Lab Work: Disparity Map Estimation Using Graph Cuts

1. Problem statement

Given two rectified images:

- Compute a disparity map using a graph cut. See course on graph cuts for details.
- Display the resulting disparity map.
- Compare with the region-growing algorithm: precision, smoothness and computation time.

(You may want to add a slight blur to the result to extra smooth it a bit.)

2. Imagine++ library

Installation and documentation: see http://imagine.enpc.fr/~monasse/Imagine++/

2.1. Header

The header will need to access library services is as follows:

```
#include <Imagine/Images.h>
using namespace Imagine;
```

2.2. Useful types and functions

- small integer (typically for greyscale intensity) between 0 and 255: `byte`
- image class: `Image<byte>`, `Image<double>`
- image creation: `Image<double> I(w,h)`
- image access: `I(x,y)=0; return I(x+a,y+b);`
- image size: `I.height()`, `I.width()`
- sub-image creation: `I.getSubImage(x,y,w,h)`
- image enlargement by given factor (with interpolation): `enlarge(I,fact)`
- image blurring with Gaussian of given sigma: `blur(I,sigma)`
- representation of a scalar image (e.g., double) by a greyscale (e.g., byte): `bI=grey(dI)`
- reference to file located in source directory: `srcPath("file_in_src_dir.txt")`
- image loading: `load(I,srcPath("face00R.png"))`
- window opening: `openWindow(w,h)`
- image displaying in open window at given offsets: `display(I,x=0,y=0)`
- wait for mouse click in active window: `click()`

3. Graph cut library

A graph cut library is provided in the `maxflow` directory. The library documentation is available as comments in the file `maxflow/graph.h`. An example of the use of the library is described in file `maxflow/README.txt` and is available in file `exampleGC.cpp`. It can be compiled via `cmake`.

3.1. Header

The header will need to access library services is as follows:

```
#include "maxflow/graph.h"
```

3.2. Useful classes and functions

- graph class (with type of flow/capacities): `Graph<int,int,int>`
- graph creation: `Graph<int,int,int> G(nbNodeMax,nbEdgeMax)`
- graph node creation (excluding preallocated *s* & *t* nodes): `G.add_node(nbNodes)`
- terminal edge (t-link) creation: `G.add_tweights(nodeNum,sWeight,tWeight)`
- n-link creation: `G.add_edge(nodeNum1,nodeNum2,weight12,weight21)`
- max-flow computation: `minE=G.maxflow()`
- type of the source node: `Graph<int,int,int>::SOURCE`
- type of the sink node: `Graph<int,int,int>::SINK`
- type a node after cut: `G.what_segment(nodeNum)`

Note that all n-links are assumed bidirectional. To create mono-directional n-links, use "infinite" weights. For this, use capacities that are much larger that any flow, but do not use `INT_MAX` or `FLT_MAX` as it may overflow after an addition.

4. Provided material

4.1. Images

Two rectified images are provided. (But you can use others!) Their size is 512x512 but your code should be independent of that. You might want to clip these images a bit (i.e. remove a small margin) to mostly focus on pixels that are visible in both images.

4.2. Orders of magnitude for the provided images

- clipping margin: 20 to 30 pixels
- NCC neighborhood size: 3 pixel radius, i.e. 7x7 pixel patch
- disparity (not to over-dimension the graph): dmin = 10, dmax=55
- base for interaction potential: $\lambda = 2$
- base for weight of non-correlating pixels: $w_0 = 20$

4.3. Optimization

To prevent redundant computations when computing NCC, you may want to precompute and store the average image intensity in patch for each pixel of both images. (This may be stored into a "pseudo" image whose pixel intensity is that mean value.)

4.4. Template

A file named `stereoGCtemplate.cpp` contains a skeleton for loading the images, constructing the graph, compute its minimum cut (maximum flow), reading disparities from the minimum cut, constructing a 3D mesh from these disparities and displaying it. It can be compiled via `cmake`. However, the result will not make sense until you complete the code as appropriate (see comments "BEGIN CODE TO BE COMPLETED … END CODE TO BE COMPLETED").

5. Packaging your work

Please comment your results: play with parameters, change pictures, what happens? Submit both your code and report in a single, clean archive (no executable code).