MarsImg Manual

Miryam Chaabouni

June 26, 2014

1 Description

MarsImg is a program that uses a parallel implementation of David Lowe's SIFT feature detector designed to detect and match specific features from very large Mars HiRISE images. It is based on Changchang Wu's SiftGPU. The program comes as a project within the Visual Studio solution SiftGPU.

2 Requirements

SiftGPU and MarsImg were implemented with Microsoft Visual Studio 2010, and it is recommended to use this version to compile and build the program. MarsImg was based on SimpleSIFT project by SiftGPU author, and does not use the GUIs provided with the original program.

According to the description of SiftGPU, the program requires a GPU that has large memory and supports dynamic branching. However, MarsImg ran successfully on a basic AMD graphic card (ATI Mobility Radeon). Though SiftGPU provides a solution for NVidia graphic cards (SiftGPU_CUDA_Enabled), only the default GLSL solution was tested for MarsImg.

MarsImg uses Kakadu software to cut the original large JPEG2000 images into small blocks. A demo version can be downloaded for free and used for six months at the address :
`http://www.kakadusoftware.com/index.php?option=com_content&task=view&id=26&Itemid=22`.

It should be installed on your machine under the path : **"C:/Program Files (x86)/Kakadu/kdu_expand.exe"**. Kakadu can also be used in Mac and Linux, but the program was only tested on Windows.

3 Input Images and parameters

All the program's parameters should be indicated as global variables at the beginning of the file MarsImg.cpp.
MarsImg.cpp compares two JPEG2000 images (extension .JP2) that should be placed in the folders Image1 and Image2 in the data folder. The paths to both images must be indicated at the beginning of MarsImg.cpp along with their dimensions. These can be easily found using Kakadu's application kdu_jp2info.
Both input images will be divided into smaller sub-images. The choice of the grid size is up to the user, but notice that SiftGPU will automatically perform down-sampling when the size of the sub-images are larger than ≈500×500 pixels.
You also need to specify two thresholds :

Area threshold Minimum number of matches between two sub-images to assess their similarity. Between 100 and 4000 depending on the image and the grid size.

SCF threshold Minimum number of matches between a sub-image and reference Mars pits images to asses that the key point corresponds to a Swiss-cheese feature. Between 3 and 30 depending on the density of Mars pits.

Example :

```
//*******************Global variables*********************************
//Specify images paths and dimensions
std::string inimage1 = " -i ../data/Image1/upleft.JP2 ";
std::string inimage2 = " -i ../data/Image2/shifted.JP2 ";
int width1 = 5012, height1 = 5000, width2 = 5012, height2 = 5000;
float nRows = 5., nCols = 5.; //float to avoid integer division
```

```
//specify thresholds for matching points
int thrSCF = 25;
int thrAr = 1500;

//*******************************************************************
```

4 Output

The program creates a text file "matching_points.txt" in the data folder. It contains four columns of numbers corresponding to the coordinates of the matching points. For example, if $P_1(x_1, y_1)$ and $P_2 = (x_2, y_2)$ are two matching point coming respectively from Image1 and Image2, the program will write in the ouput file :
$x_1 \quad y_1 \quad x_2 \quad y_2$
The sub-images created by the program will be saved into the corresponding folders data/Image1 and data/Image2.
To visualize the matching points using Matlab, an example is provided in the folder data/Matlab_MatchPlot.