

Organizational Setup

Time line:

- Lecture (September 11-15)
 - You made it till this point, congrats!!
- Project (September 15 - October 8)
 - Implement an advanced application assigned to your group
 - Group of three students
- Demo day (October 9)
 - Prepare a presentation and demo
 - Showing off what your group achieved throughout the project phase

Grade bonus of 0.3 – 0.4:

- Deadline: **Sunday 11.59 p.m.**
- Hand in solution for all exercises (Folder per exercise with a `readme.txt`, each `readme.txt` describes how to compile code for the exercise (copy paste style) + written answers if needed)
- Each student has to hand in separately and code must be individual, i.e. copied code will not be graded and thus fail
- Grade bonus achieved, if 80% or more are correct
- Achieved grade bonus will be announced during project phase

Organizational Setup

Project Phase:

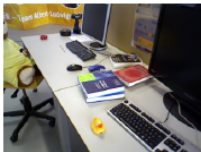
- Implement a computer vision algorithm in CUDA
- Form groups of three students per group, i.e. eight groups in total
- Pick one of the projects we suggest on Friday or
- Suggest your own project
- Let us know your group and your three preferred projects by **Friday 11.59 p.m.**
- **!!!Project workload is supposed to be full-time!!!**
- Meet your advisor regularly
- If we detect cheating, everyone involved gets the grade 5.0

Demo day:

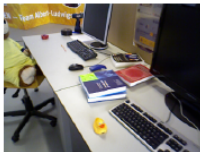
- Prepare a presentation of 15–20 minutes per group
- Explain the assigned problem/project
- How did you proceed to solve it
- Each group member presents and describes his/her task in the project
- Show your results

Robust Odometry Estimation for RGB-D Cameras

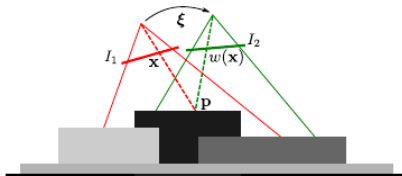
Kerl, Sturm, Cremers; ICRA 2013



(a) reference image I_1



(b) current image I_2



Problem: Given two consecutive images I_1 and I_2 , find relative pose ξ .

Given:

- Real-time CPU implementation
- Paper
- Hands-on with Kinect

Goals:

- Refactor to real-time (or even faster)
- Estimate ξ by minimizing photometric and geometric error
- Implement different weightings: Huber, Student-t, Cauchy
- Implement iteratively re-weighted least squares

Supervisor: Bjoern

Depth Super-Resolution Meets Uncalibrated Photometric Stereo

Peng, Haefner, Quéau, Cremers; ICCVW 2017



Input HR RGB images and LR depth maps



Output HR albedo and depth maps



Relighting

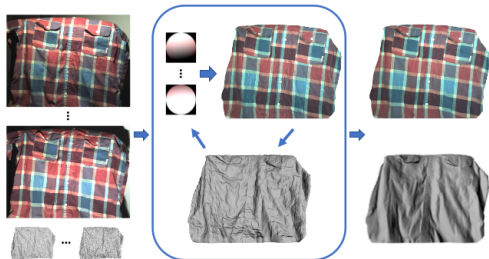
Problem: Noisy and low-resolution depth maps

$$\min_{\substack{\mathbf{z}: \Omega_{\text{HR}} \rightarrow \mathbb{R} \\ \rho: \Omega_{\text{HR}} \rightarrow \mathbb{R}^3 \\ \{\mathbf{I}^i \in \mathbb{R}^{12}\}_i}} \left\{ \lambda \sum_{i=1}^n \left\| \mathbf{A}^i(\mathbf{z}, \rho, \mathbf{I}^i)^\top \begin{bmatrix} \nabla \mathbf{z} \\ \mathbf{z} \end{bmatrix} - \mathbf{b}^i(\rho, \mathbf{I}^i) \right\|_{\ell^2(\Omega_{\text{HR}})}^2 + \sum_{i=1}^n \|\mathbf{Kz} - \mathbf{z}_0^i\|_{\ell^2(\Omega_{\text{LR}})}^2 \right\}$$

Result: High in detail and resolution depth images + true color RGB image

Depth Super-Resolution Meets Uncalibrated Photometric Stereo

Peng, Haefner, Quéau, Cremers; ICCVW 2017



Algorithm

Given:

- Fully working MATLAB code
- Paper
- Hands-on with Kinect

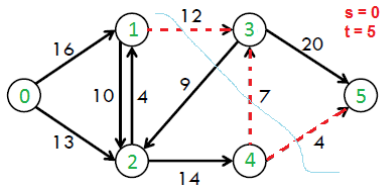
Supervisor: Bjoern

Goals:

- Port code from MATLAB to GPU
- Write specific kernel for downsampling K
- Implement conjugate gradient algorithm
- Solve linear least squares
- Smart `memcpy` between host and device

max-flow/min-cut on GPUs

- find maximum flow through graph from source to sink



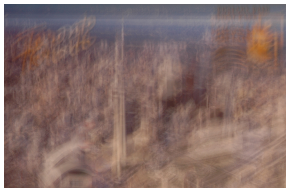
- idea:** implement massively parallel (but very simple) algorithm on GPU

$$\mathbf{x}^{t+1} = \Pi_{[0,1]^{|V|}} (\mathbf{x}^t + T(\operatorname{div}_w \mathbf{y}^t + \mathbf{f})),$$

$$\mathbf{y}^{t+1} = \Pi_{[-1,1]^{|E|}} (\mathbf{y}^t + \mathbf{S}(\nabla_w(2\mathbf{x}^{t+1} - \mathbf{x}^t))).$$

- challenge:** efficient parallel implementation of divergence and gradient operators on (huge) weighted graphs
- goal:** outperform current state-of-the-art combinatorial solvers

total variation blind deconvolution

observed input f solution u, k

- **task:** minimize the energy

$$\min_{u,k} \|u * k - f\|_2^2 + \lambda \|\nabla u\|_1, \quad \text{s.t. } k \geq 0, |k|_1 = 1.$$

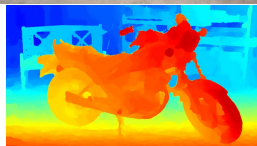
- **required:** clean & efficient CUDA implementation of convolution (and its transpose) with various boundary conditions/modes
- **MATLAB reference implementation available:**
<http://www.cvg.unibe.ch/dperrone/tvdb/index.html>

Depth Adaptive Diffusion



Pock, et al.: “A convex formulation of continuous multi-label problems,” *ECCV 2008*

- ▶ General idea: From two rectified stereo images we compute a depth map. Then we run a diffusion on the image, where the diffusion tensor depends on the depth.
- ▶ What to do:
 1. Implement stereo matching using a functional lifting approach. (\Rightarrow *Paper*)
 2. Diffuse the image depending on the result.
 3. Generate nice results.

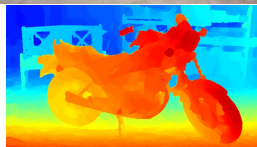


Depth Adaptive Diffusion



Pock, et al.: "A convex formulation of continuous multi-label problems," *ECCV 2008*

- ▶ General idea: From two rectified stereo images we compute a depth map. Then we run a diffusion on the image, where the diffusion tensor depends on the depth.
- ▶ What to do:
 1. Implement stereo matching using a functional lifting approach. (\Rightarrow *Paper*)
 2. Diffuse the image depending on the result.
 3. Generate nice results.

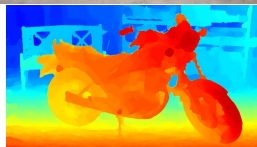


Depth Adaptive Diffusion



Pock, et al.: "A convex formulation of continuous multi-label problems," *ECCV 2008*

- ▶ General idea: From two rectified stereo images we compute a depth map. Then we run a diffusion on the image, where the diffusion tensor depends on the depth.
- ▶ What to do:
 1. Implement stereo matching using a functional lifting approach. (\Rightarrow *Paper*)
 2. Diffuse the image depending on the result.
 3. Generate nice results.



Project: Convolutional Networks with Broadcasting

- 2D convolutional networks use $X \times Y \times C$ data as input, where C = number of channels (e.g. color channels)
- In some applications:
 - We have $X \times Y \times C_1$, $X \times 1 \times C_2$, $1 \times Y \times C_3$ data
 - We want to replicate certain rows/columns to obtain $X \times Y \times C_1$, $X \times Y \times C_2$, $X \times Y \times C_3$, respectively
 - Concatenate to $X \times Y \times (C_1 + C_2 + C_3)$
 - Apply convolutional layer
- But this can be done memory-efficiently, by smart indexing instead of making redundant copies of data
 - This is called “**broadcasting**”, explained for NumPy here: <https://docs.scipy.org/doc/numpy/user/basics.broadcasting.html>
- Several possibilities to implement ConvNets with broadcasting:
 - Implement broadcasting for symbolic concatenation operation
 - Implement broadcasting (and several inputs) for convolutional operation, for example for this one: <https://github.com/Theano/Theano/blob/master/theano/gpuarray/blas.py#L444>

Project: Neural Networks with Sparse Weight Matrices

- In fully-connected neural networks, some input features are irrelevant for some output features
- Thus, the weight matrix can be constrained to have a certain sparsity structure
- This saves memory and helps to prevent overfitting
- Network learns to use given sparsity structure optimally
- Sparsity patterns can be simple, or advanced, inspired by:
 - Distribution of data
 - Progress of training
 - Typical prediction errors
- Goal: explore state of the art and possibilities, implement, optionally test