```
        MATLAB File Help: GraphCut
        View code for GraphCut
        Default Topics


  GraphCut
      Performing Graph Cut energy minimization operations on a 2D grid.

Usage:
    [gch ...] = GraphCut(mode, ...);


Inputs:
- mode: a string specifying mode of operation. See details below.

Output:
- gch: A handle to the constructed graph. Handle this handle with care
       and don't forget to close it in the end!

Possible modes:
- 'open': Create a new graph object
        [gch] = GraphCut('open', DataCost, SmoothnessCost);
        [gch] = GraphCut('open', DataCost, SmoothnessCost, vC, hC);
        [gch] = GraphCut('open', DataCost, SmoothnessCost, SparseSmoothness);

    Inputs:
        - DataCost a height by width by num_labels matrix where
          Dc(r,c,l) equals the cost for assigning label l to pixel at (r,c)
          Note that the graph dimensions, and the number of labels are deduced
          form the size of the DataCost matrix.
          When using SparseSmoothness Dc is of (L)x(P) where L is the
          number of labels and P is the number of nodes/pixels in the
          graph.
        - SmoothnessCost a #labels by #labels matrix where Sc(l1, l2)
          is the cost of assigning neighboring pixels with label1 and
          label2. This cost is spatialy invariant
        - vC, hC:optional arrays defining spatialy varying smoothness cost.
                    Single precission arrays of size width*height.
                    The smoothness cost is computed using:
                    V_pq(l1, l2) = V(l1, l2) * w_pq
                    where V is the SmoothnessCost matrix
                    w_pq is spatialy varying parameter:
                    if p=(r,c) and q=(r+1,c) then w_pq = vCue(r,c)
                    if p=(r,c) and q=(r,c+1) then w_pq = hCue(r,c)
                    (therefore in practice the last column of vC and
                    the last row of vC are not used).
        - SparseSmoothness: a sparse matrix defining both the graph
            structure (might be other than grid) and the spatialy varying
            smoothness term. Must be real positive sparse matrix of size
            num_pixels by num_pixels, each non zero entry (i,j) defines a link
            between pixels i and j with w_pq = SparseSmoothness(i,j).

- 'set': Set labels
        [gch] = GraphCut('set', gch, labels)

    Inputs:
        - labels: a width by height array containing a label per pixel.
          Array should be the same size of the grid with values
          [0..num_labels].


- 'get': Get current labeling
        [gch labels] = GraphCut('get', gch)

    Outputs:
        - labels: a height by width array, containing a label per pixel.
          note that labels values are in range [0..num_labels-1].
```

```
- 'energy': Get current values of energy terms
        [gch se de] = GraphCut('energy', gch)
        [gch e] = GraphCut('energy', gch)

    Outputs:
        - se: Smoothness energy term.
        - de: Data energy term.
        - e = se + de


- 'expand': Perform labels expansion
        [gch labels] = GraphCut('expand', gch)
        [gch labels] = GraphCut('expand', gch, iter)
        [gch labels] = GraphCut('expand', gch, [], label)
        [gch labels] = GraphCut('expand', gch, [], label, indices)

    When no inputs are provided, GraphCut performs expansion steps
    until it converges.

    Inputs:
        - iter: a double scalar, the maximum number of expand
                 iterations to perform.
        - label: scalar denoting the label for which to perfom
                  expand step (labels are [0..num_labels-1]).
        - indices: array of linear indices of pixels for which
                     expand step is computed.

    Outputs:
        - labels: a width*height array of type int32, containing a
           label per pixel. note that labels values must be is range
           [0..num_labels-1].


- 'swap': Perform alpha - beta swappings
        [gch labels] = GraphCut('swap', gch)
        [gch labels] = GraphCut('swap', gch, iter)
        [gch labels] = GraphCut('swap', gch, label1, label2)

    When no inputs are provided, GraphCut performs alpha - beta swaps steps
    until it converges.

    Inputs:
        - iter: a double scalar, the maximum number of swap
                  iterations to perform.
        - label1, label2: int32 scalars denoting two labels for swap
                                       step.

    Outputs:
        - labels: a width*height array of type int32, containing a
           label per pixel. note that labels values must be is range
           [0..num_labels-1].


- 'truncate': truncating (or not) violating expansion terms
              (see Rother etal. Digital Tapestry, CVPR2005)
    [gch truncate_flag] = GraphCut('truncate', gch, trancate_flag);

    When no truncate_flag is provided the function returns the current
    state of truncation

    Inputs:
        - trancate_flag: set truncate_flag to this state

    Outputs:
        - trancate_flag: current state (after modification if
                          applicable)

- 'close': Close the graph and release allocated resources.
    [gch] = GraphCut('close', gch);
```

```
  This wrapper for Matlab was written by Shai Bagon (shai.bagon@weizmann.ac.il).
  Department of Computer Science and Applied Mathmatics
  Wiezmann Institute of Science
  http://www.wisdom.weizmann.ac.il/

The core cpp application was written by Olga Veksler
(available from http://www.csd.uwo.ca/faculty/olga/code.html):

  [1] Efficient Approximate Energy Minimization via Graph Cuts
       Yuri Boykov, Olga Veksler, Ramin Zabih,
       IEEE transactions on PAMI, vol. 20, no. 12, p. 1222-1239, November
       2001.

  [2] What Energy Functions can be Minimized via Graph Cuts?
       Vladimir Kolmogorov and Ramin Zabih.
       IEEE Transactions on Pattern Analysis and Machine Intelligence
       (PAMI), vol. 26, no. 2,
       February 2004, pp. 147-159.

  [3] An Experimental Comparison of Min-Cut/Max-Flow Algorithms
       for Energy Minimization in Vision.
       Yuri Boykov and Vladimir Kolmogorov.
       In IEEE Transactions on Pattern Analysis and Machine Intelligence
       (PAMI),
       vol. 26, no. 9, September 2004, pp. 1124-1137.

  [4] Matlab Wrapper for Graph Cut.
       Shai Bagon.
       in www.wisdom.weizmann.ac.il/~bagon, December 2006.

  This software can be used only for research purposes, you should  cite ALL of
  the aforementioned papers in any resulting publication.
  If you wish to use this software (or the algorithms described in the
  aforementioned paper)
  for commercial purposes, you should be aware that there is a US patent:

      R. Zabih, Y. Boykov, O. Veksler,
      "System and method for fast approximate energy minimization via
      graph cuts ",
      United Stated Patent 6,744,923, June 1, 2004


  The Software is provided "as is", without warranty of any kind.
```