GPU Programming in Computer Vision

Final Projects

Thomas Möllenhoff, Robert Maier, Mohamed Souiai, Caner Hazirbas

Winter Semester 2014/2015

Project Phase (March 10 - March 29)

- Form groups of 3 people
- Implement a computer vision algorithm in CUDA
  - Select your 3 favorite topics
  - We will assign the projects to the groups
- Regular meetings with your supervisor
- Send source code to your supervisor until April 1
- Cheating: all involved groups will get the grade 5.0

Presentations (March 30 - April 1)

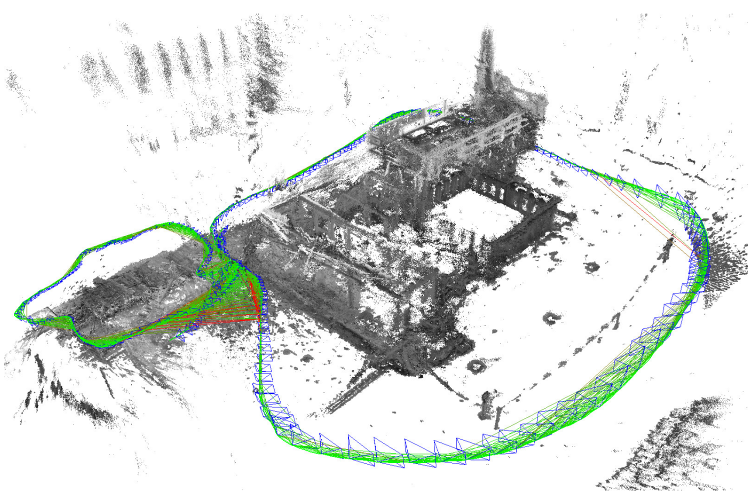
- 15 minutes per group
- Prepare slides
  - Explain the task
  - Explain how you proceeded to solve the task
  - Show your results
- Live demo
- Q&A session

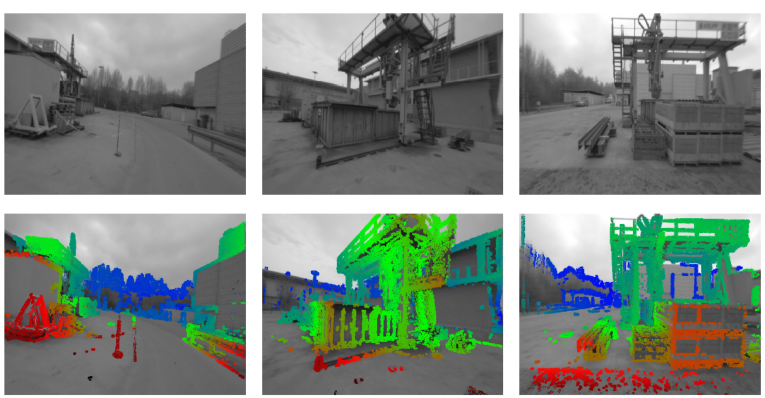
Final Project Proposals

Implement your own project idea?

1) TGV-Impainting of Semi-Dense Depth Maps (Jörg Stückler)
2) Plane Detection in RGB-D images (Lingni Ma)
3) Large Displacement Optical Flow (Mohamed Souiai)
4) Poisson Image Editing (Thomas Möllenhoff)
5) Image Smoothing via L0 Gradient Minimization (Thomas Möllenhoff)
6) Dense Visual Odometry (Robert Maier)
7) Voxel Hashing (Robert Maier)

TGV-Impainting of Semi-Dense Depth Maps





LSD-SLAM estimates semi-dense depth maps in real-time

Task:
Implement fast regularization and impainting of semi-dense depth maps (f) using Total Generalized Variation

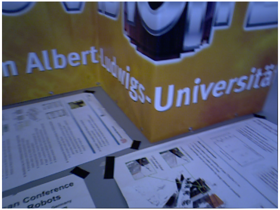
$$\min_{u,v} \left\{ \alpha_0 \int_\Omega |\nabla u - v| \, dx + \alpha_1 \int_\Omega |\nabla v| \, dx + \int_\Omega \rho \, (u - f) \, dx \right\}$$

A slow matlab implementation (CPU) is available for reference

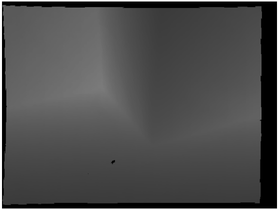
Supervisors:
Thomas Möllenhoff, Jörg Stückler

Project Introduction

plane detection in RGB-D images[1]



RGB image



depth image

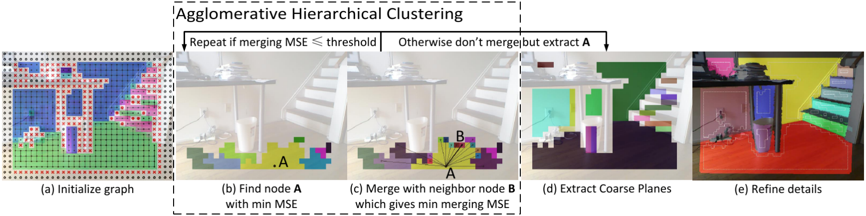
segmented image

CPU implementation: 15ms. GPU: ?

[1] Feng et al, *"fast plane extraction in organized point clouds using agglomerative hierarchical clustering"*, ICRA 2014

Algorithm Description

- convert depth image into organized point cloud
- divide cloud into blocks and compute least-square plane fitting for each block
- block-wise region merging
- pixel-wise region growing



Project Structure

convert depth images

- basic GPU programming
- memory allocation
- pixel-wise parallelization

$$\begin{cases} x = \dfrac{u - o_x}{f_x} \times d \\ y = \dfrac{v - o_y}{f_y} \times d \\ z = d \end{cases}$$

Project Structure

least-square plane fitting

- more advanced GPU programming
- shared memory
- atomic operations
- tree reduction

$$\underset{\mathbf{n},d}{\operatorname{argmin}} \sum_{i=1}^{M} (\mathbf{n}^T \mathbf{p}_i + d)^2$$

Project Structure

and more. . .

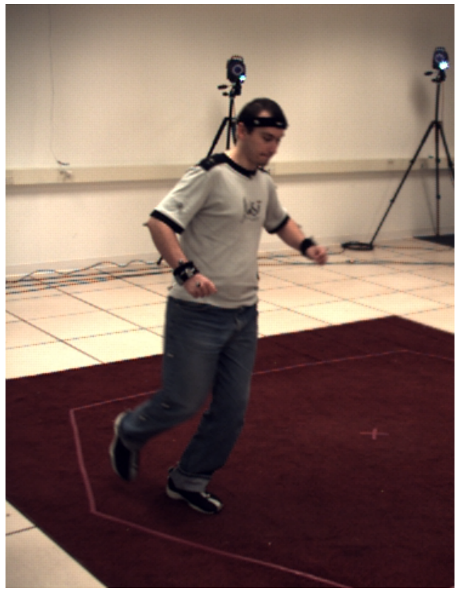
camera tracking with the detected planes

Large Displacement Optical Flow

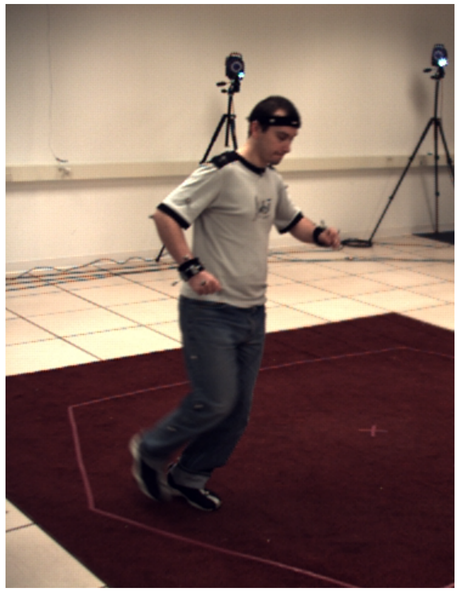
Mohamed Souiai

Different Optical Flow Strategies

Different Optical Flow Strategies

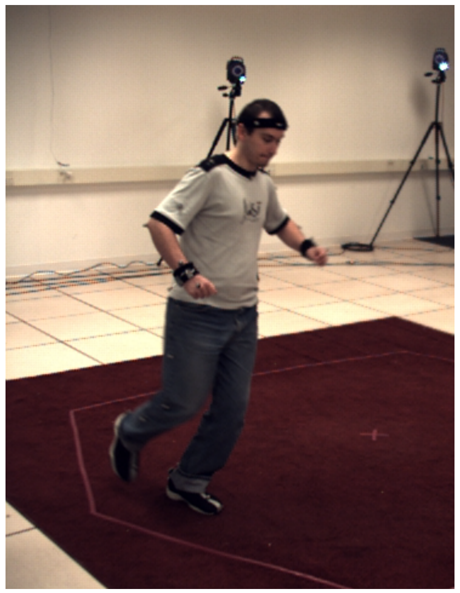


Frame 1

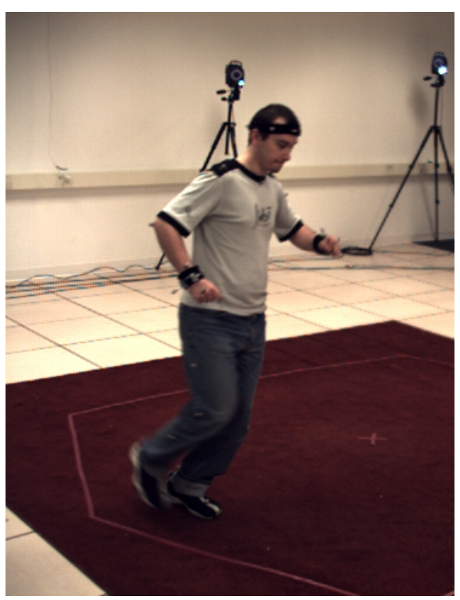


Frame 2

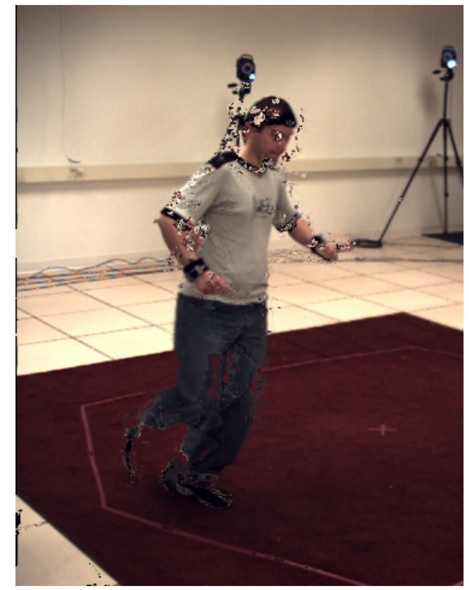
Different Optical Flow Strategies

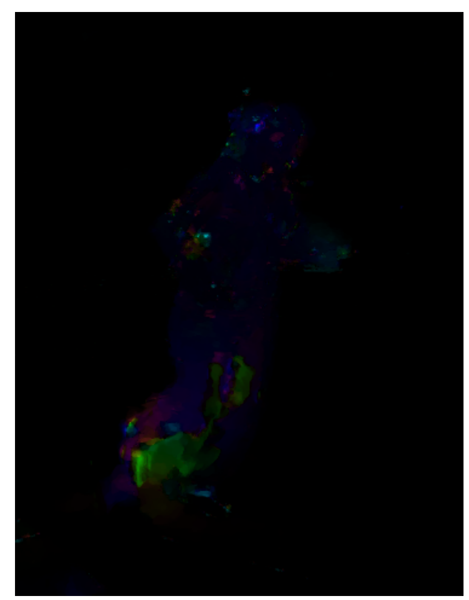


Frame 1



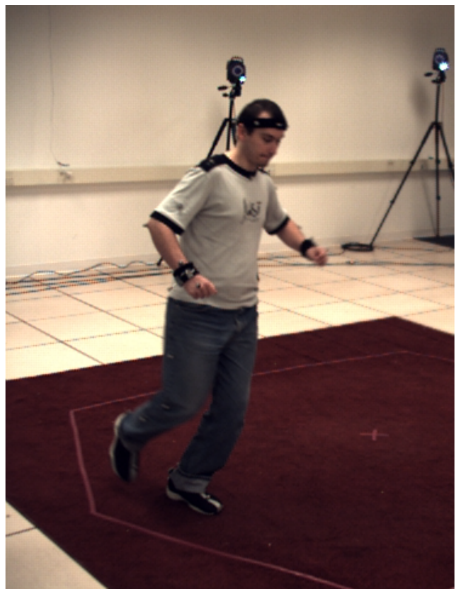
Frame 2



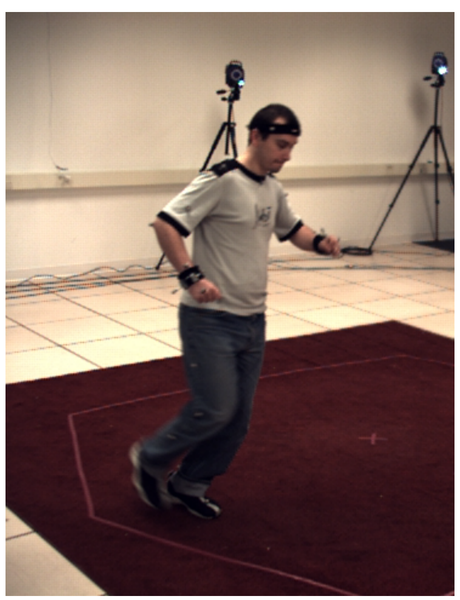


Coarse to fine OF

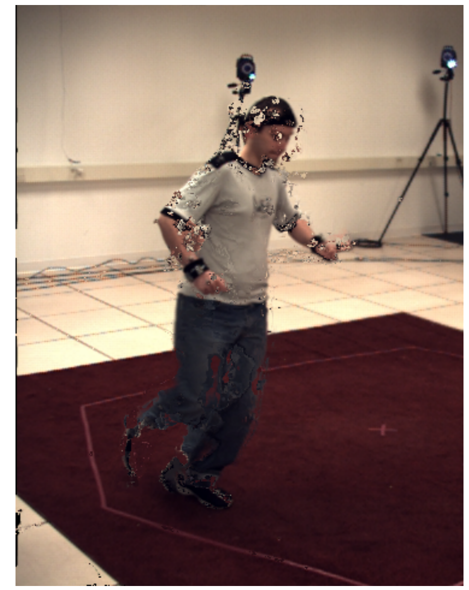
Different Optical Flow Strategies

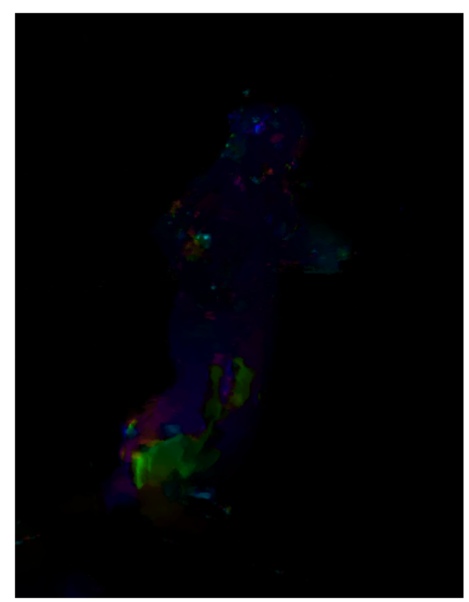


Frame 1

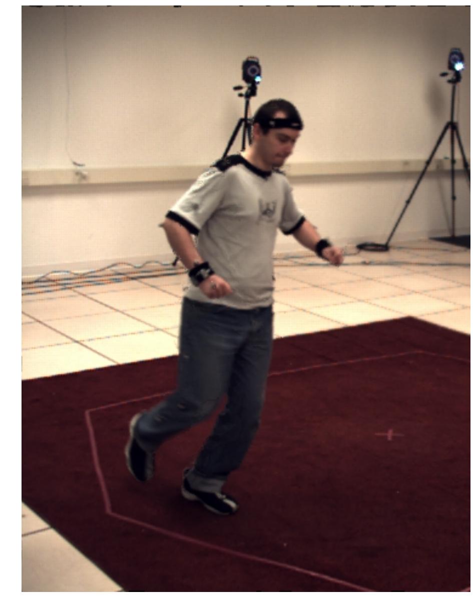


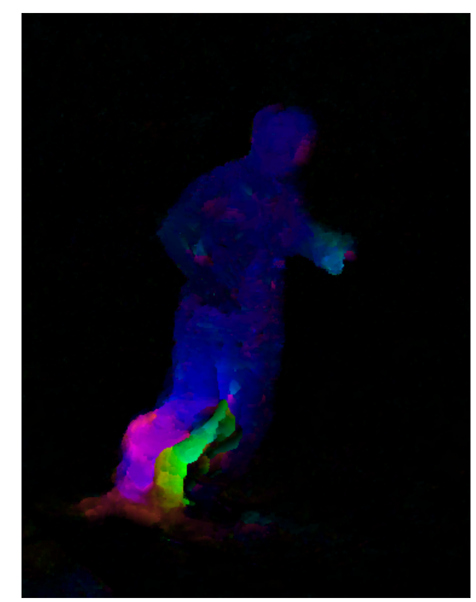
Frame 2





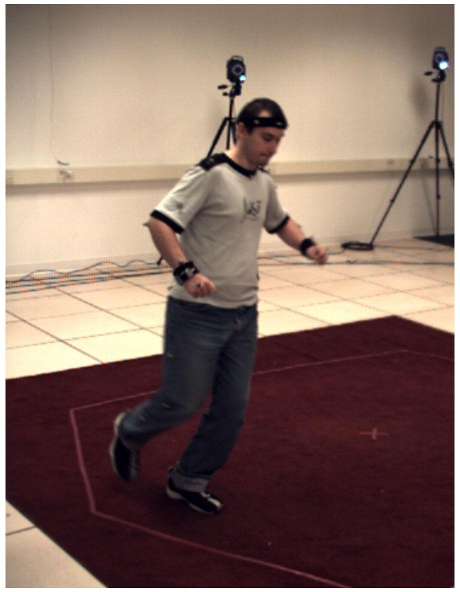
Coarse to fine OF



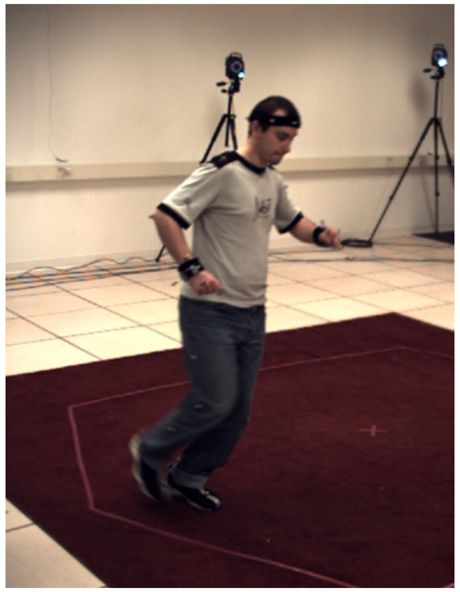


Large displ. OF

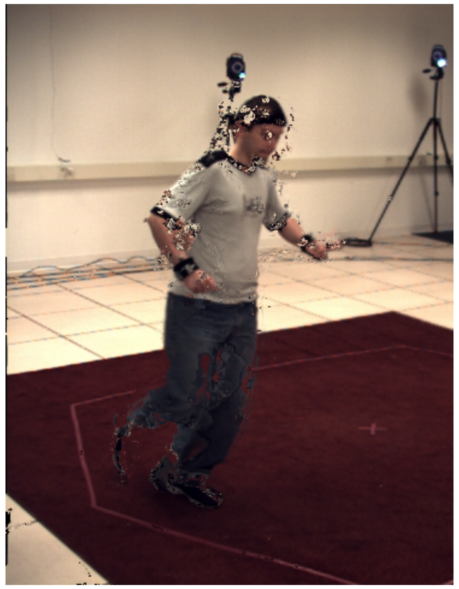
Different Optical Flow Strategies

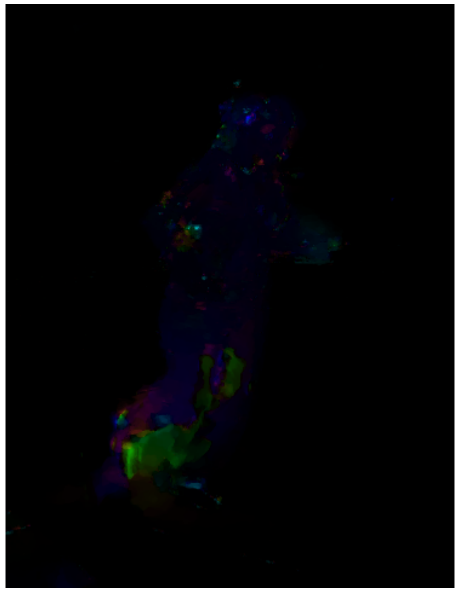


Frame 1

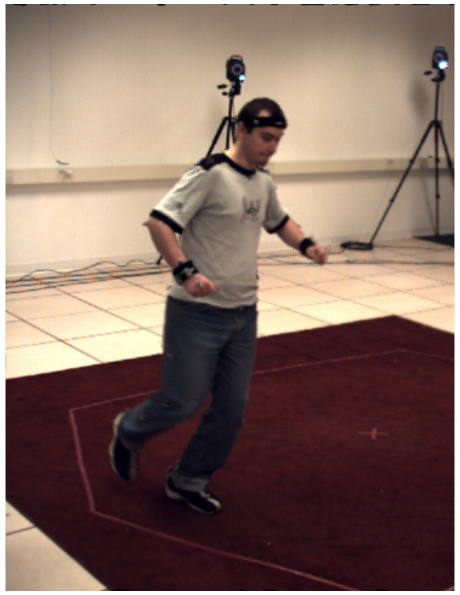


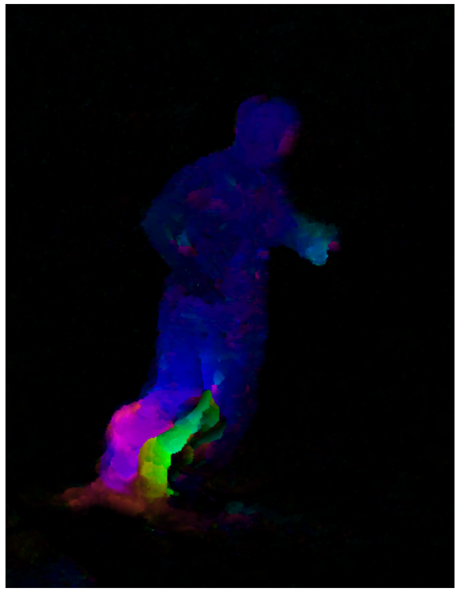
Frame 2





Coarse to fine OF





Large Displ. OF

Large Displacement Optical Flow

Large Displacement Optical Flow

Cost Function

$$E(v,u) = \int_{\Omega} \lambda\rho(v,x) + \frac{1}{2\theta}(v-u)^2 + \psi(\nabla u)\, d^2x.$$

Large Displacement Optical Flow

Cost Function

$$E(v,u) = \int_{\Omega} \lambda\rho(v,x) + \frac{1}{2\theta}(v-u)^2 + \psi(\nabla u)\, d^2x.$$

Strategy:

Large Displacement Optical Flow

Cost Function

$$E(v,u) = \int_{\Omega} \lambda\rho(v,x) + \frac{1}{2\theta}(v-u)^2 + \psi(\nabla u)\, d^2x.$$

Strategy:

- Perform complete search in non convex data term.

Large Displacement Optical Flow

Cost Function

$$E(v,u) = \int_\Omega \lambda\rho(v,x) + \frac{1}{2\theta}(v-u)^2 + \psi(\nabla u)\, d^2x.$$

Strategy:

- Perform complete search in non convex data term.
- Alternate regularization and data term via quadratic coupling.

Large Displacement Optical Flow

Cost Function

$$E(v,u) = \int_\Omega \lambda\rho(v,x) + \frac{1}{2\theta}(v-u)^2 + \psi(\nabla u)\, d^2x.$$

Strategy:

- Perform complete search in non convex data term.
- Alternate regularization and data term via quadratic coupling.
- Perform both steps in parallel using the GPU.

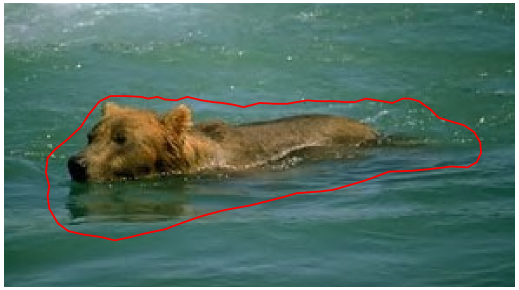
Image Smoothing via L0 Gradient Minimization

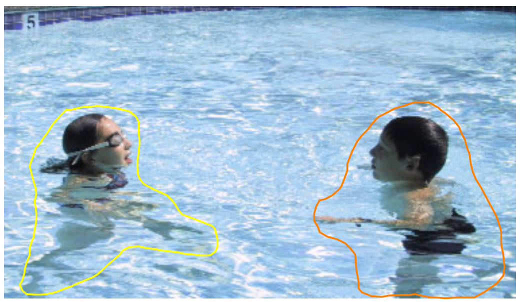
$$C(S) = \#\left\{ p \;\middle|\; |\partial_x S_p| + |\partial_y S_p| \neq 0 \right\}$$

$$\min_S \left\{ \sum_p (S_p - I_p)^2 + \lambda \cdot C(S) \right\}$$

Paper: L. Xu, C.Lu, Y. Xu, J. Jia, Image Smoothing via L0 Gradient Minimization, SIGGRAPH ASIA 2011, [pdf]

Poisson Image Editing



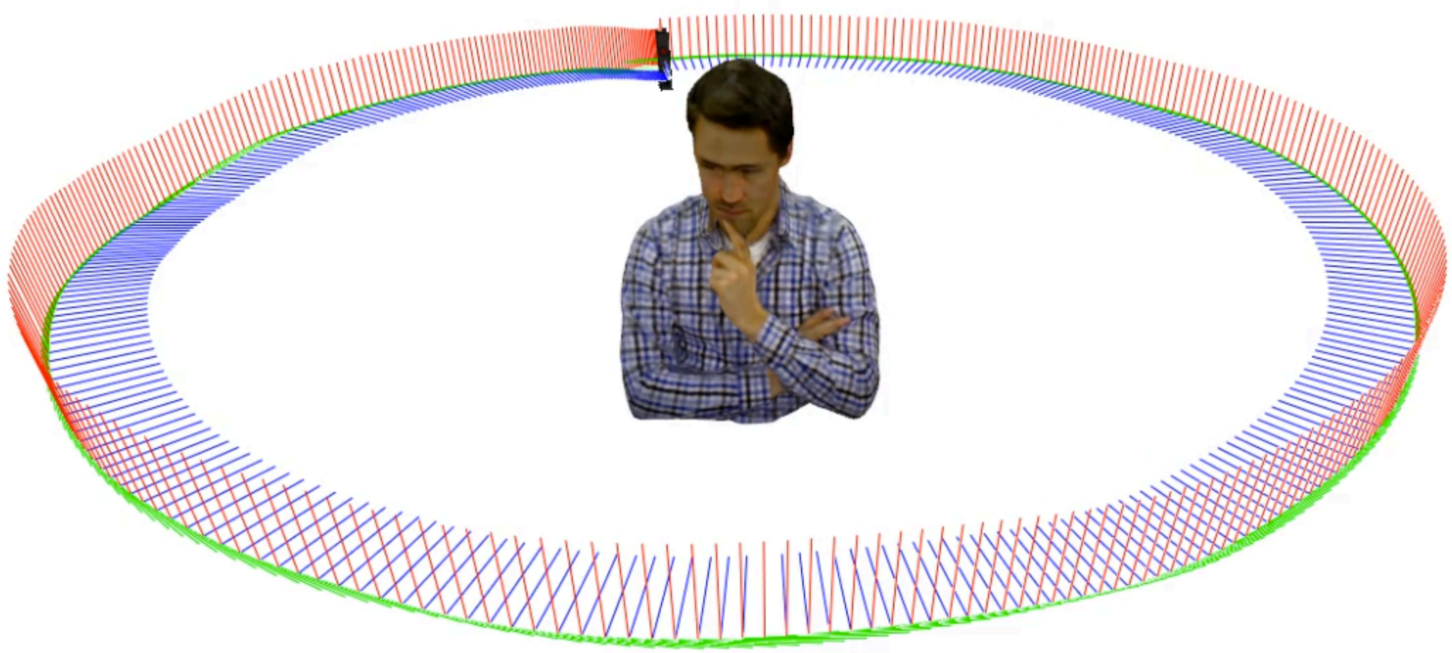


$$\min_f \iint_\Omega |\nabla f - \mathbf{v}|^2 \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

$$\Delta f = \operatorname{div}\mathbf{v} \text{ over } \Omega, \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

Paper: P. Pérez, M. Gagnet, A. Blake, Poisson Image Editing, SIGGRAPH 2003
[pdf]

Dense Visual Odometry

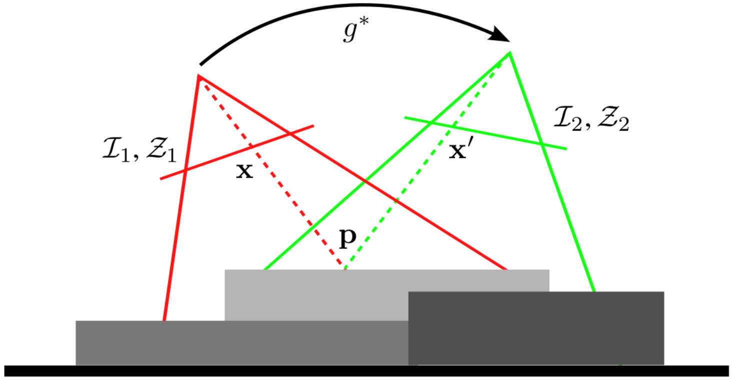


Supervisor: Robert Maier

Dense Visual Odometry

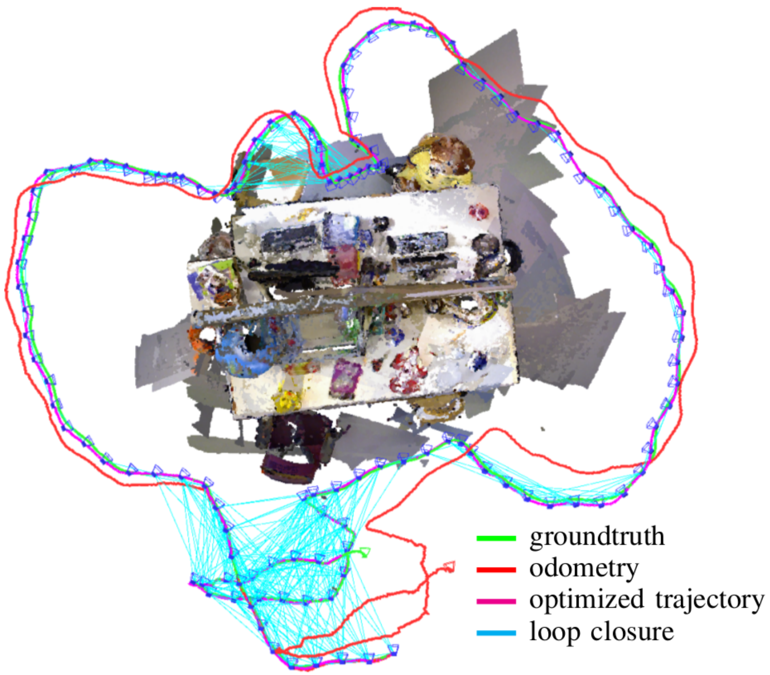
- Robust Odometry Estimation for RGB-D Cameras
  - Given: Two RGB-D frames

  - Goal: estimate camera motion g^* by minimizing photometric and geometric error



- Real-time CPU implementation (320x240)

Dense Visual Odometry



- Possible extensions:
  - Full DVO-SLAM system
  - Google Tango tablet

- References:
  - Robust Odometry Estimation for RGB-D Cameras, Kerl et al, ICRA 2013 [pdf] [github]
  - Dense Visual SLAM for RGB-D Cameras, Kerl et al, IROS 2013 [pdf]
  - Compare to GPU implementation of Whelan et al (ICRA 2013) [pdf]

Voxel Hashing

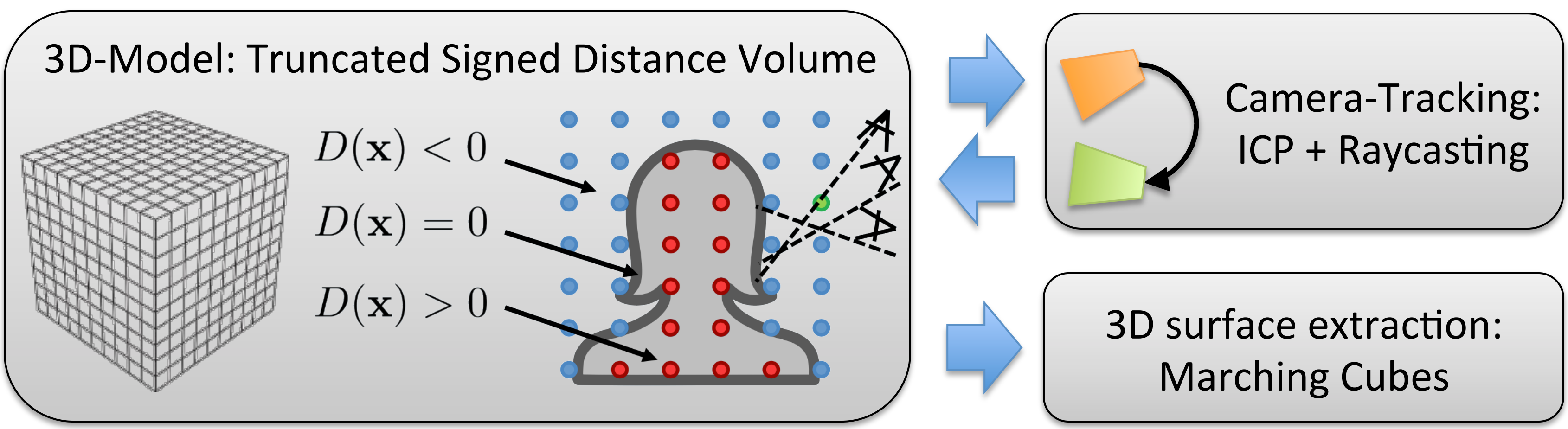
Supervisor: Robert Maier

References: Real-time 3D Reconstruction at Scale using Voxel Hashing, Nießner et al, TOG 2013 [pdf] [github] [web]

Voxel Hashing

- KinectFusion (Newcombe et al, ISMAR 2011): Real-time dense 3D reconstruction from RGB-D sensors

- 3D reconstruction algorithm:



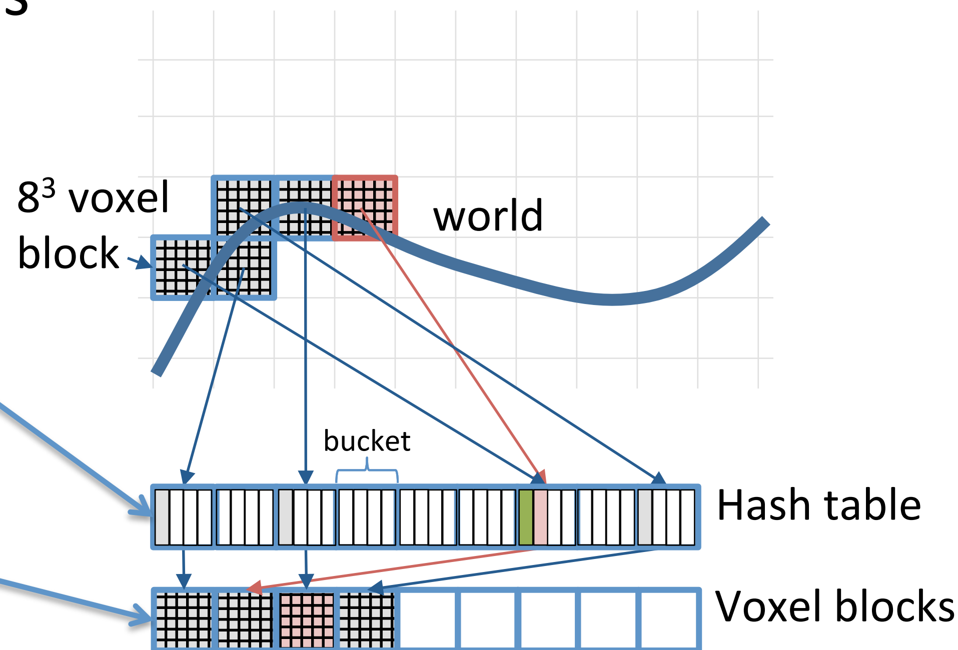
- But: limited 3D volume size, high memory consumption

Voxel Hashing

- Idea: Replace TSDF volume with unstructured 3D scene representation: High resolution, efficient updates
- Spatial hashing: $H(x, y, z) = (x \cdot p_1 \oplus y \cdot p_2 \oplus z \cdot p_3)\, mod\, n$
- Buckets to resolve collisions
- Stream blocks outside ROI to CPU (and vice versa)

```
struct HashEntry {
  short position[3];
  short offset;
  int   pointer;
};
```

```
struct Voxel {
  float sdf;
  uchar colorRGB[3];
  uchar weight;
};
```



Next steps

- Today: send email to cuda-ws1415@vision.in.tum.de
  - Group Members
  - Your 3 favorite topics

- After project assignments: meet with your supervisor

- Any questions?