Dense Visual Odometry

Christoph Ihrke, Josef Brandl, Duy Nguyen

Technische Universität München
Department of Informatics
Computer Vision Group

October 6, 2015
Outline

1. Introduction
2. Algorithm
3. Implementation
4. Performance
5. Conclusion
Outline

1. Introduction
2. Algorithm
3. Implementation
4. Performance
5. Conclusion
Introduction

- Dense Visual Odometry
  - Visual Odometry: estimate position from images.
  - Dense: every pixel → feature

- Motivation
  - often required, e.g: for volumetric reconstruction
  - current CPU implementation can be parallelized.
Outline

1. Introduction
2. Algorithm
3. Implementation
4. Performance
5. Conclusion
Overview

\[
\frac{4}{1} M = \frac{2}{1} M \frac{3}{2} M \frac{4}{3} M
\]

Christoph Ihrke, Josef Brandl, Duy Nguyen: Dense Visual Odometry
Pyramid approach

- Downsample images.
- Process coarser to finer levels.
Consecutive frame alignment

\[
E(\xi) = \sum_{x \in \Omega} (I_{\text{prev}}(x) - I_{\text{cur}}(\omega(x, D_{\text{prev}}(x), \xi)))^2
\] (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

1. Introduction
2. Algorithm
3. Implementation
4. Performance
5. Conclusion
GPU implementation

- Custom kernels:
  - Downsampling
  - Residual & Jacobian:
    - Numeric Jacobian: approximate derivatives
    - Analytic Jacobian: closed form

- cuBLAS for large matrix operations:
  - \( A = J^T J \) (6xn \cdot nx6)
  - \( b = J^T r \) (6xn \cdot 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

1. Introduction
2. Algorithm
3. Implementation
4. Performance
5. Conclusion
Performance

- **CPU vs. GPU:**

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute Trajectory Error (ATE)</td>
<td>~0.07</td>
<td>~0.06</td>
</tr>
<tr>
<td>Avg. time per frame (ms)</td>
<td>~150</td>
<td>~20</td>
</tr>
</tbody>
</table>

- **Freiburg 1 dataset:**

![Graph showing performance comparison between CPU and GPU for Freiburg 1 dataset]
Outline

1. Introduction
2. Algorithm
3. Implementation
4. Performance
5. Conclusion
Conclusion

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