Graph Cut Segmentation

Some material for these slides comes from
https://www.csd.uwo.ca/Courses/CS4487a/
http://www.robots.ox.ac.uk/~lubor/tutorial.html
http://www.csie.ntu.edu.tw/~cyy/courses/vfx/07spring/lectures/handouts/lec13_GrabCut.ppt
Segmentation Goals

• We want to group pixels such that
  – Pixels in a region have similar appearance (statistics)
  – Boundaries between regions are short and located at visible discontinuities

• Let’s just consider segmentation into two regions (background and foreground)
Markov Random Field

- We treat the image as a grid (or graph) with links between neighboring pixels
- Let $y$ be the measurements (e.g., intensities) at each pixel
  \[ y = [d(0, 0) \ldots d(m - 1, n - 1)] \]
- Let $x$ be the unknowns (i.e., the label for each pixel; foreground or background)
  \[ x = [f(0, 0) \ldots f(m - 1, n - 1)]. \]
- This is called a Markov Random Field
- We want to find $x$ such that $p(x|y)$ is a maximum
Energy Minimization Problem

- We want to find $x$ such that $p(x|y)$ is a maximum.
- From Bayes’ rule,
  \[ p(x|y) = \frac{p(y|x)p(x)}{p(y)} \]
- Taking the negative log of both sides, we get
  \[ -\log p(x|y) = -\log p(y|x) - \log p(x) + C. \]
- Maximizing $p(x|y)$ is the same as minimizing $-\log p(x|y)$, which can be thought of as an energy
  \[ E(x, y) = E_d(x, y) + E_p(x) \]
  - Data energy
  - Prior energy
Graph cuts for optimal boundary detection

Minimum cost cut can be computed in polynomial time
(max-flow/min-cut algorithms)
Graph Cuts Basics

Goal: divide the graph into two parts separating red and blue nodes

A graph with two terminals $S$ and $T$

- Cut cost is a sum of severed edge weights
- **Minimum cost $s$-$t$ cut can be found in polynomial time**
Min-Cut Problem

Task:

Minimize the cost of the cut

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

\[ \min_{S,T} \sum_{i \in S, j \in T} c_{ij} \]
Min-Cut Problem

\[
\min_{S,T} \sum_{i \in S, j \in T} c_{ij} \quad s.t. \quad s \in S, \quad t \in T
\]
Min-Cut Problem

\[
\min_{S,T} \sum_{i \in S, j \in T} c_{ij}
\]

s.t. \quad s \in S, \quad t \in T

source set \quad \text{sink set}

edge costs

\[
\text{cost} = 18
\]
Min-Cut Problem

\[ \min_{S,T} \sum_{i \in S, j \in T} c_{ij} \]

s.t. \( s \in S, \; t \in T \)

Source set \( S \) and Sink set \( T \). Costs on the edges.

Cost = 25
Min-Cut Problem

\[
\begin{aligned}
\text{source set} & \quad \text{sink set} \\
S & \quad T
\end{aligned}
\]

\[
\text{cost} = 23
\]

\[
\begin{aligned}
\min_{S,T} & \quad \sum_{i \in S, j \in T} c_{ij} \\
\text{s.t.} & \quad s \in S, \quad t \in T
\end{aligned}
\]
Min-Cut Problem

\[
\begin{align*}
\min_x & \quad \sum_{(i,j) \in E} c_{ij}(1 - x_i)x_j \\
\text{s.t.} & \quad x_s = 0, \quad x_t = 1 \\
x_i = 0 & \implies x_i \in S \quad x_i = 1 \implies x_i \in T
\end{align*}
\]
Min-Cut Problem

\[
\min_x \sum_{(i,j) \in E} c_{ij} (1 - x_i)x_j
\]

\[
s.t. \quad x_s = 0, \quad x_t = 1
\]

transformable into Linear program (LP)

\[
\min_{x,d} \quad c_{ij}d_{ij}
\]

\[
s.t. \quad d_{ij} \geq x_j - x_i \quad d_{ij} \geq 0
\]

\[
x_s = 0 \quad x_t = 1
\]
s/t min cut algorithms are widely studied (combinatorial optimization)

- Augmenting paths [Ford & Fulkerson, 1962]
- Push-relabel [Goldberg-Tarjan, 1986]
“Augmenting Paths”

- Find a path from S to T along non-saturated edges
- Increase flow along this path until some edge saturates

A graph with two terminals
“Augmenting Paths”

- Find a path from S to T along non-saturated edges
  - Increase flow along this path until some edge saturates
- Find next path...
- Increase flow…

A graph with two terminals

“source”

“sink”
“Augmenting Paths”

- Find a path from $S$ to $T$ along non-saturated edges
- Increase flow along this path until some edge saturates

Iterate until all paths from $S$ to $T$ have at least one saturated edge
Graph cut is an old standard problem with tons of applications outside vision

From Harris & Ross [1955]

Soviet rail system
Graph cut is an old standard problem with tons of applications outside vision.

From Harris & Ross [1955]

Soviet rail system
3D Reconstruction
Grab Cut

- Iteratively runs graph cut
- At each iteration, it re-estimates the region statistics (modeled as a mixtures of Gaussians in color space)

Grab Cut

• Needs minimal user input, such as a single bounding box
  – Background color model is initialized from pixels outside the box
  – Foreground color model is initialized from interior pixels, but quickly converges to a better estimate

• The user can also place additional strokes to refine the segmentation as the solution progresses
Grab Cut

Result

Energy after each Iteration

Guaranteed to converge
Gaussian Mixture Model (typically 5-8 components)
Coherence Model

An object is a coherent set of pixels:

Learn color and spatial layout jointly
Moderately straightforward examples
Background Subtraction

\[ I = I_{\text{obj}} - I_{\text{bkg}} \]

Threshold intensities \( S = \{ p : I_p > T \} \)
Background Subtraction

\[ I = I_{obj} - I_{bkg} \]

Thresholding

\[ S = \{ p : I_p > T \} \]

Optimal cut

Threshold intensities
Difficult Examples

Camouflage & Low Contrast

Initial Rectangle

Initial Result

Fine structure
Figure 5: **User editing.** After the initial user interaction and segmentation (top row), further user edits (fig. 3) are necessary. Marking roughly with a foreground brush (white) and a background brush (red) is sufficient to obtain the desired result (bottom row).
Graphcuts in multiple images

- If you have multiple images of a 3D object from different viewpoints, you can use this to improve segmentation.
- The segmentation must be consistent across all views.

Approach

• Estimate pose of each view
• Assume that the object is completely visible in all views
• Find the “common view volume” (CVV) by intersecting all view frustrums
• Project the CVV onto each image - assume outside area is background
• Segment images into superpixels
• Use graph cuts to label superpixels as foreground or background
• Enforce consistency across views
Algorithm 1: The iterative segmentation algorithm.

Input
- A calibrated set of \( M \) images \( I_1 \ldots I_M \)

Initialisation
- Obtain bounding volume from visibility

\textbf{foreach} image \( I_m, \ m = 1 \ldots M \) \textbf{do}
  - Group pixels into superpixels \( \{s_i\} \)
  - Extract fixation point
\textbf{end}
- Learn background colour model from outside bounding box
- Learn object colour model from fixation points
- Generate the edge matrix \( W \)

Main Loop
\textbf{while} visual hull not converged \textbf{do}
  \textbf{foreach} image \( I_m, \ m = 1 \ldots M \) \textbf{do}
    - Evaluate object likelihood
  \textbf{end}
  - Perform graph-cut to label superpixels
  - Enforce silhouette consistency
  - Update object colour model from new silhouettes
\textbf{end}

Output
- The converged object silhouettes and visual hull
Figure 2: Illustration of the construction of the edge matrix $W$. (a) Each of the initial images $I_m$ is over-segmented to produce a superpixel representation $\{s_i\}$. (b) Every superpixel $s_i$ is projected into a set of neighbouring images using epipolar geometry. (c) Each of the neighbouring images $I_\mu \in N(I_m)$ is selected in turn. (d) The set of superpixels $\{s_j\}$ that lie along the corresponding epipolar line is found. (e) The depth and (f) colour consistency are found for each $s_j$ and used to perform the soft stereo depth binning of Equation (8).
Results
(a) Images of a vase (4 of 24)

(i) Our result

(j) Our result shown as a visual hull
OpenCV grabCut

```cpp
void cv::grabCut ( InputArray img, 
    InputOutputArray mask, 
    Rect rect, 
    InputOutputArray bgdModel, 
    InputOutputArray fgdModel, 
    int iterCount, 
    int mode = GC_EVAL )
```

Runs the GrabCut algorithm.

The function implements the GrabCut image segmentation algorithm.

**Parameters**

- **img**: Input 8-bit 3-channel image.
- **mask**: Input/output 8-bit single-channel mask. The mask is initialized by the function when mode is set to GC_INIT_WITH_RECT. Its elements may have one of the `cv::GrabCutClasses`.
- **rect**: ROI containing a segmented object. The pixels outside of the ROI are marked as "obvious background". The parameter is only used when mode==GC_INIT_WITH_RECT.
- **bgdModel**: Temporary array for the background model. Do not modify it while you are processing the same image.
- **fgdModel**: Temporary arrays for the foreground model. Do not modify it while you are processing the same image.
- **iterCount**: Number of iterations the algorithm should make before returning the result. Note that the result can be refined with further calls with mode==GC_INIT_WITH_MASK or mode==GC_EVAL.
- **mode**: Operation mode that could be one of the `cv::GrabCutModes`.

**Examples:**

- `grabcut.cpp`
Demo Program

// This program was adapted from the example tutorial on the OpenCV distribution.
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
using namespace std;
using namespace cv;

cv::String filename = "C:/OpenCV/sources/opencv/samples/data/lena.jpg";

static void help()
{
    cout << "This program demonstrates GrabCut segmentation -- select an object in a region\n"    "and then grabcut will attempt to segment it out.\n"    "Select a rectangular area around the object you want to segment\n"    "Hot keys: \n"    "tESC - quit the program\n"    "tR - restore the original image\n"    "tN - next iteration\n"    "\n"    "tLeft mouse button - set rectangle\n"    "\n"    "tCTRL+left mouse button - set GC_BGD pixels\n"    "tSHIFT+left mouse button - set GC_FGD pixels\n"    "\n"    "tCTRL+right mouse button - set GC_PR_BGD pixels\n"    "tSHIFT+right mouse button - set GC_PR_FGD pixels" << endl;
}

doGrabCut.cpp