Dense Visual Odometry

Christoph Ihrke, Josef Brandl, Duy Nguyen

Technische Universität München
Department of Informatics
Computer Vision Group

October 6, 2015

Outline

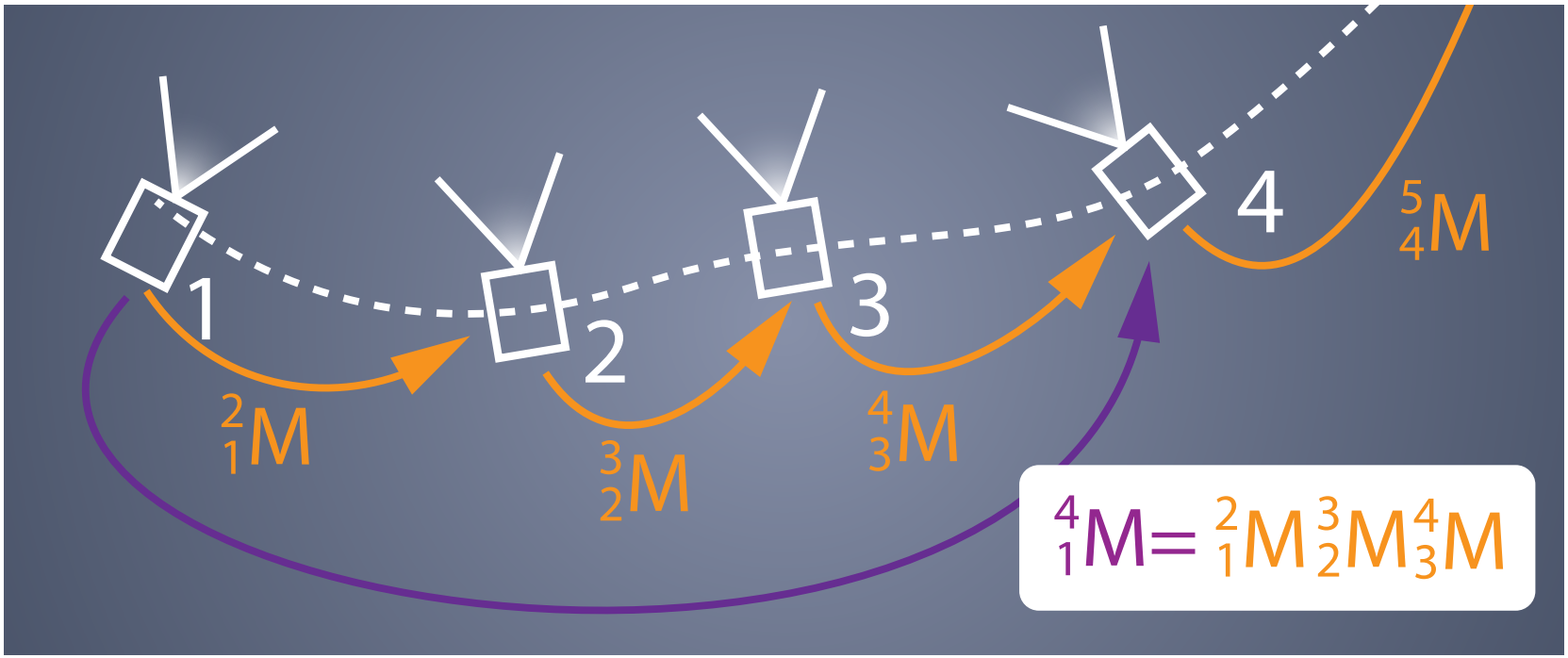
Outline

Introduction

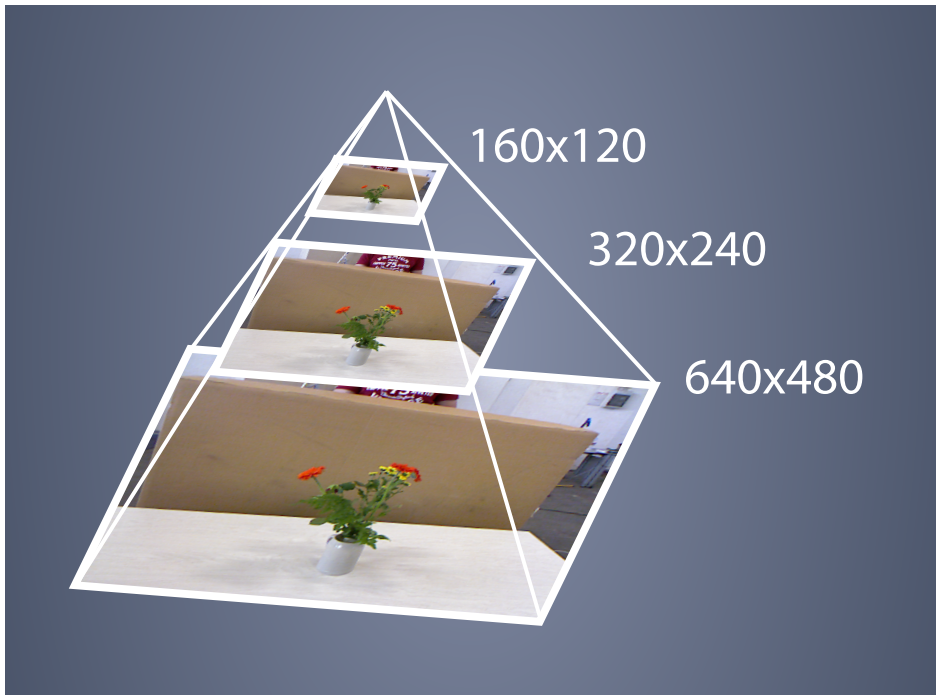
- Dense Visual Odometry
  - Visual Odometry: estimate position from images.
  - Dense: every pixel $\rightarrow$ feature
- Motivation
  - often required, e.g: for volumetric reconstruction
  - current CPU implementation can be parallelized.

Outline

Overview

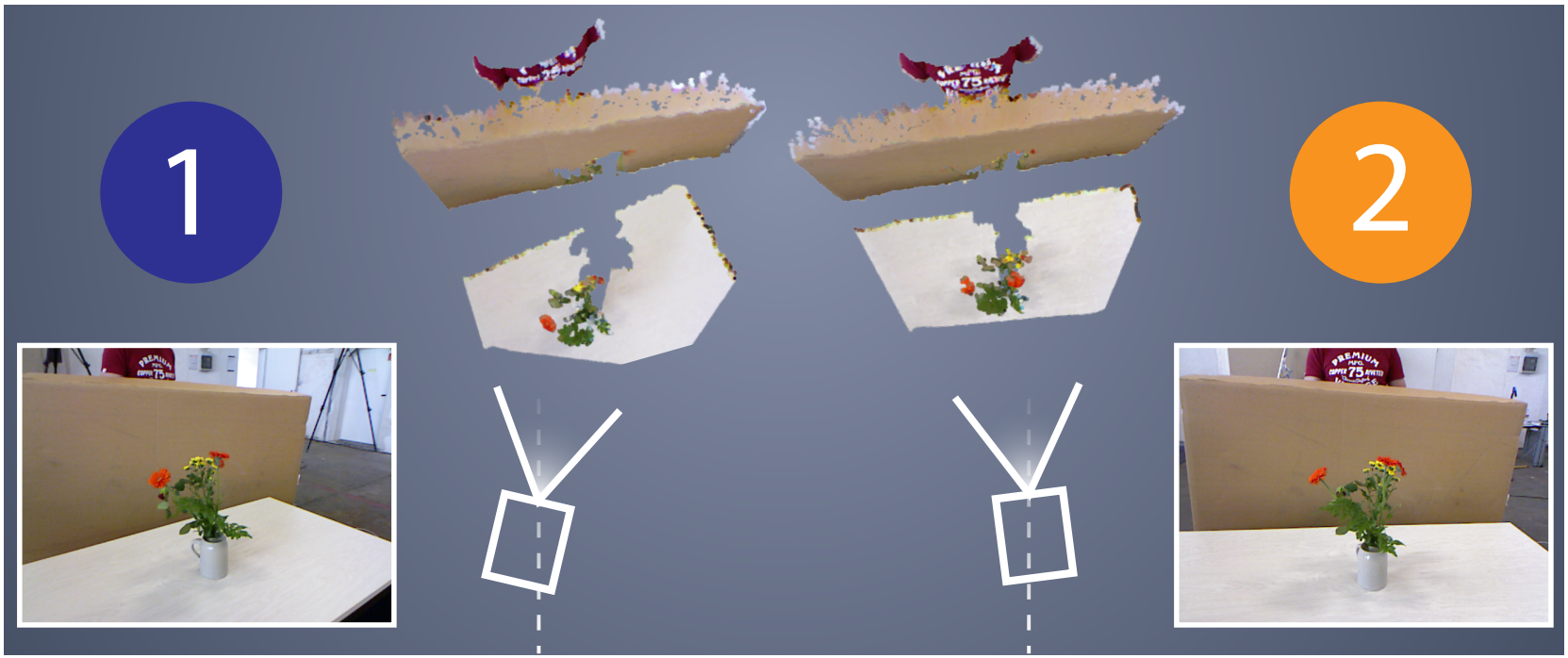


Pyramid approach



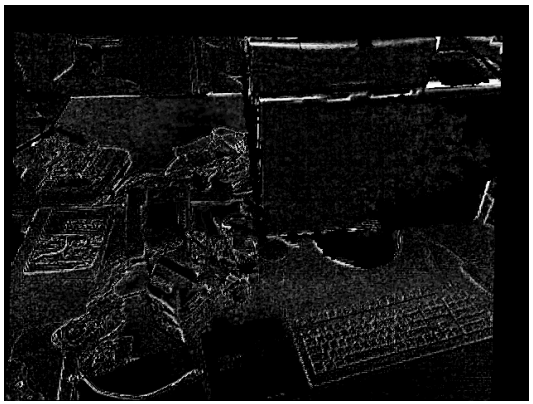
- Downsample images.
- Process coarser to finer levels.

Consecutive frame alignment



$$E(\xi) = \sum_{\boldsymbol{x} \in \Omega} \left(I_{prev}(\boldsymbol{x}) - I_{cur}(\omega(\boldsymbol{x}, D_{prev}(\boldsymbol{x}), \xi)) \right)^2 \qquad (1)$$

Consecutive frame alignment



- Non-linear least squares problem: $\min E(\xi) \rightarrow \xi$
- Algorithms:
  - Gradient descent
  - Gauss-Newton
  - Levenberg-Marquardt
- Robust weights:
  - eliminate outliers
  - Huber, T-Dist

Outline

GPU implementation

- Custom kernels:
  - Downsampling
  - Residual & Jacobian:
    - Numeric Jacobian: approximate derivatives
    - Analytic Jacobian: closed form
- cuBLAS for large matrix operations:
  - A = J^TJ (6xn · nx6)
  - b = J^Tr (6xn · nx1)
- Eigen for some small matrix operations

Lessons learned

- Custom matrix struct with operations on GPU
  - Avoid calling Eigen (in CPU)
  - No memory copying back and forth
- Matrices stored in column major
  - Consistent memory storage: cuBLAS, Eigen and our own
- Use existing library such as cuBLAS for common problem
- **cublasSgemm()** is faster than **cublasSgemv()**

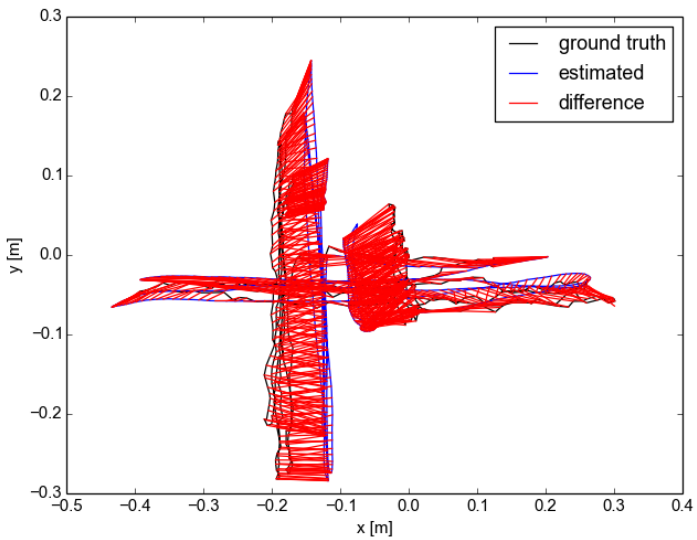
Outline

Performance

- CPU vs. GPU:

	CPU	GPU
Absolute Trajectory Error (ATE)	**~0.07**	**~0.06**
Avg. time per frame (ms)	**~150**	**~20**

- Freiburg 1 dataset:



Outline

Conclusion

- Problem was easily parallelizable
- Noticable speed-up with GPU

DEMO