GPU - accelerated High Quality Semi-global Stereo Matching

Hermann Fürntratt
JOANNEUM RESEARCH, Austria

Hannes Fassold
JOANNEUM RESEARCH, Austria

Jakub Rosner
Silesian University of Technologie, Poland

Andreas Wimmer
JOANNEUM RESEARCH, Austria

Roland Perko
JOANNEUM RESEARCH, Austria

JOANNEUM RESEARCH
Forschungsgesellschaft mbH

DIGITAL
Institute for Information and
Communication Technologies

Steyrergasse 17
8010 Graz, Austria

Tel. +43 316 876-1199
Fax +43 316 876-1191

hermann.fuerntratt@joanneum.at

digital@joanneum.at
www.joanneum.at/digital

This work was supported by the Austrian Research Promotion Agency (FFG) under the contract 838299 "HiTES3D", IS-Instruments and JOANNEUM RESEARCH

Jakub Rosner is a scholarship holder in project „SWIFT (Stypendia Wspomagające Innowacyjne Forum Technologii)" POKL.08.02.01 24-005/10 and in project UDA POKL-.04.-01.01-00-106/09 both supported by the European Union from the European Social Fund

Introduction

Stereo matching is used to find corresponding pixels in a pair of images in order to calculate the depth of those pixels for 3D reconstruction.

Semi-global Stereo Matching [1]

- is a well-known approach in stereo vision
- runs quite fast with acceptable accuracy
- is widely-used in photogrammetry as well

Two steps

- Cost value creation
- Semi-global optimization

$$E(D) = \sum_{\mathrm{p}} \left(C(\mathrm{p}, D_{\mathrm{p}}) + \sum_{\mathrm{q} \in N_p} P_1 \mathrm{T}[|D_{\mathrm{p}} - D_{\mathrm{q}}| = 1] + \sum_{\mathrm{q} \in N_p} P_2 \mathrm{T}[|D_{\mathrm{p}} - D_{\mathrm{q}}| > 1] \right)$$

GPU Implementation

for our heavily templatized dense matcher with

- pixel types of 8, 16 bit (s/us), float
- handling of missing pixel
- census 9 cost function
- disparity predictor and shift predictor
- disparity range 15 (half warp optimized)
- cost values in float or half float
- cost aggregation along 8 paths
- matching direction forward / backward
- integration into a hierarchical workflow

based on the new Kepler (GK104+) features using

- CUB template library (http://github.com/NVlabs/cub/)

All kernels are fully templatized, intermediate results (census and prediction) reused.

Census Transform

Census 9 transform is done in shared memory [2], but with additional handling for missing pixel data.

$$R_T(P) = \bigotimes_{[i,j] \in D} \xi(P, P + [i,j])$$

along with a bit string of existing valid pixels

$$V(P) = \bigotimes_{[i,j] \in D} \nu(P, P + [i,j])$$

and $$\nu(P) = \begin{cases} 0, & I(P) \in \mathrm{I}_{invalid} \\ 1, & \text{else} \end{cases}$$

Census- and valid bitstrings are stored in three 32 bit chunks (96 bit) each, padded to four Uints (128 bit) each for performance reasons.

Usage of cub::SHL_ADD() for 32 bit segments further increases performance.

Normalized hamming distance in 32 bit chunks:

$$d = \frac{\_popc\big((refbits \& validbits)\,\hat{}\,(searchbits \& validbits)\big)}{\_popc(validbits)}$$

Cost Aggregation

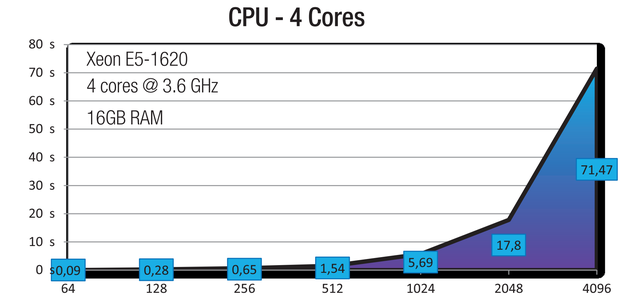
Disparity image aggregation is done in four kernels, each calculating two directions (e.g. North/South) with warp based functions like

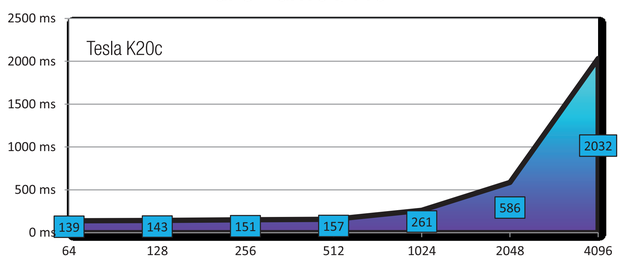
- __shfl()
- __ballot()
- __any() and
- cub::WarpReduce<datatype, 1, N>(Min())

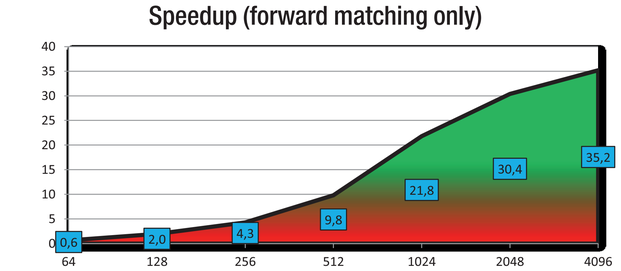
Disparities from cost selection uses subpixel accurate prediction (bilinear interpolation).

Calculation is done in float, storage is done in float or half float depending on template parameter.

Evaluation







Conclusion

- GPU implementation 20-39 times faster than CPU implementation (forward and backward matching, speedup higher for larger images)
- Kernel templates allow fine grained implementation (e.g. for more disparity ranges and matching functions) with minimal maintenance overhead
- Easily extensible to census 11 transform (120 bit) and disparity range 31
- CUB library is an excellent lightweight tool

[1] H. Hirschmüller: "Accurate and Efficient Stereo Processing by Semi-Global Matching and Mutual Information", CVPR, 2005

[2] C. D. Pantilie, S. Nedevschi: "Optimizing the Census Transform on CUDA enabled GPUs", ICCP, 2012

THE INNOVATION COMPANY