbf{p}_k$

$\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k\mathbf{A}\mathbf{p}_k$

if $r_{k+1}$ is sufficiently small then exit loop

$\beta_k := \dfrac{\mathbf{r}_{k+1}^{\mathrm{T}}\mathbf{r}_{k+1}}{\mathbf{r}_k^{\mathrm{T}}\mathbf{r}_k}$

$\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k\mathbf{p}_k$

$k := k + 1$

end repeat

The result is $\mathbf{x}_{k+1}$

# Libraries/Software that can solve linear systems

- Eigen Library
- LAPACK
- Matlab

# Assignment

- Write a Sparse Matrix version of Gauss-Seidel solver.