Visual Navigation for Flying Robots

Bundle Adjustment and Dense 3D Reconstruction

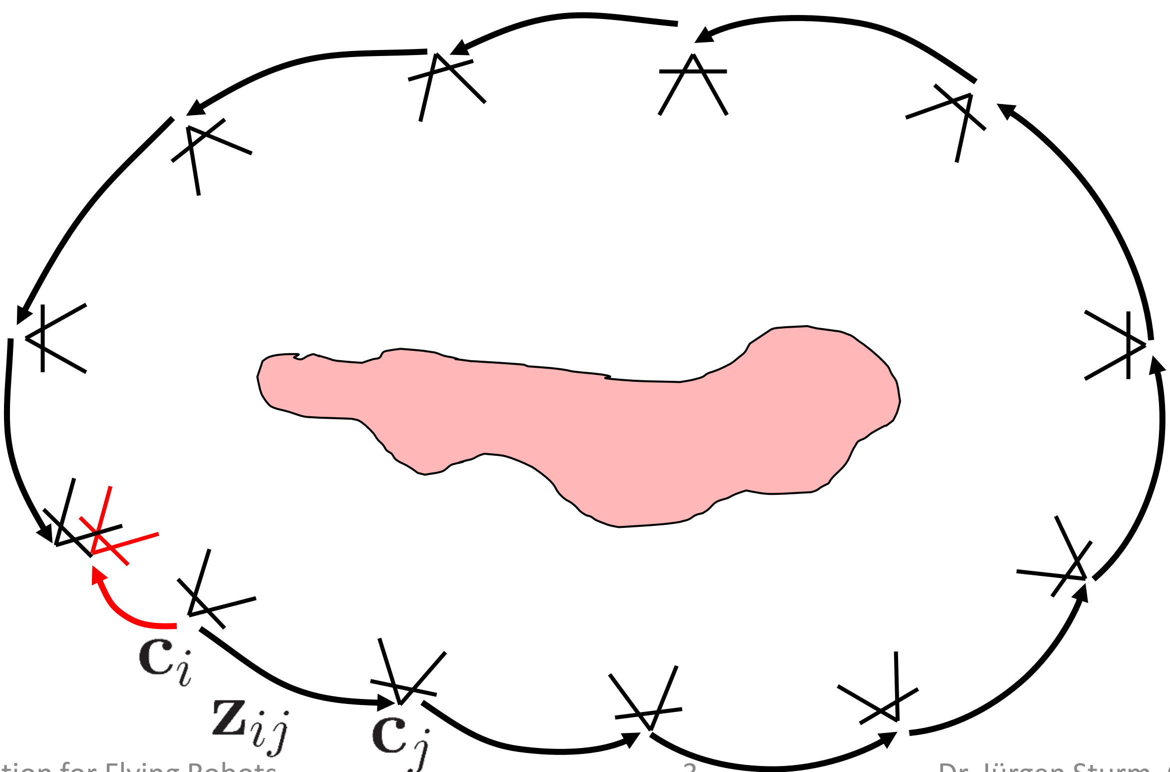
Dr. Jürgen Sturm

Agenda for Today

- Bundle adjustment
- Depth cameras
- Occupancy grid maps
- Signed distance functions

Reminder: Pose Graph SLAM

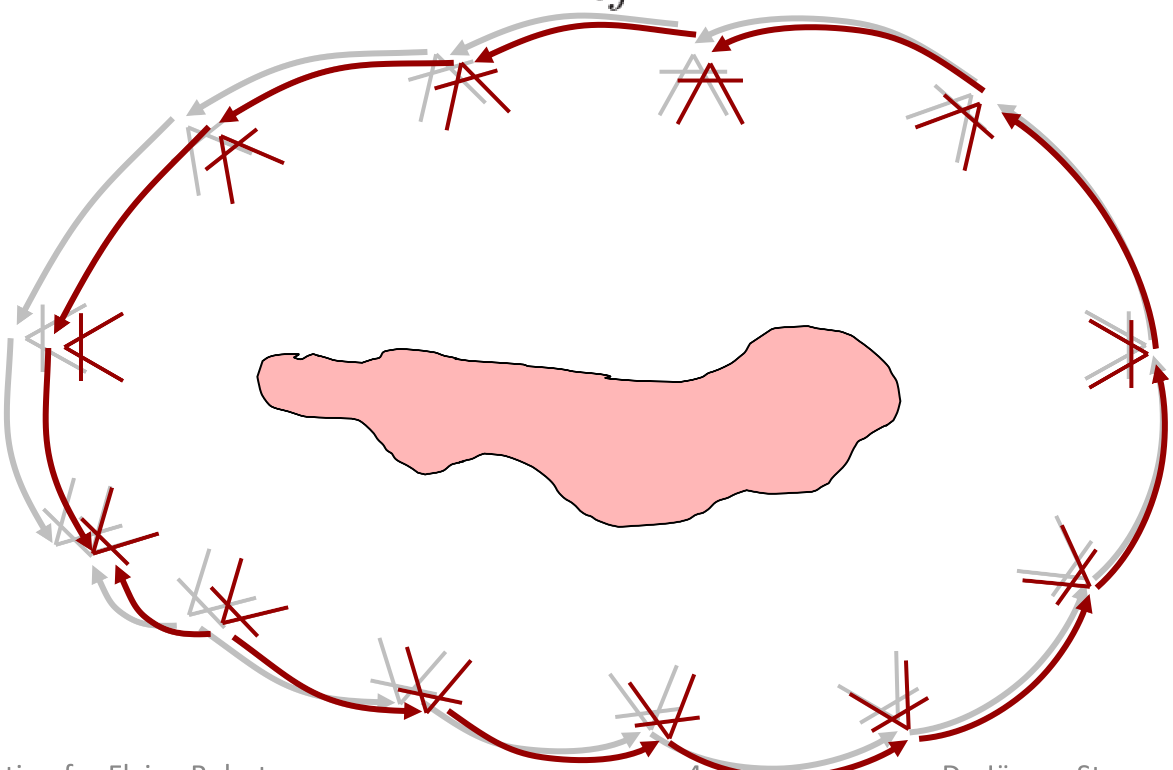
- **Given:** Set of relative pose observations $\mathbf{z}_{ij} \in \mathbb{R}^6$
- **Wanted:** Set of camera poses $\mathbf{c}_1, \ldots, \mathbf{c}_n \in \mathbb{R}^6$



Reminder: Pose Graph SLAM

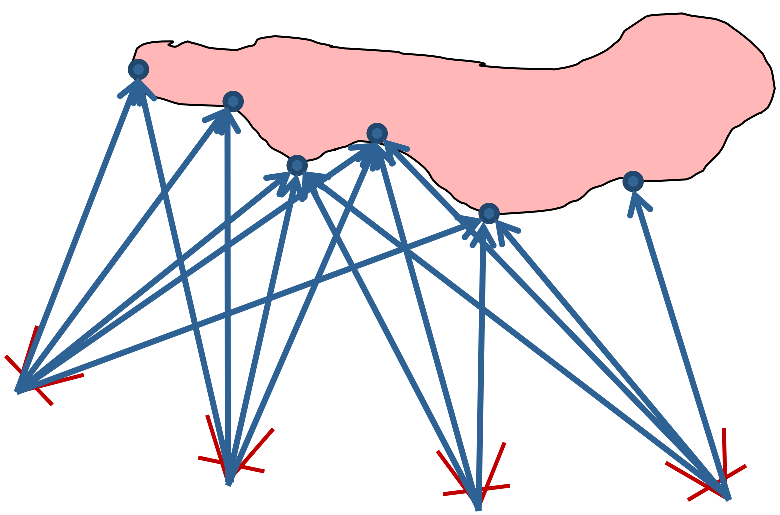
- **Goal:** Minimize the error over all constraints

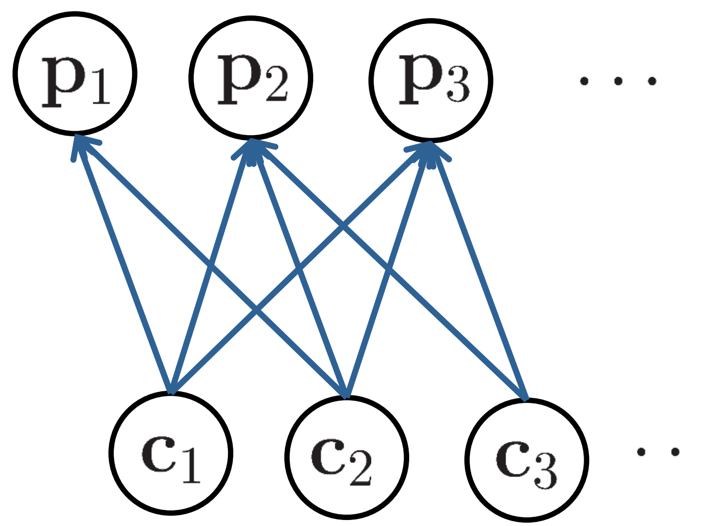
$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \sum_{ij} \mathbf{e}_{ij}(\mathbf{x})^\top \Sigma_{ij}^{-1} \mathbf{e}_{ij}(\mathbf{x})$$



Bundle Adjustment

- Each camera sees several points
- Each point is seen by several cameras
- Cameras are independent of each other (given the points), same for the points





Bundle Adjustment

- **Graph SLAM:** Optimize (only) the camera poses

$$\mathbf{x} = (\mathbf{c}_1^\top, \ldots, \mathbf{c}_n^\top)^\top \in \mathbb{R}^{6n}$$

- **Bundle Adjustment:** Optimize both 6DOF camera poses and 3D (feature) points

$$\mathbf{x} = (\underbrace{\mathbf{c}_1^\top, \ldots, \mathbf{c}_n^\top}_{\mathbf{c} \in \mathbb{R}^{6n}}, \underbrace{\mathbf{p}_1^\top, \ldots, \mathbf{p}_m^\top}_{\mathbf{p} \in \mathbb{R}^{3m}})^\top \in \mathbb{R}^{6n+3m}$$

- Typically $m \gg n$ (why?)

Error Function

- Camera pose $\mathbf{c}_i \in \mathbb{R}^6$
- Feature point $\mathbf{p}_j \in \mathbb{R}^3$
- Observed feature location $\mathbf{z}_{ij} \in \mathbb{R}^2$
- Expected feature location

$$g(\mathbf{c}_i, \mathbf{p}_j) = R_i^\top (\mathbf{t}_i - \mathbf{p}_j)$$

$$h(\mathbf{c}_i, \mathbf{p}_j) = g_{x,y}(\mathbf{c}_i, \mathbf{p}_j) / g_z(\mathbf{c}_i, \mathbf{p}_j)$$

Error Function

- Difference between observation and expectation

$$\mathbf{e}_{ij} = \mathbf{z}_{ij} - h(\mathbf{c}_i, \mathbf{p}_j)$$

- Error function

$$f(\mathbf{c}, \mathbf{p}) = \sum_{ij} \mathbf{e}_{ij}^\top \Sigma^{-1} \mathbf{e}_{ij}$$

- Covariance Σ is often chosen isotropic and on the order of one pixel

Primary Structure

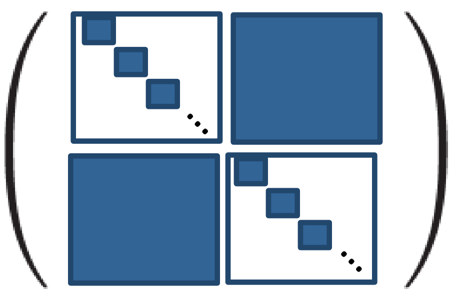
- Characteristic structure

$$\begin{pmatrix} J_{\mathbf{c}}^\top J_{\mathbf{c}} & J_{\mathbf{c}}^\top J_{\mathbf{p}} \\ J_{\mathbf{p}}^\top J_{\mathbf{c}} & J_{\mathbf{p}}^\top J_{\mathbf{p}} \end{pmatrix} \begin{pmatrix} \Delta \mathbf{c} \\ \Delta \mathbf{p} \end{pmatrix} = \begin{pmatrix} -J_{\mathbf{c}}^\top \mathbf{e}_{\mathbf{c}} \\ -J_{\mathbf{p}}^\top \mathbf{e}_{\mathbf{p}} \end{pmatrix}$$

$$\begin{pmatrix} H_{\mathbf{cc}} & H_{\mathbf{cp}} \\ H_{\mathbf{pc}} & H_{\mathbf{pp}} \end{pmatrix} \begin{pmatrix} \Delta \mathbf{c} \\ \Delta \mathbf{p} \end{pmatrix} = \begin{pmatrix} -J_{\mathbf{c}}^\top \mathbf{e}_{\mathbf{c}} \\ -J_{\mathbf{p}}^\top \mathbf{e}_{\mathbf{p}} \end{pmatrix}$$

Primary Structure

- **Insight:** $H_{\mathbf{cc}}$ and $H_{\mathbf{pp}}$ are block-diagonal (because each constraint depends only on one camera and one point)



$$\begin{pmatrix} \Delta\mathbf{c} \\ \Delta\mathbf{p} \end{pmatrix} = \begin{pmatrix} -J_{\mathbf{c}}^{\top}\mathbf{e}_{\mathbf{c}} \\ -J_{\mathbf{p}}^{\top}\mathbf{e}_{\mathbf{p}} \end{pmatrix}$$

- This can be efficiently solved using the Schur Complement

Schur Complement

- Given: Linear system

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{a} \\ \mathbf{b} \end{pmatrix}$$

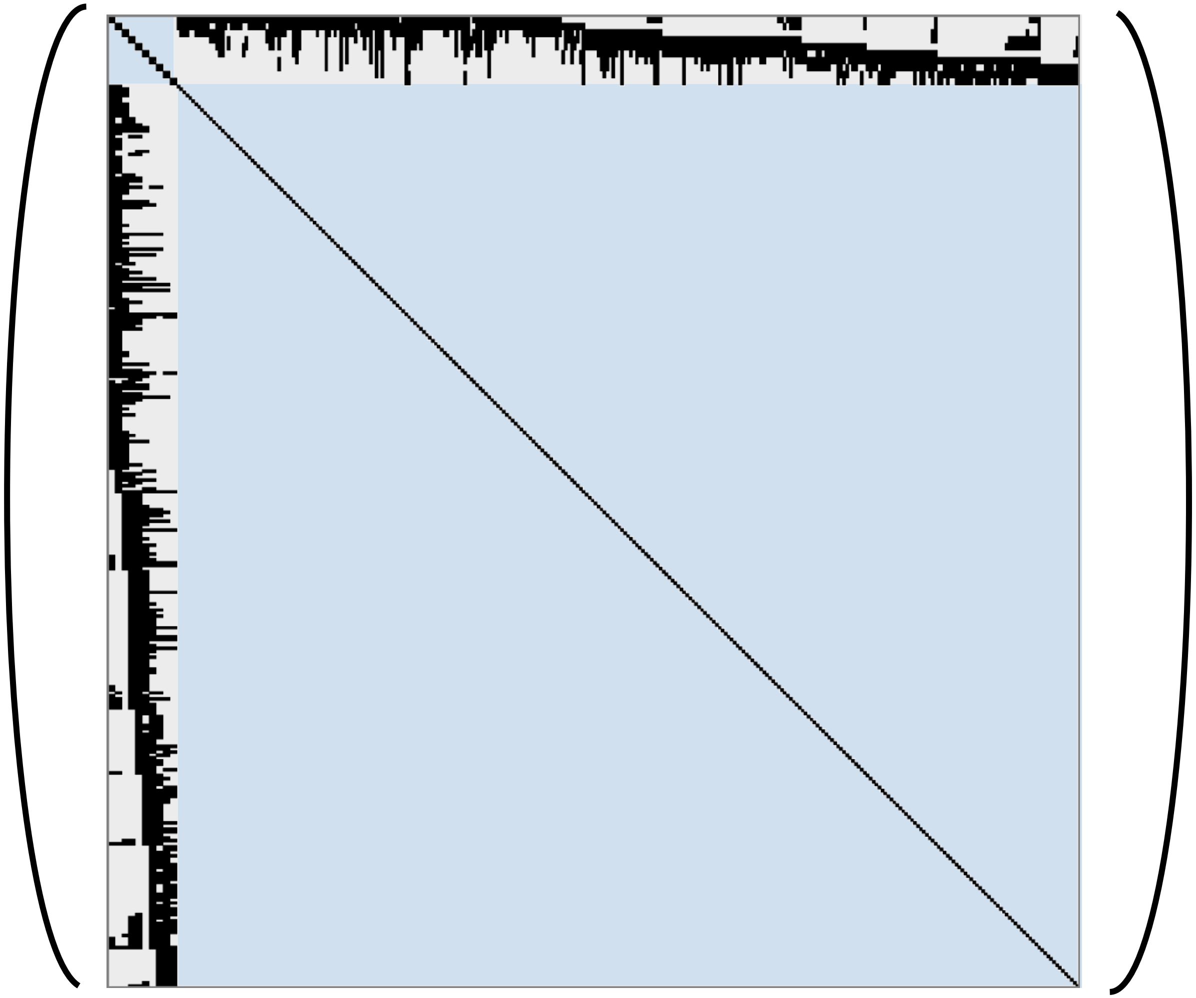
- If D is invertible, then (using Gauss elimination)

$$\begin{aligned} (A - BD^{-1}C)\mathbf{x} &= \mathbf{a} - BD^{-1}\mathbf{b} \\ \mathbf{y} &= D^{-1}(\mathbf{b} - C\mathbf{x}) \end{aligned}$$

- **Reduced complexity**, i.e., invert one $p \times p$ and $p \times p$ matrix instead of one $(p+q) \times (p+q)$ matrix

Example Hessian (Lourakis and Argyros, 2009)

$$H = $$



Two Examples

- PTAM

  G. Klein and D. Murray, Parallel Tracking and Mapping for Small AR Workspaces, International Symposium on Mixed and Augmented Reality (ISMAR), 2007

  http://www.robots.ox.ac.uk/~gk/publications/KleinMurray2007ISMAR.pdf

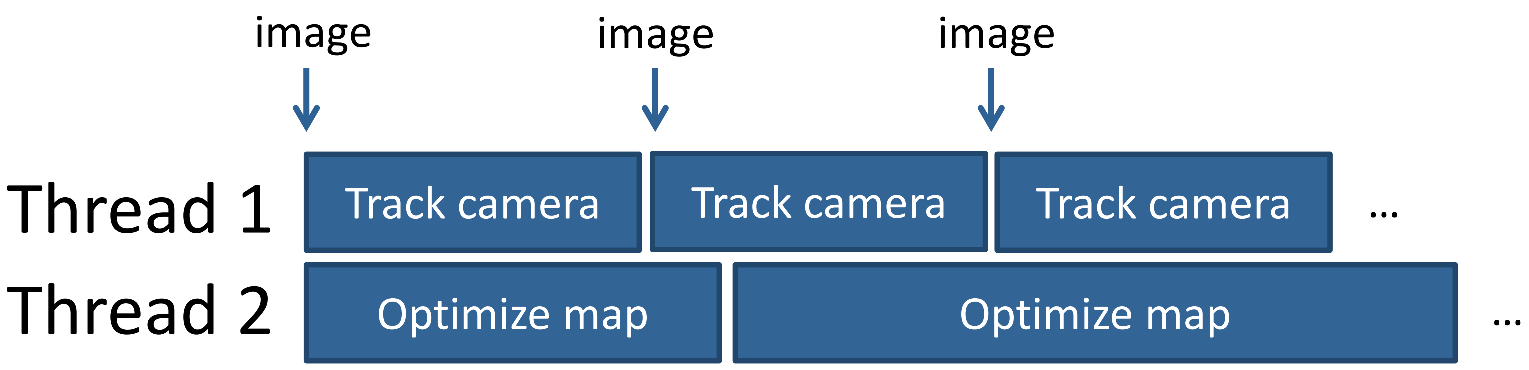
- Photo Tourism

  N. Snavely, S. M. Seitz, R. Szeliski, Photo tourism: Exploring photo collections in 3D, ACM Transactions on Graphics (SIGGRAPH), 2006

  http://phototour.cs.washington.edu/Photo_Tourism.pdf

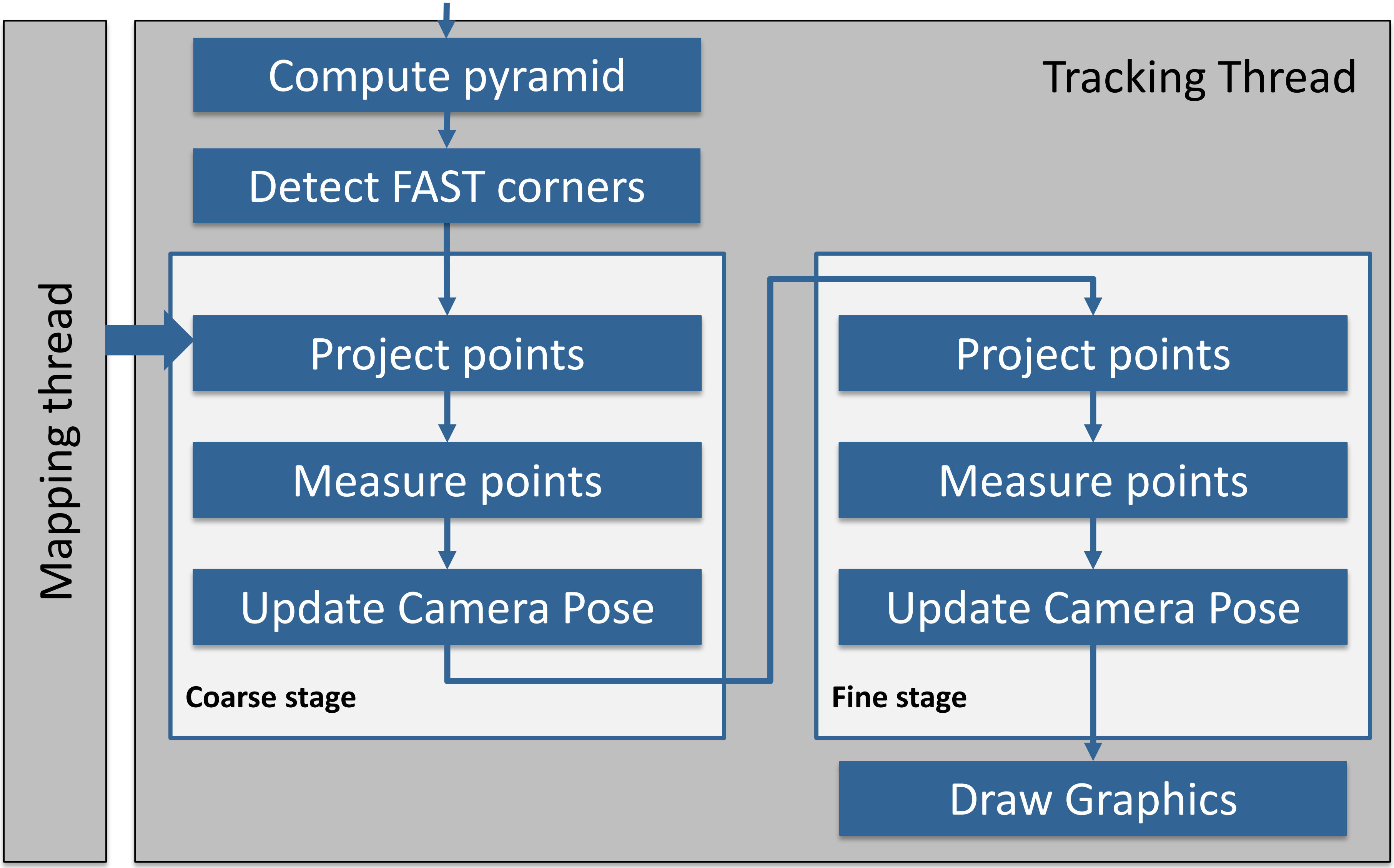
PTAM (2007)

- Architecture optimized for dual cores



- Tracking thread runs in real-time (30Hz)
- Mapping thread is not real-time

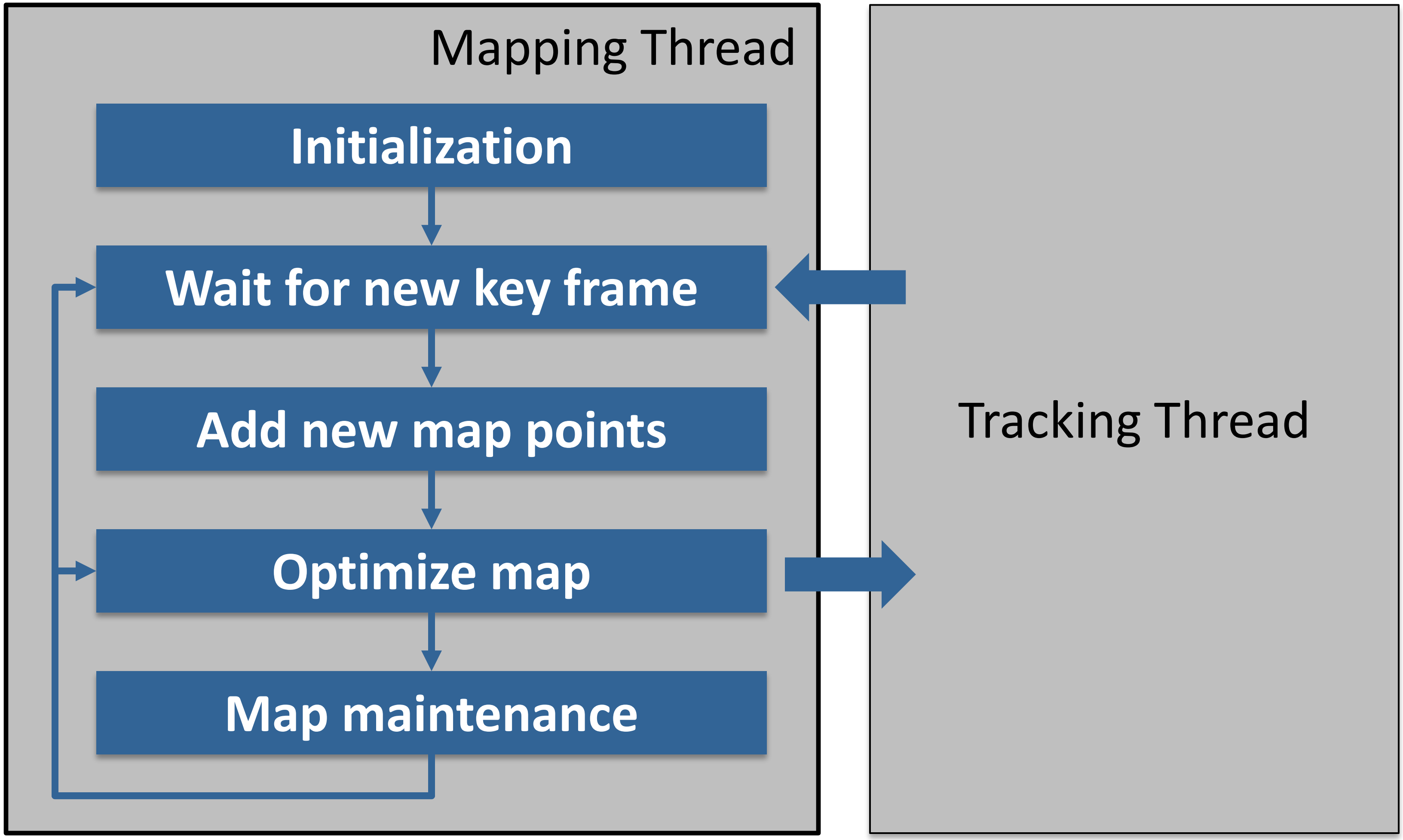
PTAM – Tracking Thread



PTAM – Feature Tracking

- Generate 8x8 matching template (warped from key frame to current pose estimate)
- Search a fixed radius around projected position
  - Using SSD
  - Only search at FAST corner points

PTAM – Mapping Thread



PTAM – Example Timings

- Tracking thread

Total	19.2 ms
Key frame preparation	2.2 ms
Feature Projection	3.5 ms
Patch search	9.8 ms
Iterative pose update	3.7 ms

- Mapping thread

Key frames	2-49	50-99	100-149
Local Bundle Adjustment	170 ms	270 ms	440 ms
Global Bundle Adjustment	380 ms	1.7 s	6.9 s

PTAM Video

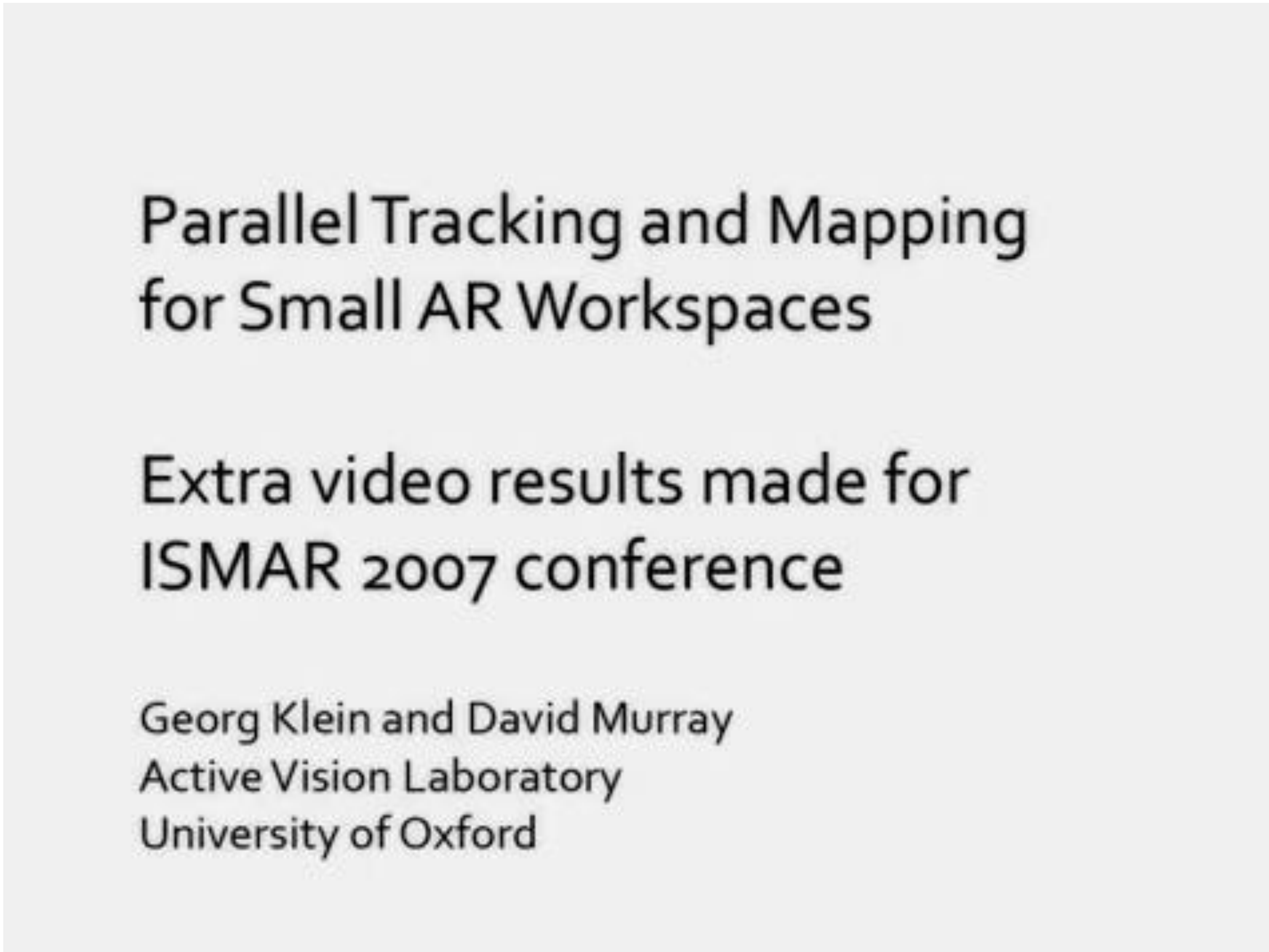
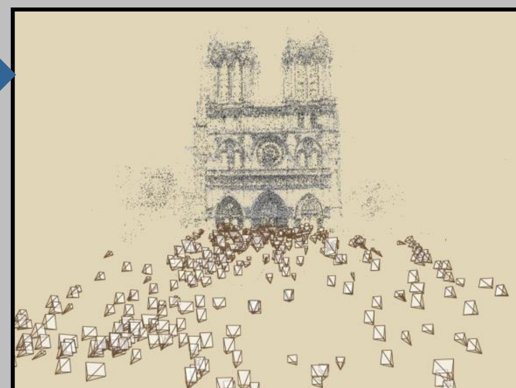

Parallel Tracking and Mapping
for Small AR Workspaces
Extra video results made for
ISMAR 2007 conference
Georg Klein and David Murray
Active Vision Laboratory
University of Oxford

Photo Tourism (2006) / Bundler

- Overview

Input Photographs
(from Flickr)

Scene Reconstruction



Relative camera positions
and orientations

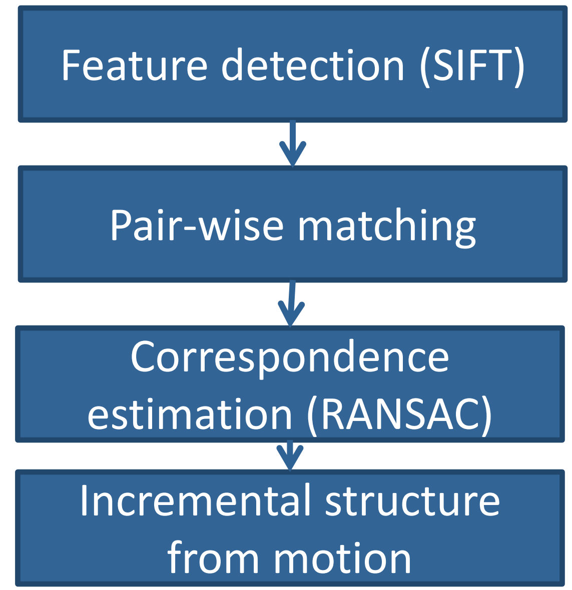
Point cloud

Sparse correspondence

Photo Explorer

Photo Tourism – Scene Reconstruction

- Processing pipeline



- Automatically estimate
  - Position, orientation and focal length of all cameras
  - 3D positions of point features

Photo Tourism – Input Images

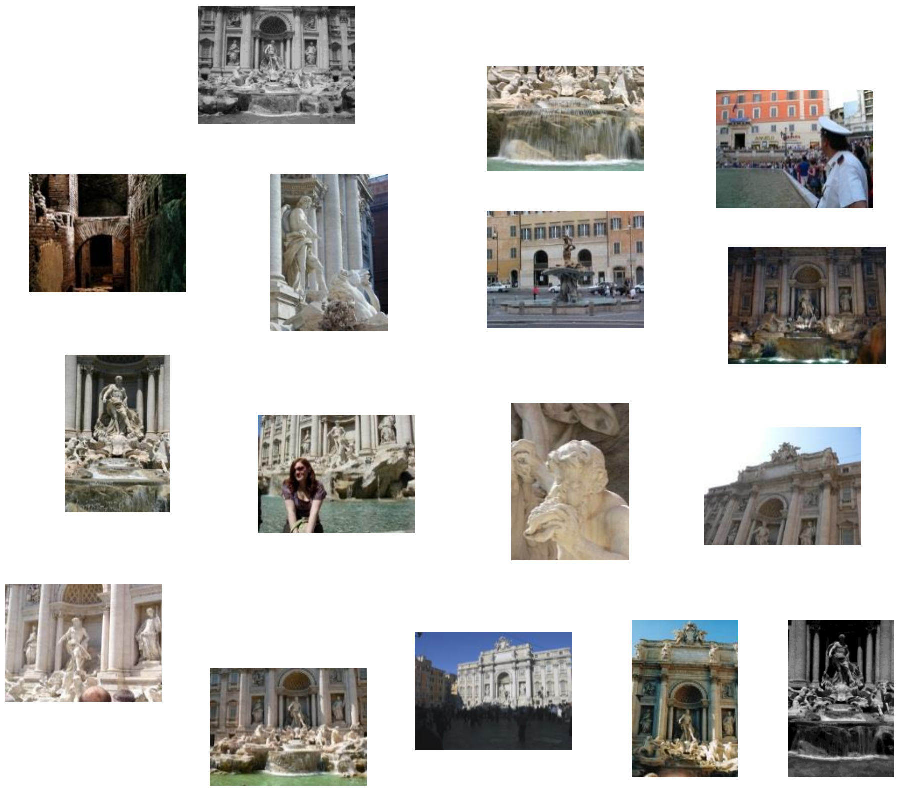


Photo Tourism – Feature Detection

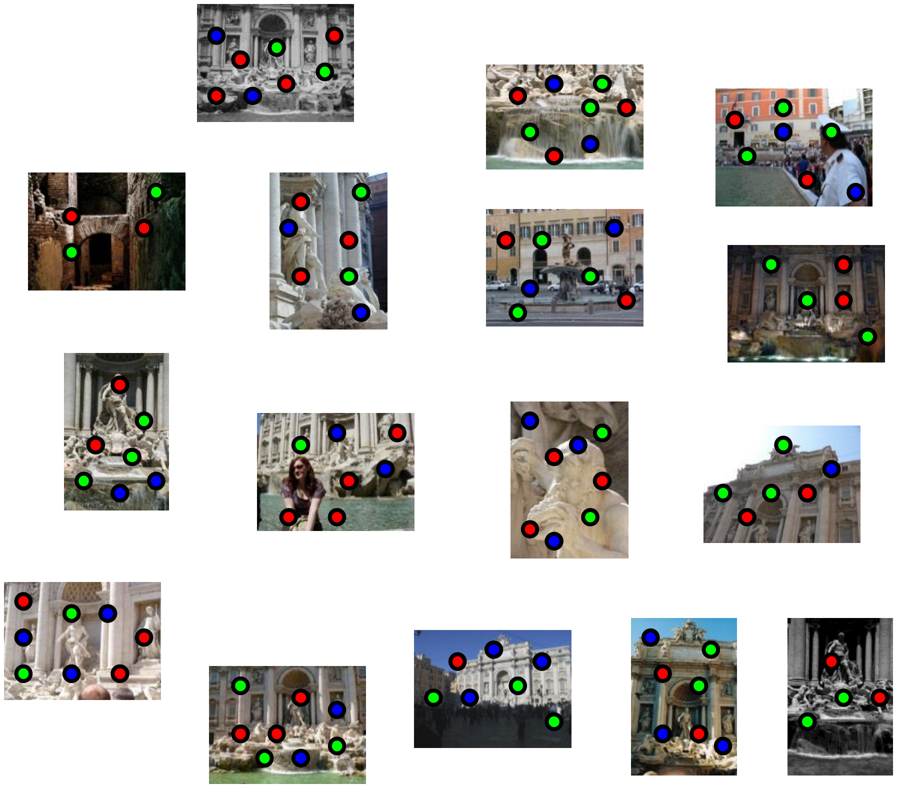
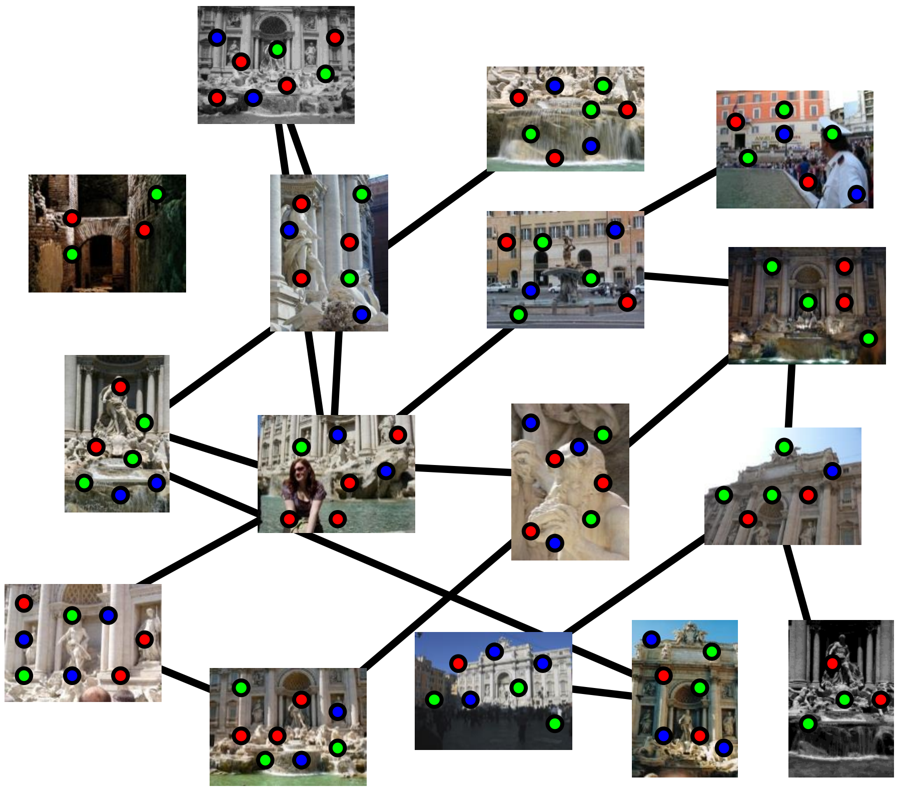


Photo Tourism – Feature Matching



Incremental Structure From Motion

- To help get good initializations, start with two images only (compute pose, triangulate points)
- Non-linear optimization
- Iteratively add more images

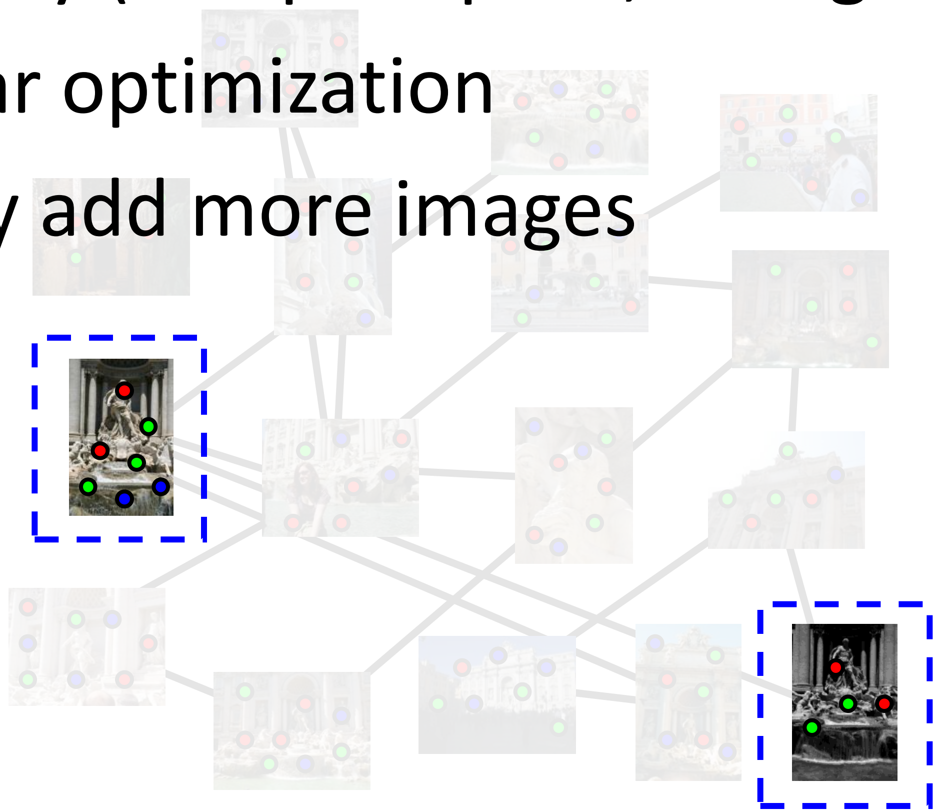
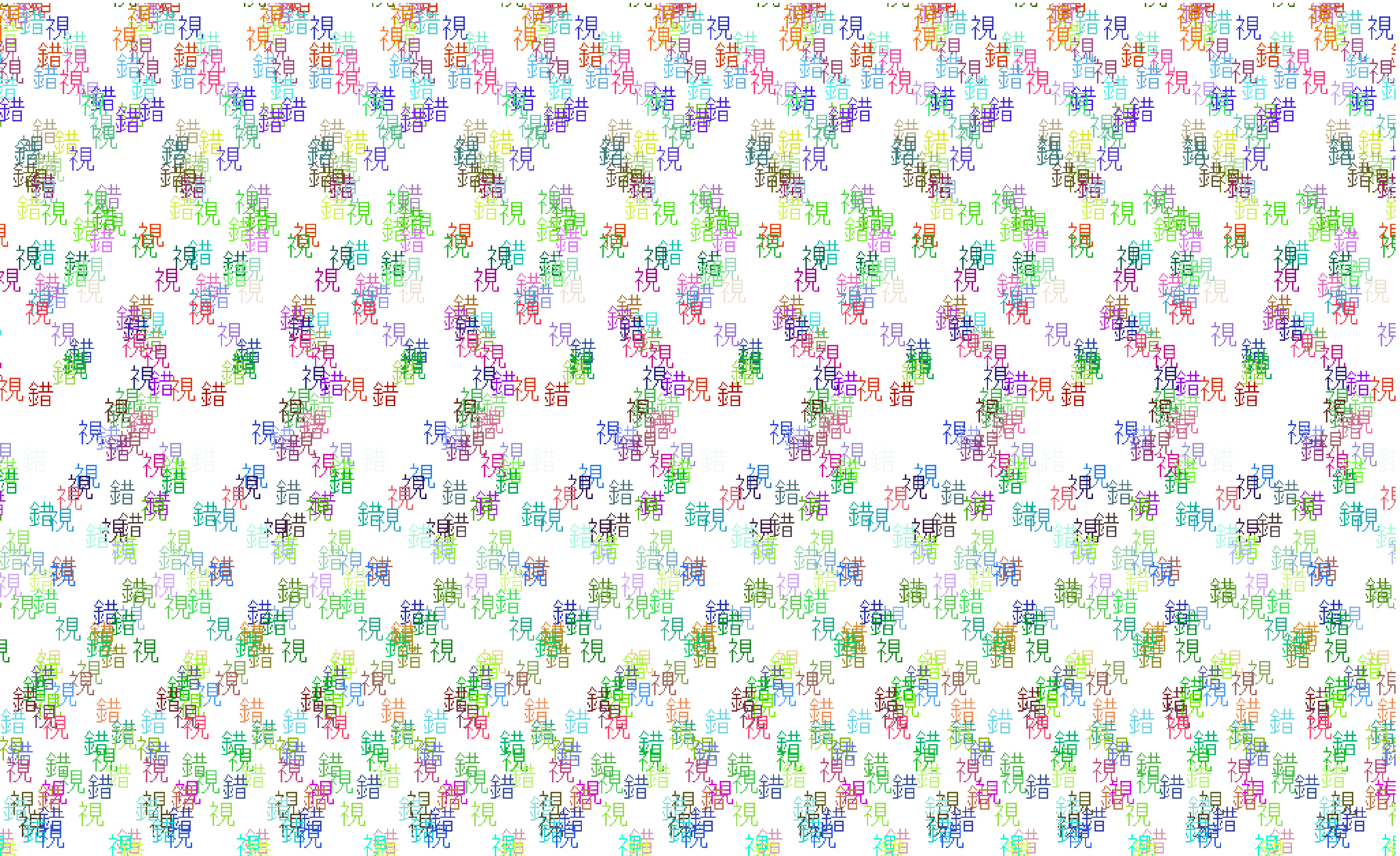


Photo Tourism – Video

From Sparse Maps to Dense Maps

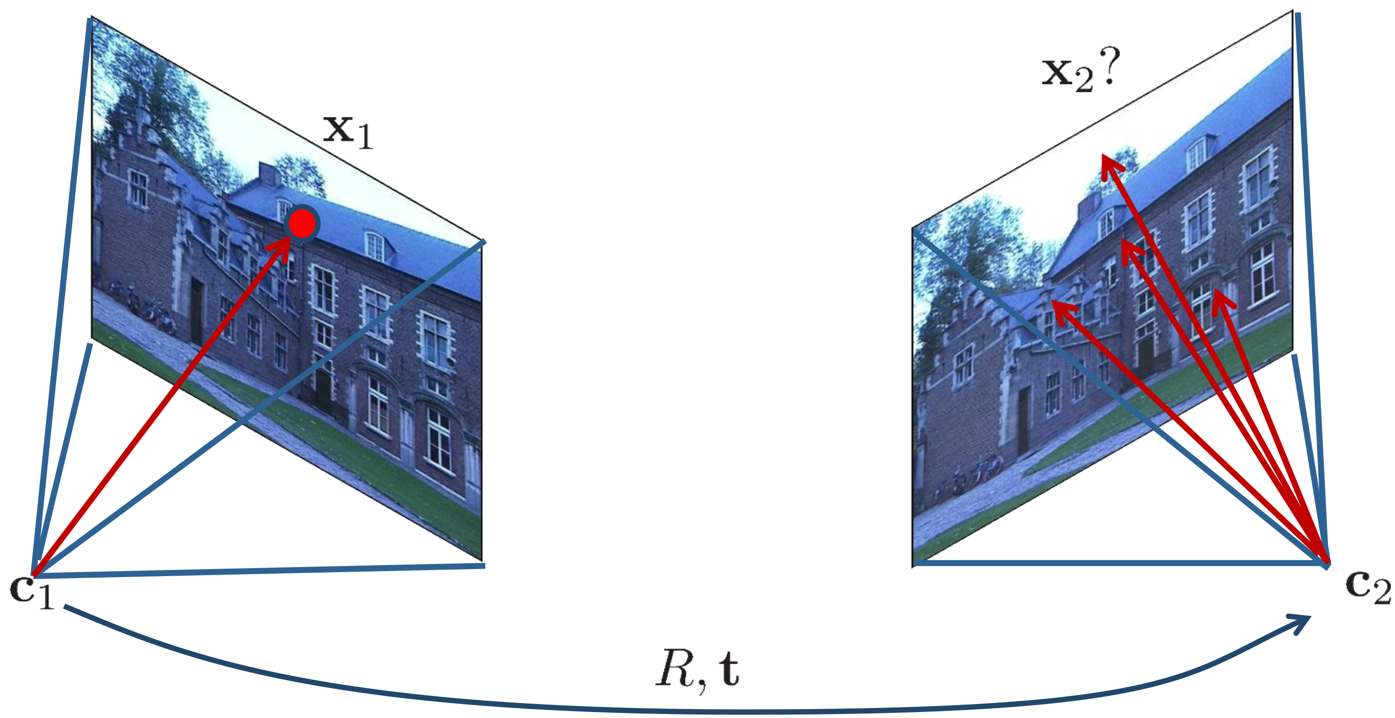
- So far, we only looked at sparse 3D maps
  - We know where the (sparse) cameras are
  - We know where the (sparse) 3D feature points are
- How can we turn these models into volumetric 3D models?

Human Stereo Vision



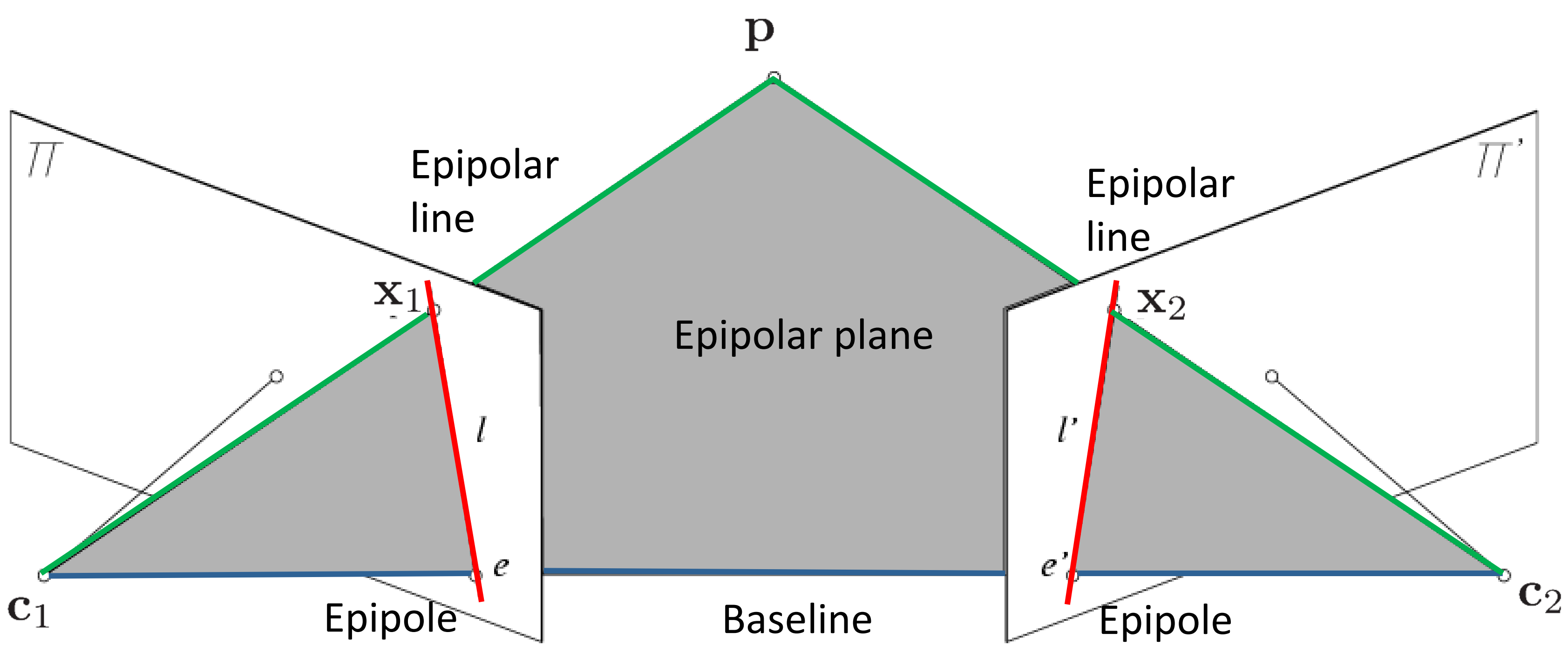
Stereo Correspondence Constraints

- Given a point in the left image, where can the corresponding point be in the right image?



Reminder: Epipolar Geometry

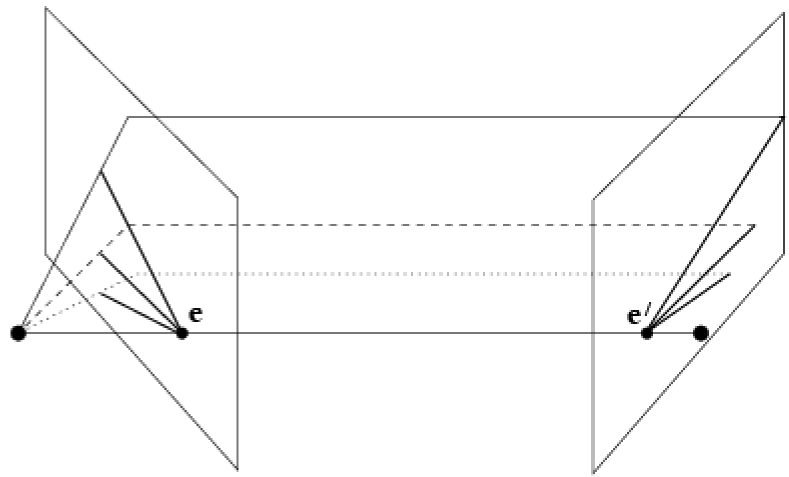
- A point in one image "generates" a line in another image (called the **epipolar line**)
- Epipolar constraint $\hat{\mathbf{x}}_2^\top E \hat{\mathbf{x}}_1 = 0$

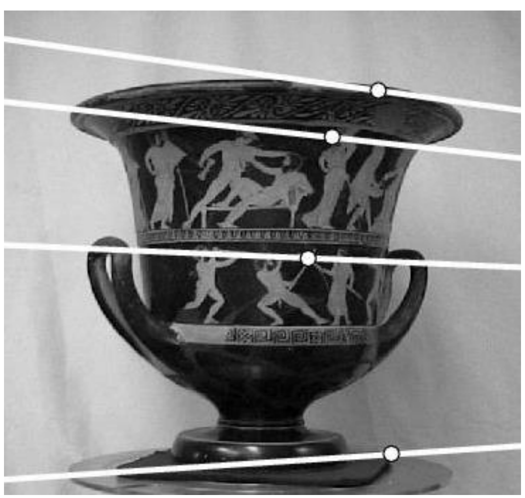


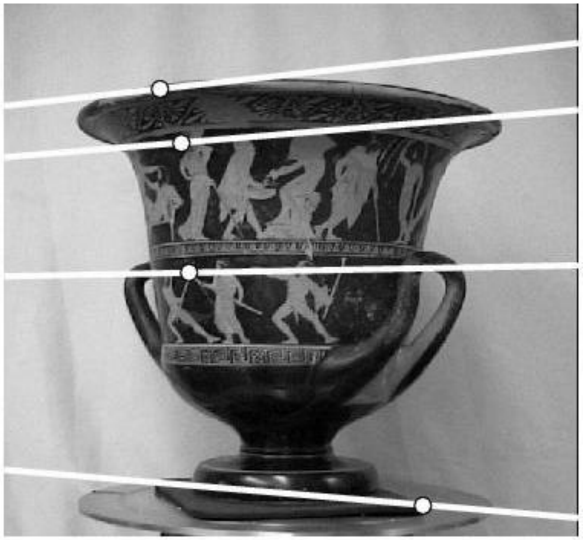
Epipolar Constraint

- This is useful because it reduces the correspondence problem to a 1D search along an epipolar line

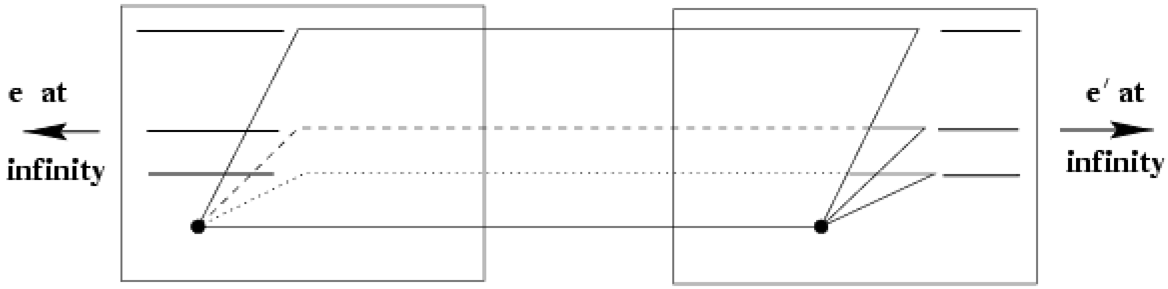
Example: Converging Cameras

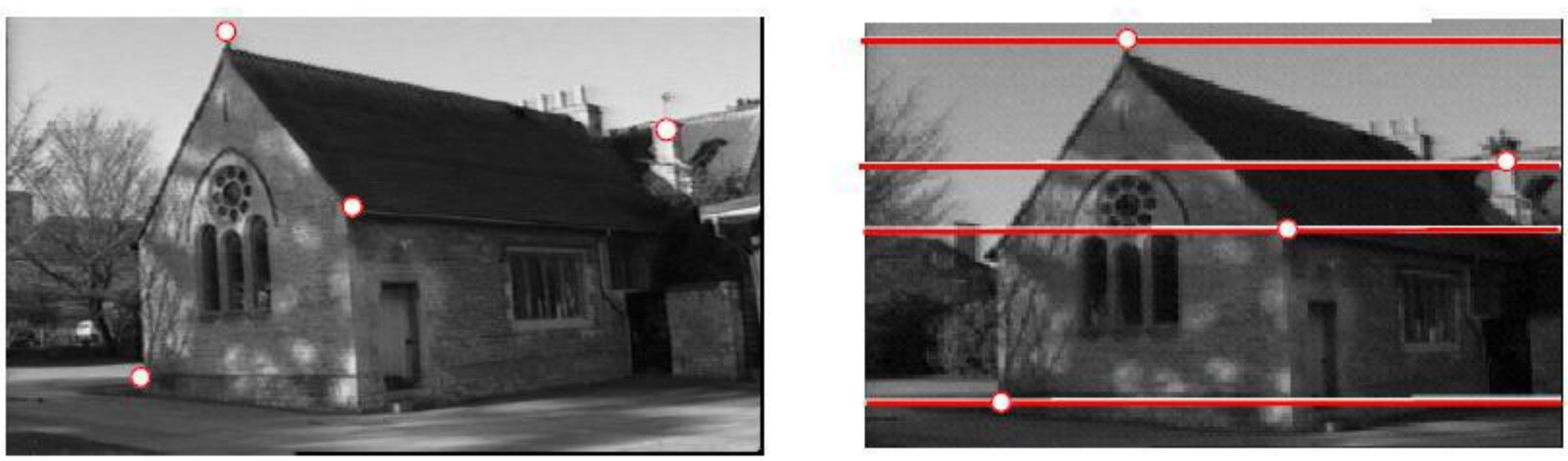






Example: Parallel Cameras



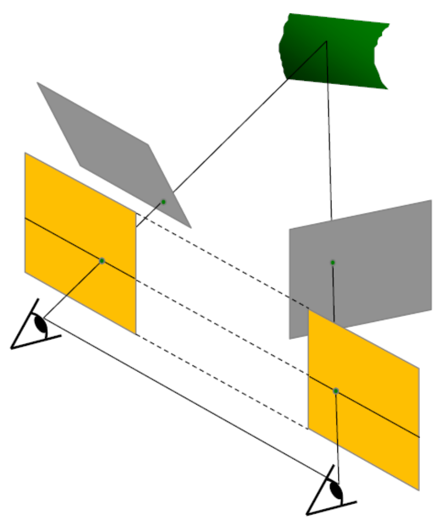


Rectification

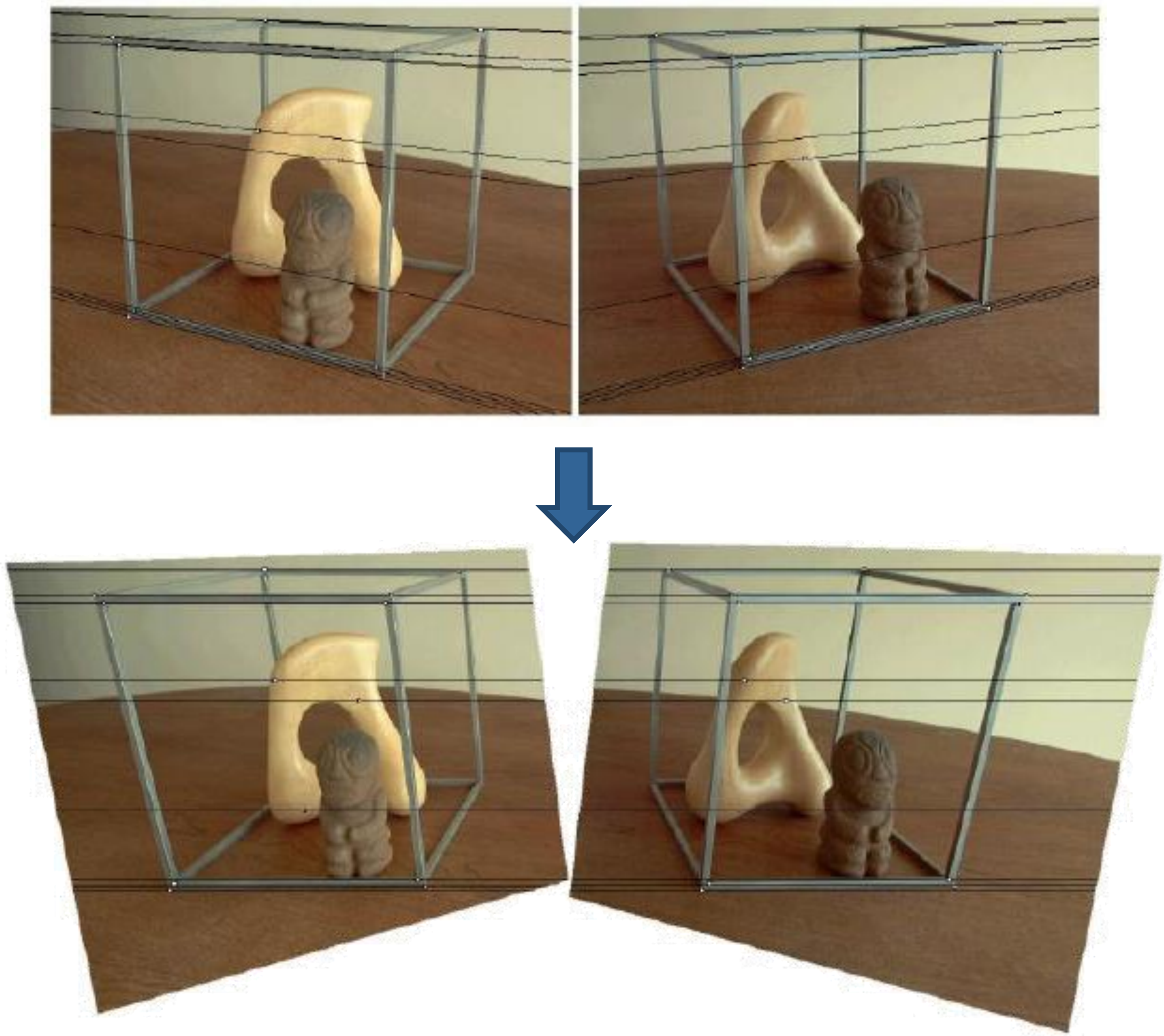
- In practice, it is convenient if the image scanlines (rows) are the epipolar lines

→ Reproject image planes onto a common plane parallel to the baseline (two 3x3 homographies)

- Afterwards pixel motion is horizontal

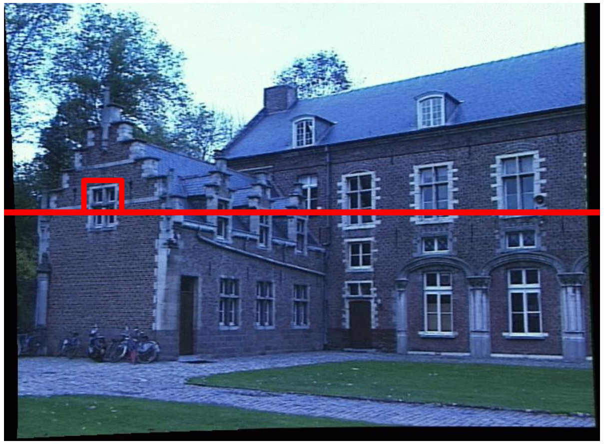


Example: Rectification



Basic Stereo Algorithm

- For each pixel in the left image
  - Compare with every pixel on the same epipolar line in the right image
  - Pick pixel with minimum matching cost (noisy)
  - Better: match small blocks/patches (SSD, SAD, NCC)



left image

right image

Block Matching Algorithm

Input: Two images and camera calibrations

Output: Disparity (or depth) image

Algorithm:

1. Geometry correction (undistortion and rectification)
2. Matching cost computation along search window
3. Extrema extraction (at sub-pixel accuracy)
4. Post-filtering (clean up noise)

Example

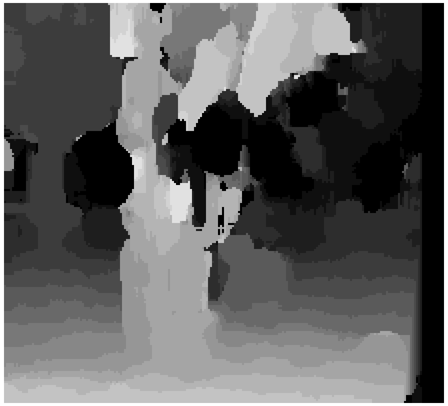
- Input


- Output

What is the Influence of the Block Size?

- Common choices are 5x5 .. 11x11
- Smaller neighborhood: more details
- Larger neighborhood: less noise
- Suppress pixels with low confidence (e.g., check ratio best match vs. 2^{nd} best match)

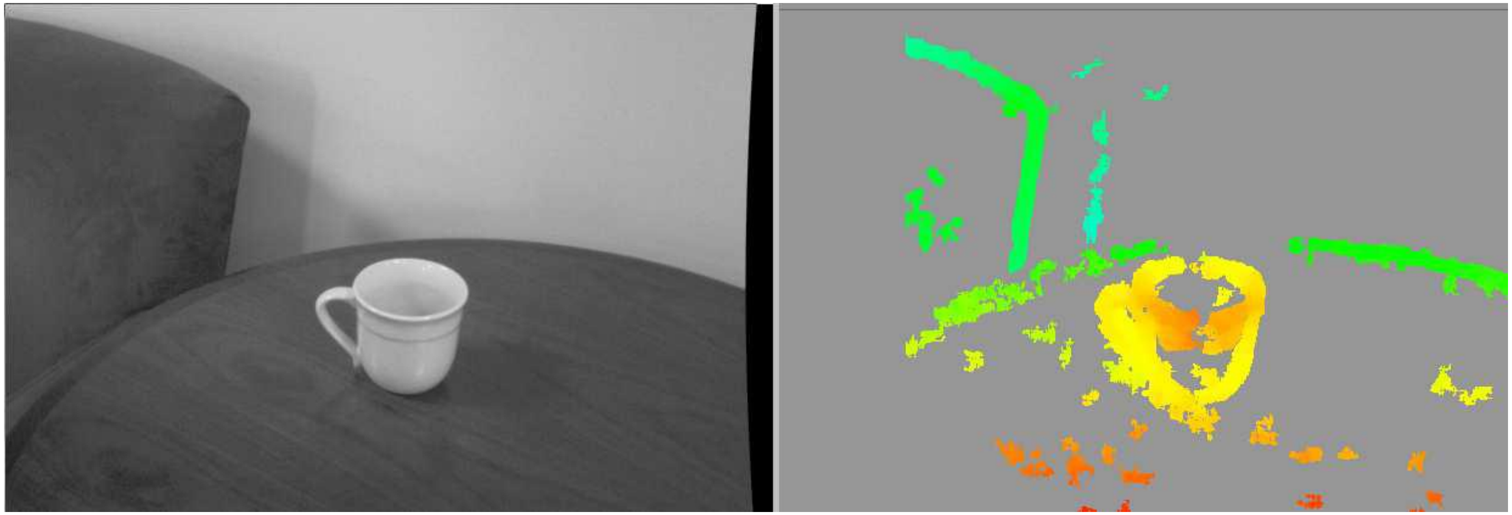
3x3



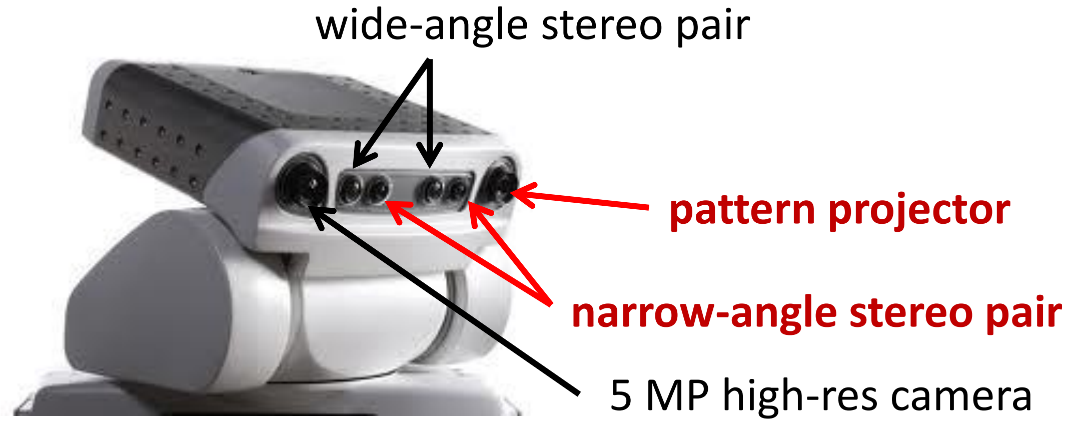
20x20

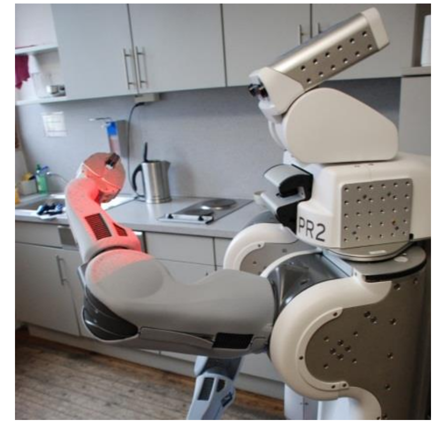
Problems with Stereo

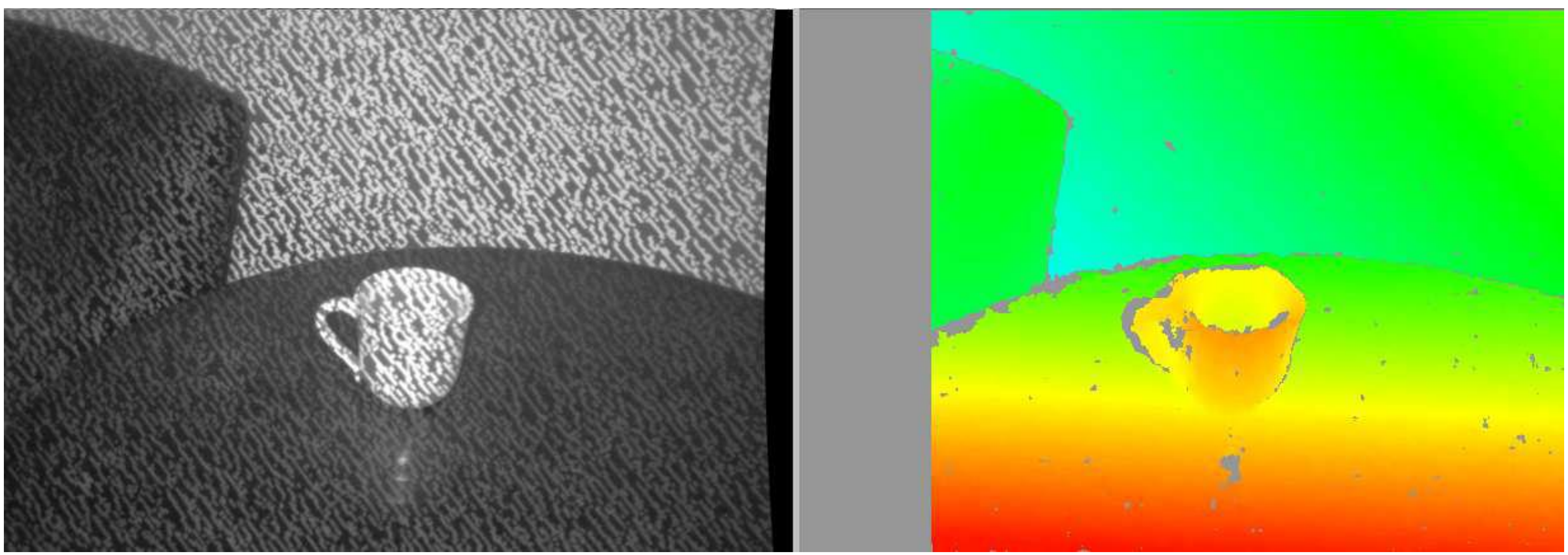
- Block matching typically fails in regions with low texture
  - Global optimization/regularization (speciality of our research group)
  - Additional texture projection



Example: PR2 Robot with Projected Texture Stereo

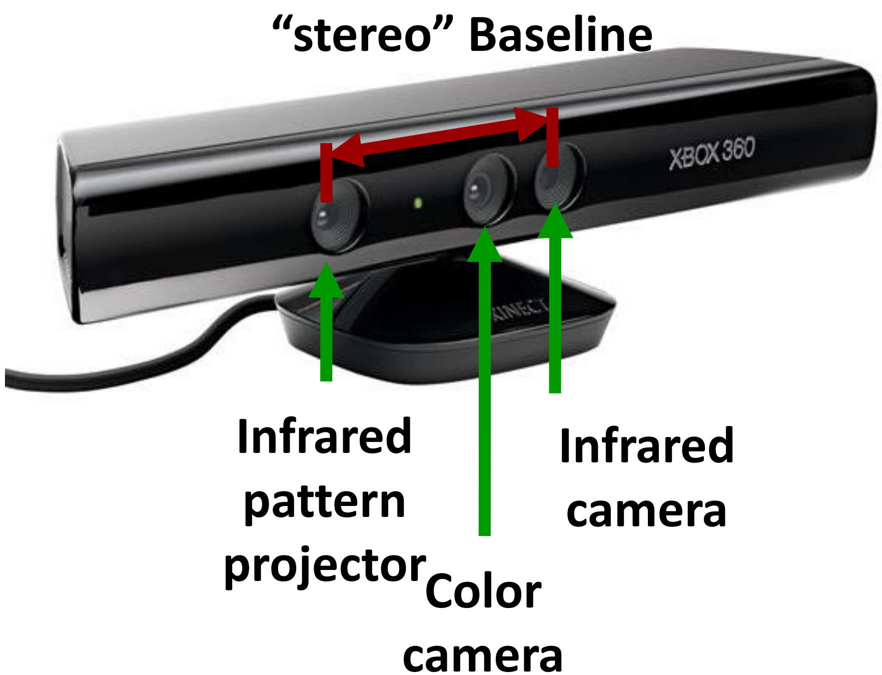






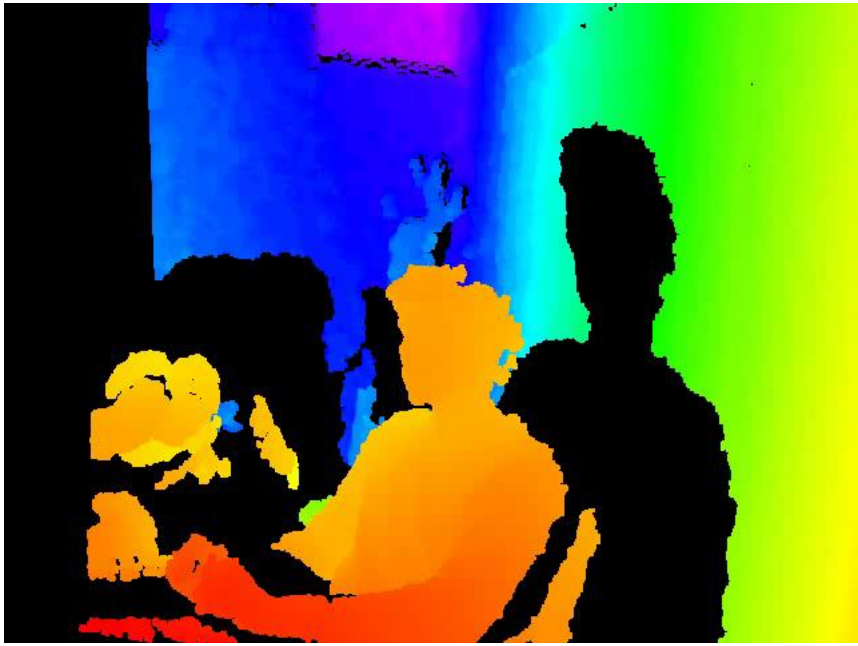
Sensor Principle of Kinect

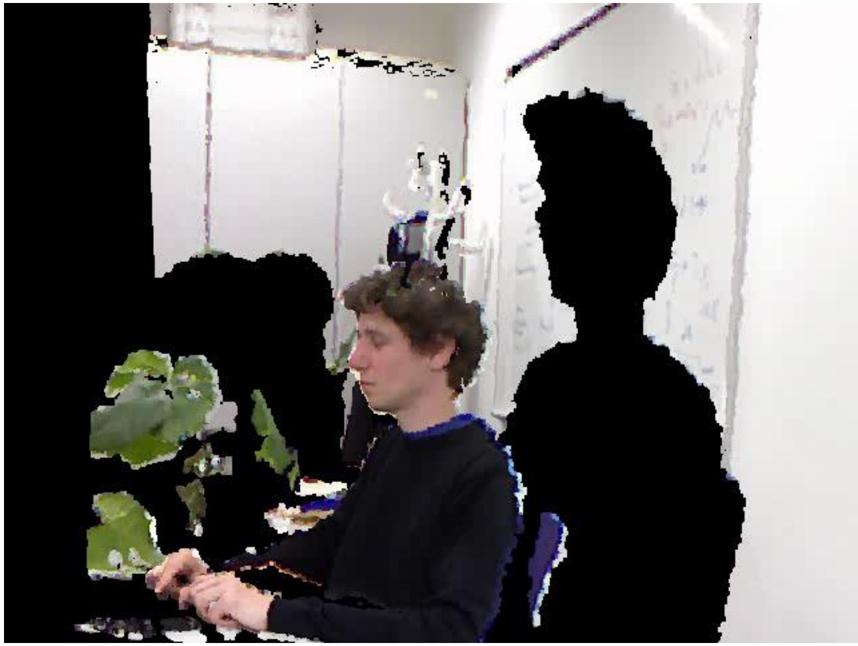
- Kinect projects a diffraction pattern (speckles) in near-infrared light
- CMOS IR camera observes the scene



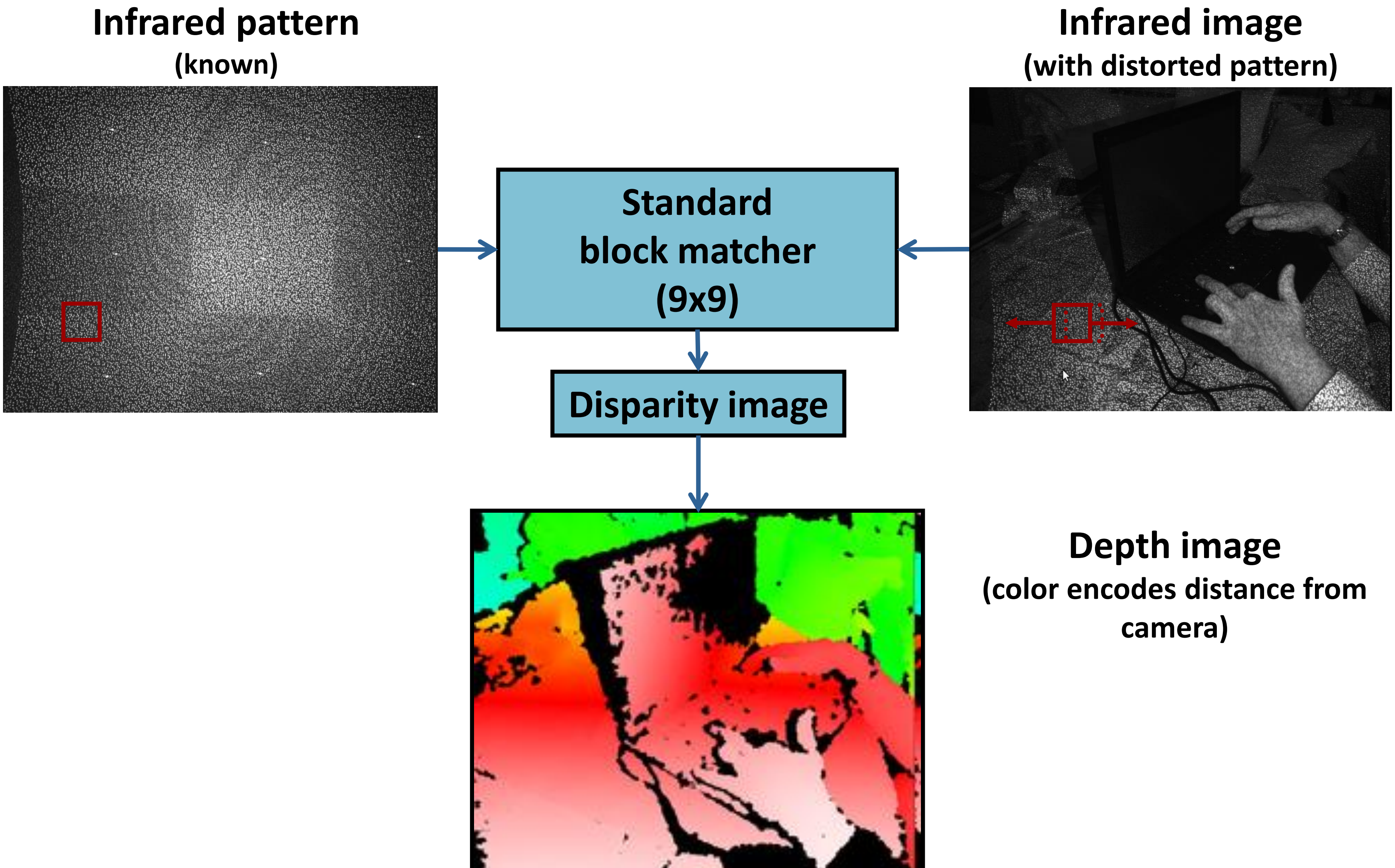
Example Data

- Kinect provides color (RGB) and depth (D) video
- This allows for novel approaches for (robot) perception





Sensor Principle of Kinect



Sensor Principle of Kinect

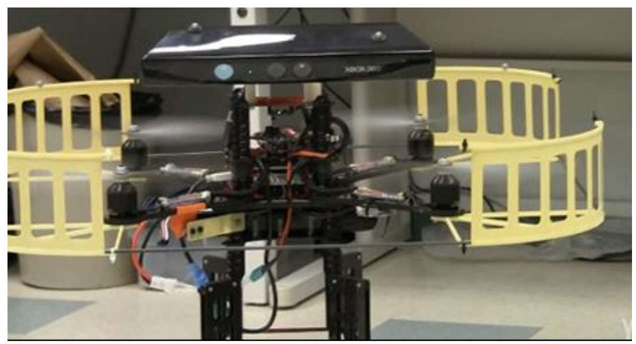
- Pattern is memorized at a known depth
- For each pixel in the IR image
  - Extract 9x9 template from memorized pattern
  - Correlate with current IR image over 64 pixels and search for the maximum
  - Interpolate maximum to obtain sub-pixel accuracy (1/8 pixel)
  - Calculate depth by triangulation

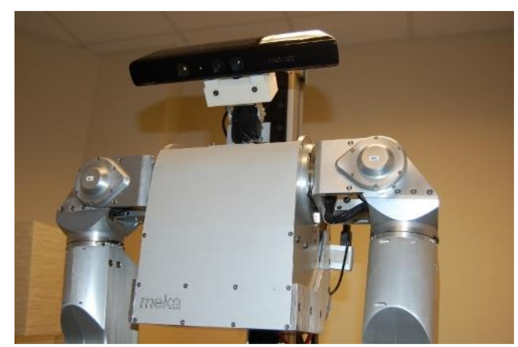
Technical Specs

- Infrared camera has 640x480 @ 30 Hz
  - Depth correlation runs on FPGA
  - 11-bit depth image
  - 0.8m – 5m range
  - Depth sensing does not work in direct sunlight (why?)
- RGB camera has 640x480 @ 30 Hz

Impact of the Kinect Sensor

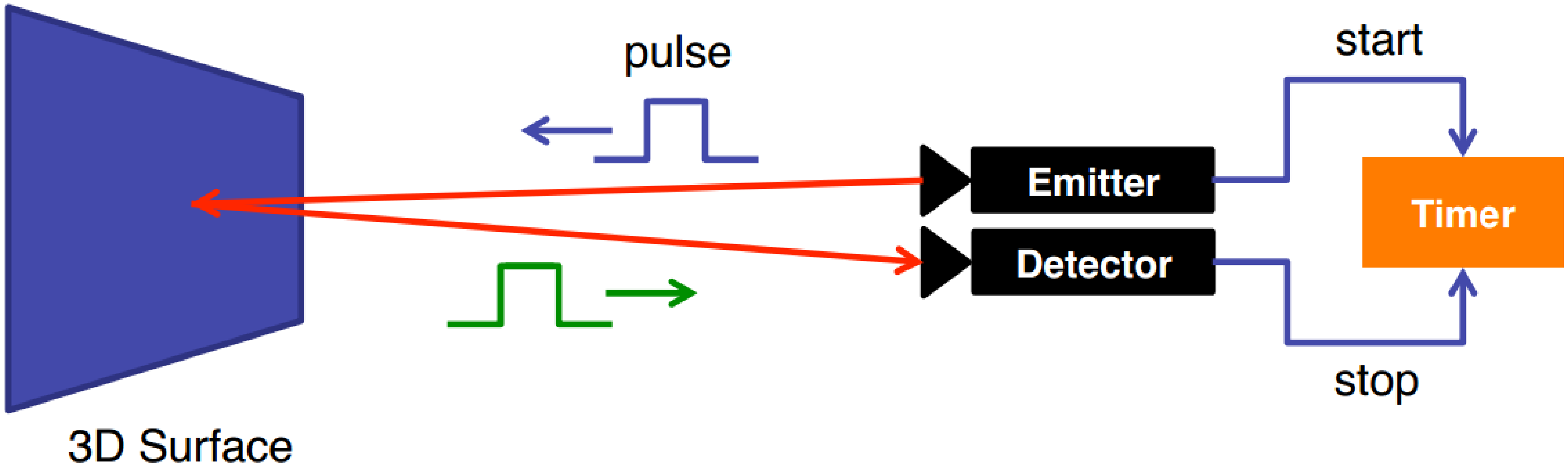
- Sold >18M units
- Has become a "standard" sensor in robotics
- Several variants (Asus Xtion Pro, PrimeSense)

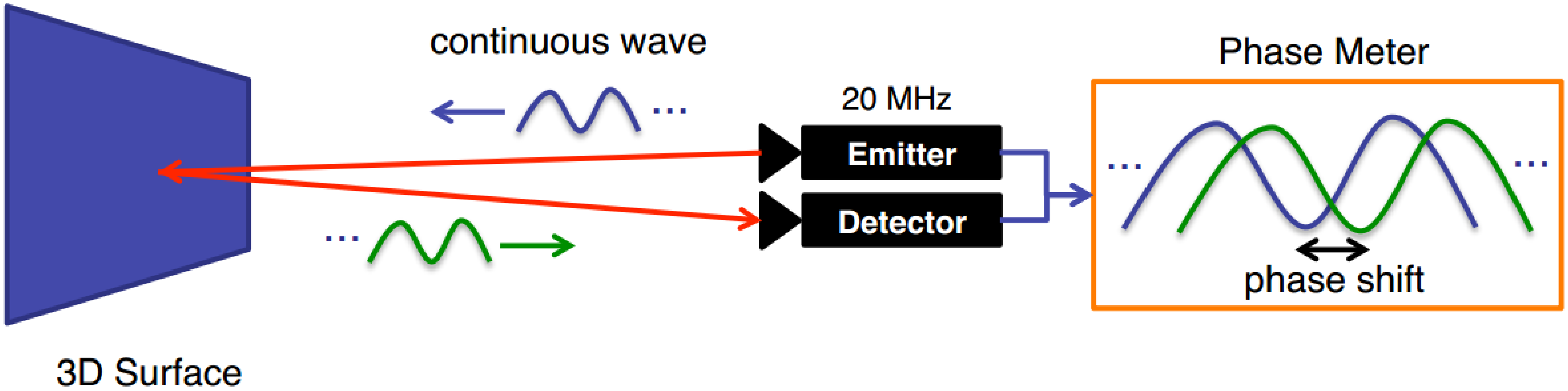
Time-of-Flight Cameras

- Direct time-of-flight measurement
  - Emit short light pulse (flash)
  - Every pixel counts time until signal is detected



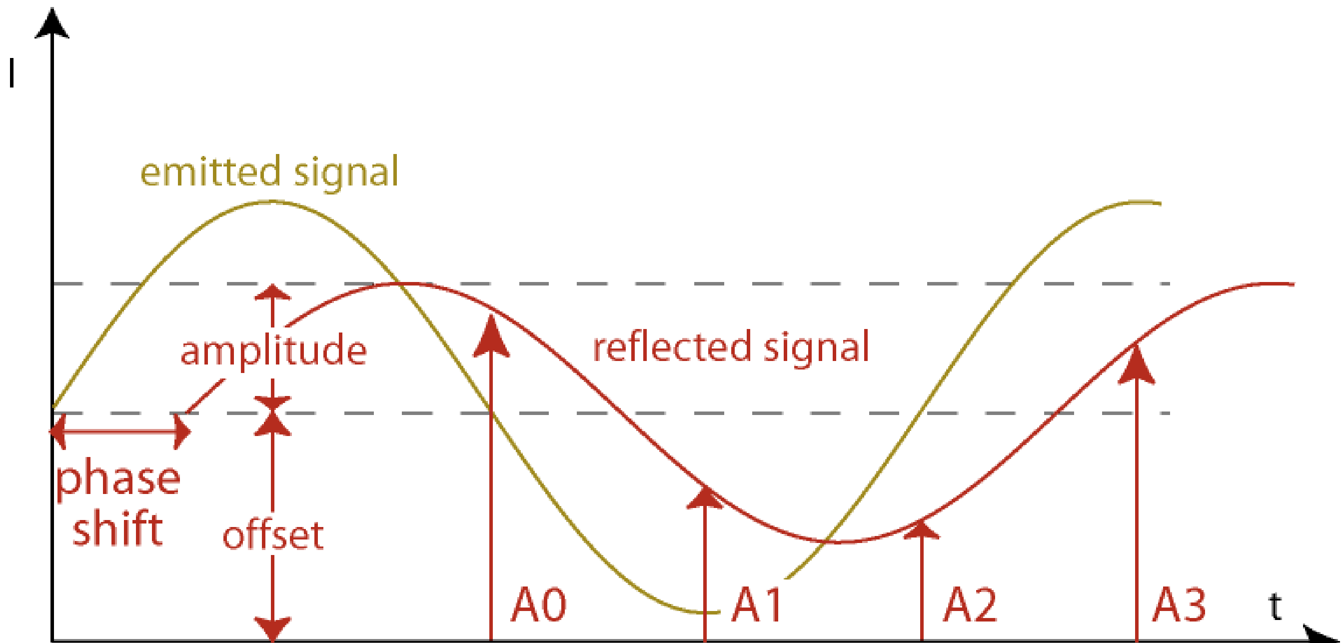
Time-of-Flight Cameras

- Indirect measurement (phase shift)
  - Emit modulated light (e.g., at 30 MHz → 10m wave length)
  - Every pixel measures the phase shift



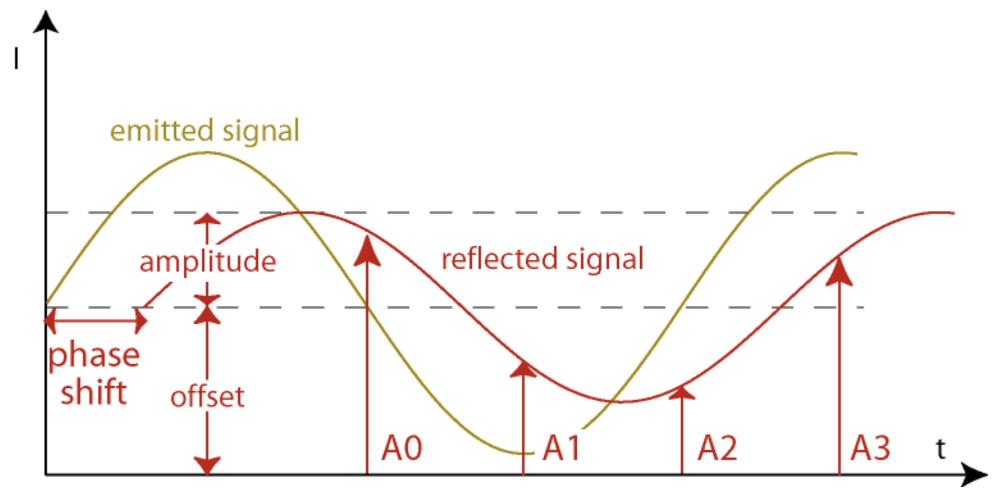
Decoding the Phase

- Take four intensity measurements at 90° angle
- Integrate over several waves to reduce noise
- Decode amplitude, offset, phase shift



Decoding the Phase

- Amplitude (=quality) $A = \frac{1}{2}\sqrt{(A_3 - A_1)^2 + (A_2 - A_0)^2}$
- Offset (=intensity) $B = A_0 + A_1 + A_2 + A_3$
- Phase shift (=distance) $\phi = \arctan \frac{A_3 - A_1}{A_2 - A_0}$



Commercial Time-Of-Flight Sensors

- Mesa SwissRanger ($4300), 176x144
- PMDTec: 200x200
- Intel Creative Camera ($150), 320x240
- Xbox One Kinect, 512x424, 13-bit depth

Xbox Kinect One

Intel Perceptual Computing Challenge

- https://perceptualchallenge.intel.com/
- API for raw data + hand gesture recognition
- This Saturday (22.6.2013): Hacknight
  - Every participating team gets a sensor for free
  - WERK1, Kultfabrik, Grafinger Str. 6, 81671 München
  - Sign up on Facebook page: http://www.facebook.com/groups/154028434769715/
- Cash Prizes for the Hacknight
  - €2.500 for the Best app
  - €1.000 for the Most Innovative app
  - €1.000 for the Best User Experience app

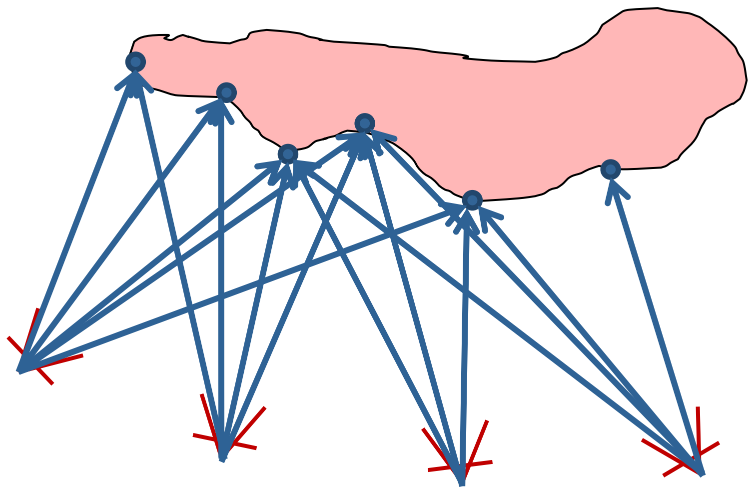
Project with Ardrones?

Agenda for Today

- Bundle adjustment ✓
- Depth cameras ✓
- Occupancy grid maps
- Signed distance functions

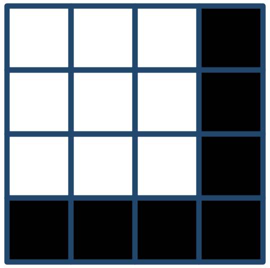
Mapping and 3D Reconstruction

- So far: We have camera poses and 3D points



- Robot needs a map for:
  - Path planning and collision-free navigation
  - Exploration of unmapped areas
- How can we estimate such a map?

Occupancy Grid



Idea:

- Represent the map $\mathbf{m}$ using a grid
- Each cell is either free or occupied

$$\mathbf{m} = (m_1, \ldots, m_n) \in \{\text{empty}, \text{occ}\}^n$$

- Robot maintains a belief $\mathrm{Bel}(\mathbf{m})$ on map state

Goal: Estimate the belief from sensor observations

$$\mathrm{Bel}(\mathbf{m}) = P(\mathbf{m} \mid \mathbf{z}_1, \ldots, \mathbf{z}_t)$$

Occupancy Grid - Assumptions

- Map is static
- Cells have binary state (empty or occupied)
- All cells are independent of each other

- As a result, each cell m_i can be estimated independently from the sensor observations
- Will also drop index i (for the moment)

Mapping

- **Goal:** Estimate

$$\mathrm{Bel}(m) = P(m \mid z_1, \ldots, z_n)$$

- How can this be computed?

Binary Bayes Filter

- **Goal:** Estimate

$$\mathrm{Bel}(m) = P(m \mid z_1, \ldots, z_n)$$

- How can this be computed?
- Using the (binary) Bayes Filter from Lecture 3

$$P(m \mid z_{1:t}) =$$
$$\left(1 + \frac{1 - P(m \mid z_t)}{P(m \mid z_t)} \frac{1 - P(m \mid z_{1:t-1})}{P(m \mid z_{1:t-1})} \frac{P(m)}{1 - P(m)}\right)^{-1}$$

Binary Bayes Filter

- **Prior probability** that cell is occupied $P(m)$ (often 0.5)
- **Inverse sensor model** $P(m \mid z_t)$ is specific to the sensor used for mapping
- The **log-odds representation** can be used to increase speed and numerical stability

$$L(x) := \log \frac{p(x)}{p(\neg x)} = \log \frac{p(x)}{1 - p(x)}$$

Binary Bayes Filter using Log-Odds

- In each time step, compute

$$L(m \mid z_{1:t}) = \overset{\text{previous belief}}{L(m \mid z_{1:t-1})} + \overset{\substack{\text{inverse} \\ \text{sensor model}}}{L(m \mid z_t)} + \overset{\text{map prior}}{L(m)}$$

- When needed, compute current belief as

$$\mathrm{Bel}_t(m) = 1 - \frac{1}{1 + \exp L(m \mid z_{1:t})}$$

Clamping Update Policy

- Often, the world is not “fully” static
- Consider an appearing/disappearing obstacle
- To change the state of a cell, the filter needs as many positive (negative) observations
- **Idea:** Clamp the beliefs to min/max values

$$L'(m \mid z_{1:t}) = \max(\min(L(m \mid z_{1:t}), l_{\max}), l_{\min})$$

Sensor Model

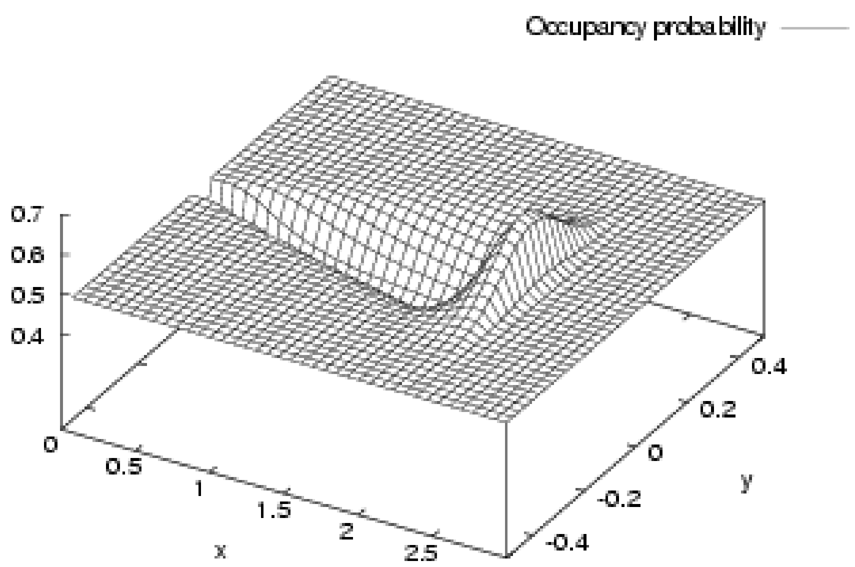
- For the Bayes filter, we need the inverse sensor model

$$p(m \mid z)$$

- Let's consider an ultrasound sensor
  - Located at (0,0)
  - Measures distance of 2.5m
  - How does the inverse sensor model look like?

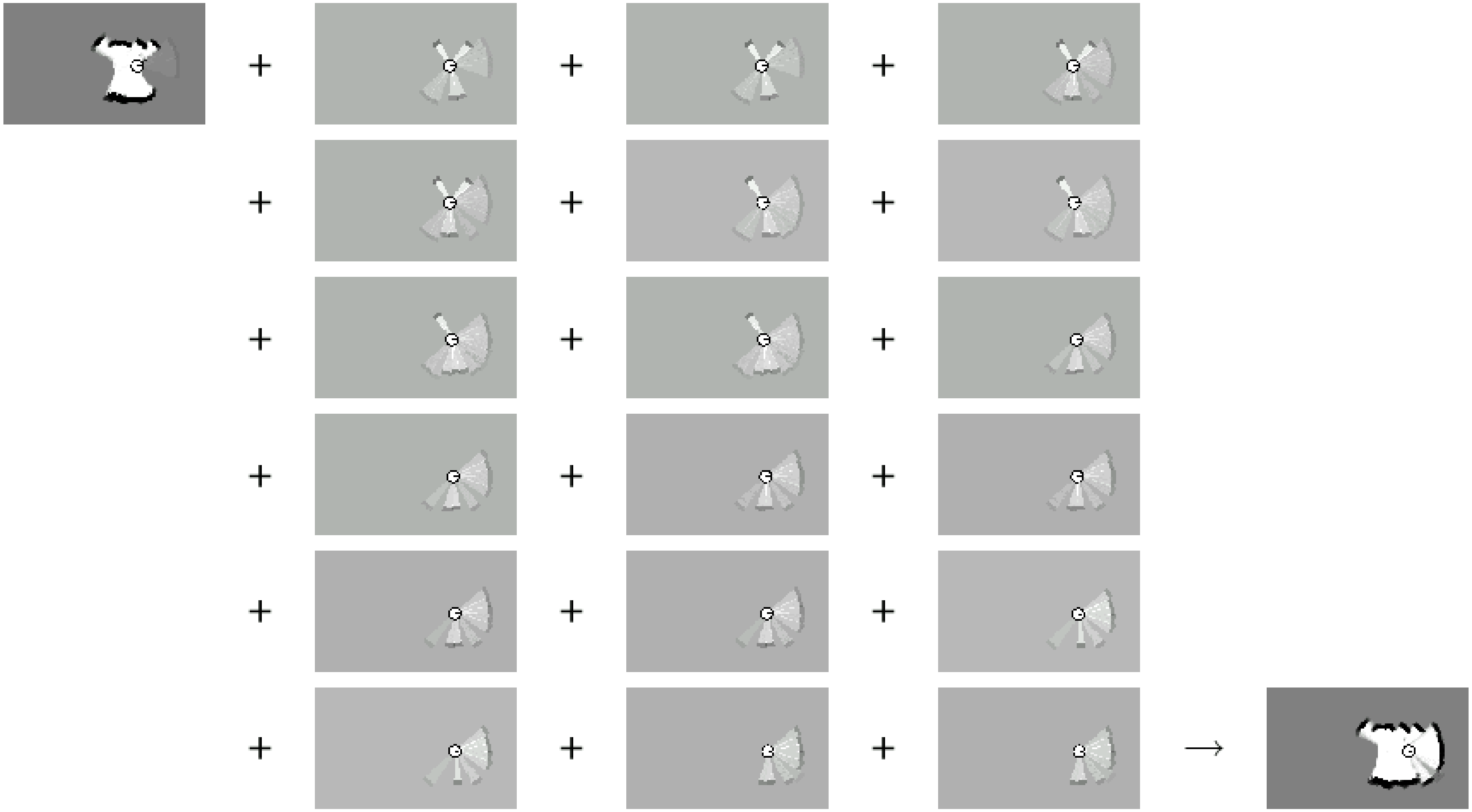
Typical Sensor Model for Ultrasound

- Combination of a linear function (in x-direction) and a Gaussian (in y-direction)

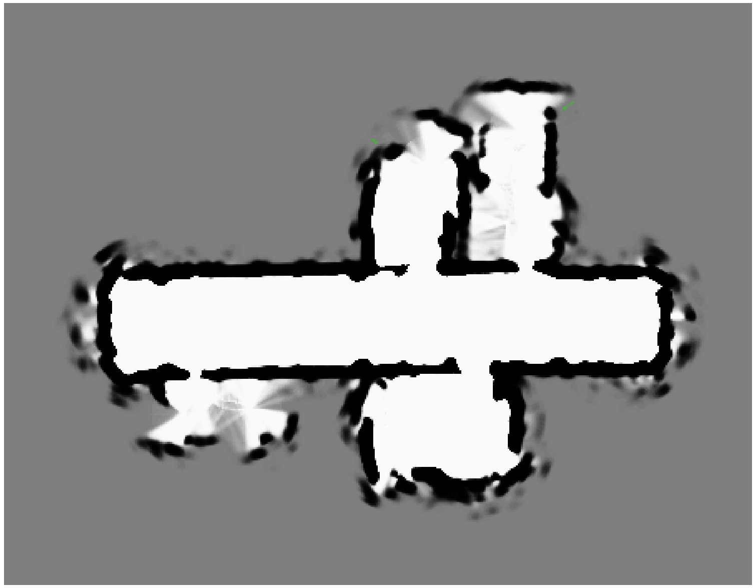


- Question: What about a laser scanner?

Example: Updating the Occupancy Grid



Resulting Map



Note: The maximum likelihood map is obtained by clipping the occupancy grid map at a threshold of 0.5

Memory Consumption

- Consider we want to map a building with 40x40m at a resolution of 0.05cm
- How much memory do we need?

Memory Consumption

- Consider we want to map a building with 40x40m at a resolution of 0.05cm
- How much memory do we need?

$$\left(\frac{40}{0.05}\right)^2 = 640.000 \text{ cells} = 4.88\text{mb}$$

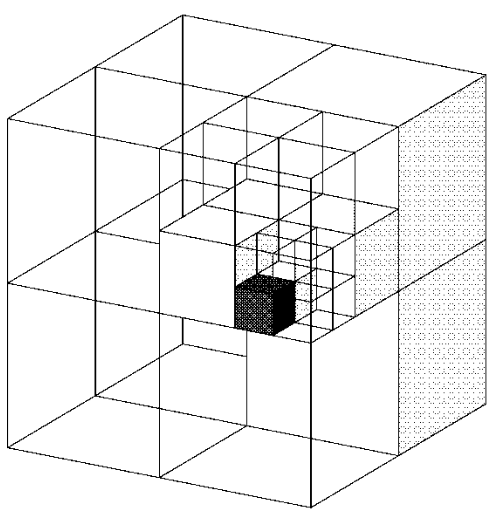
- And for 3D?

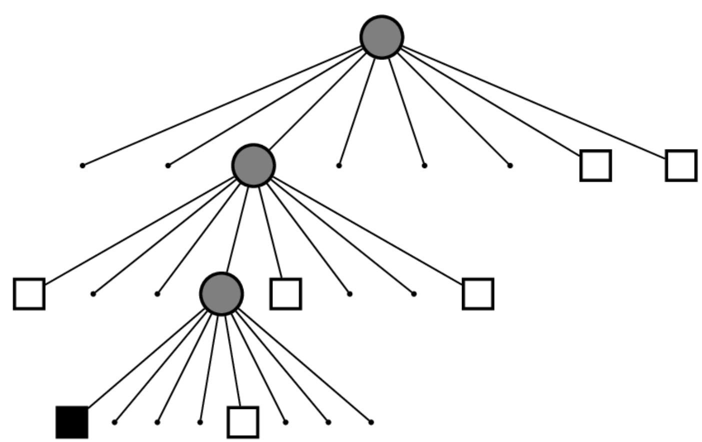
$$\left(\frac{40}{0.05}\right)^3 = 512.000.000 \text{ cells} = 3.8\text{gb}$$

- And what about a whole city?

Map Representation by Octtrees

- Tree-based data structure
- Recursive subdivision of space into octants
- Volumes can be allocated as needed
- Multi-resolution

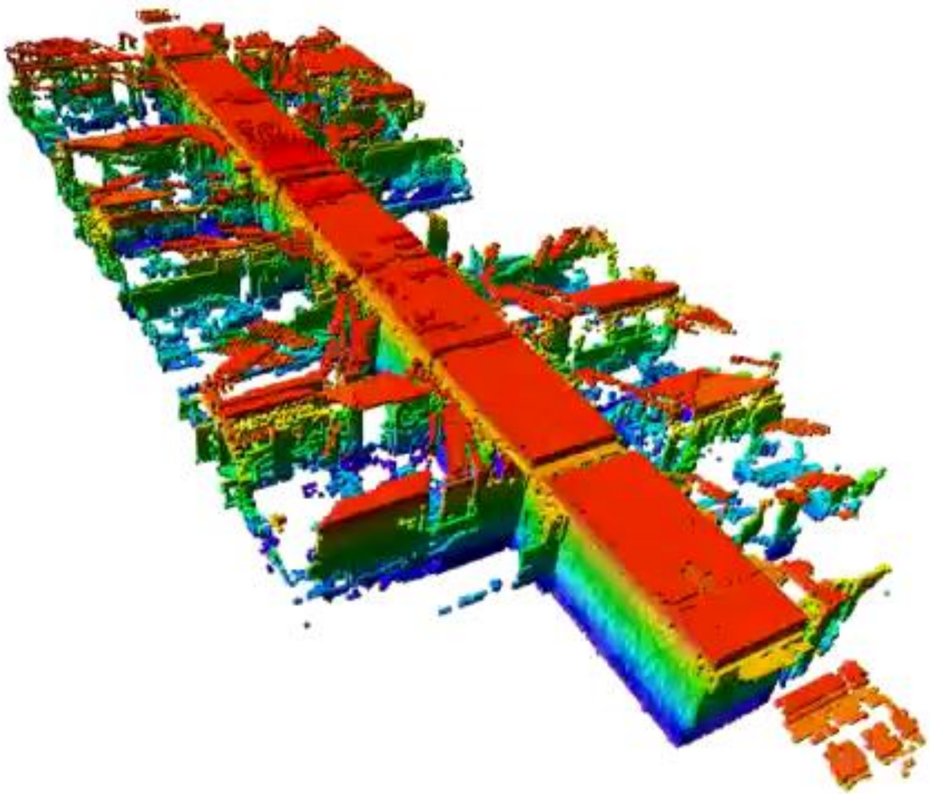




Example: OctoMap

[Wurm et al., 2011]

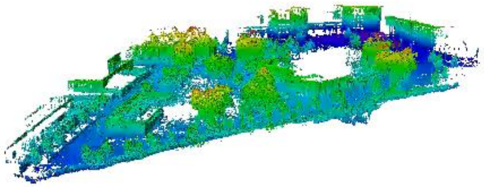
- Freiburg, building 79
  44 x 18 x 3 m^3, 0.05m resolution, 0.7mb on disk



Example: OctoMap

[Wurm et al., 2011]

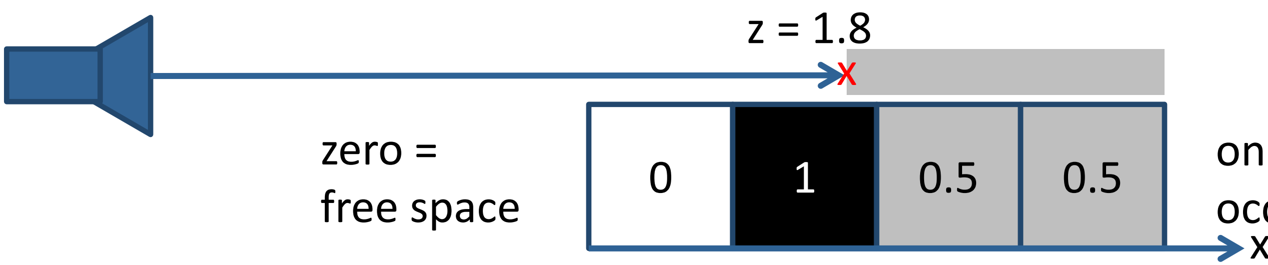
- Freiburg computer science campus
  292 x 167 x 28 m^3, 0.2m resolution, 2mb on disk



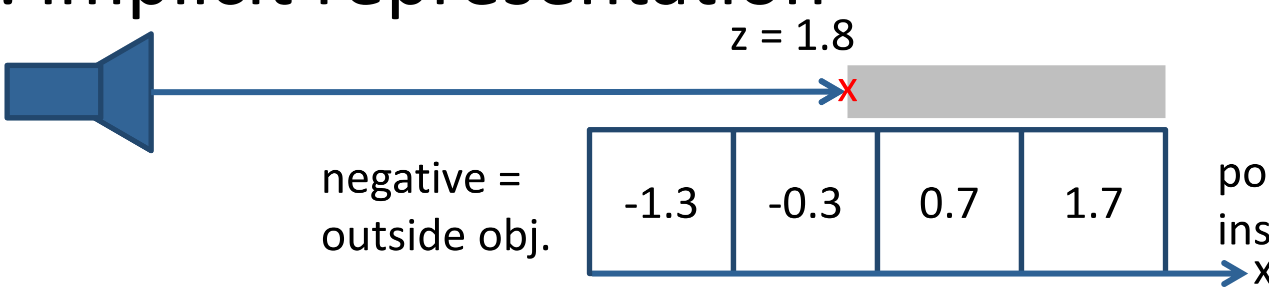
Signed Distance Field (SDF)

[Curless and Levoy, 1996]

- **Idea:** Instead of representing the cell occupancy, represent the distance of each cell to the surface
- Occupancy grid maps: explicit representation



- SDF: implicit representation

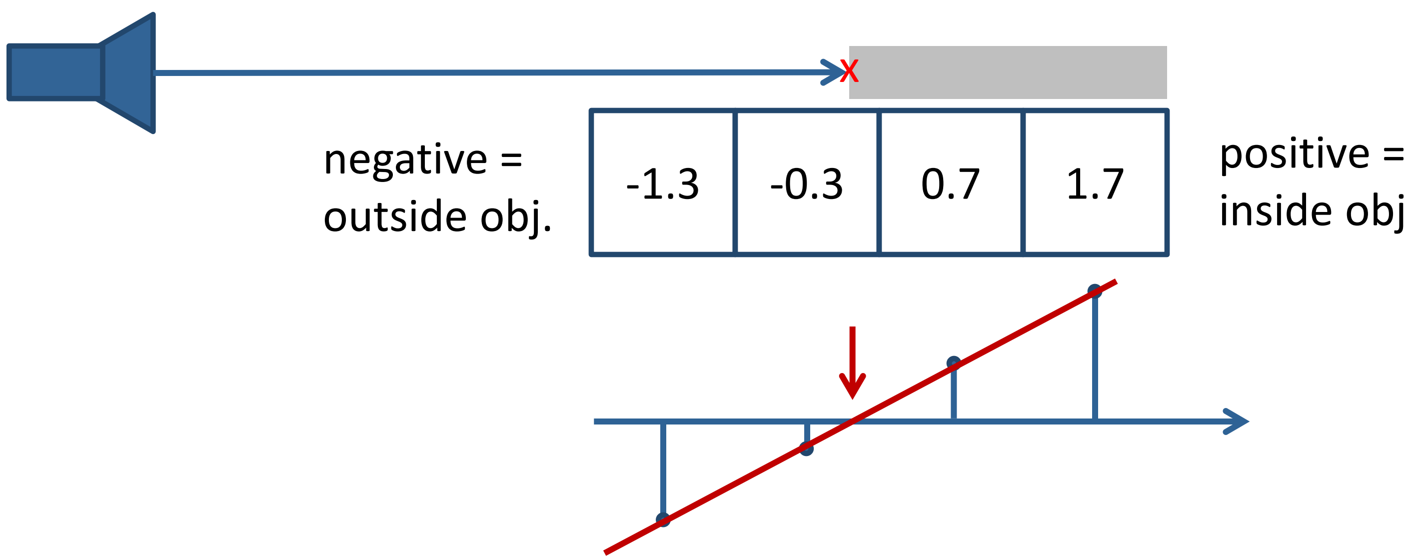


Signed Distance Field (SDF)

[Curless and Levoy, 1996]

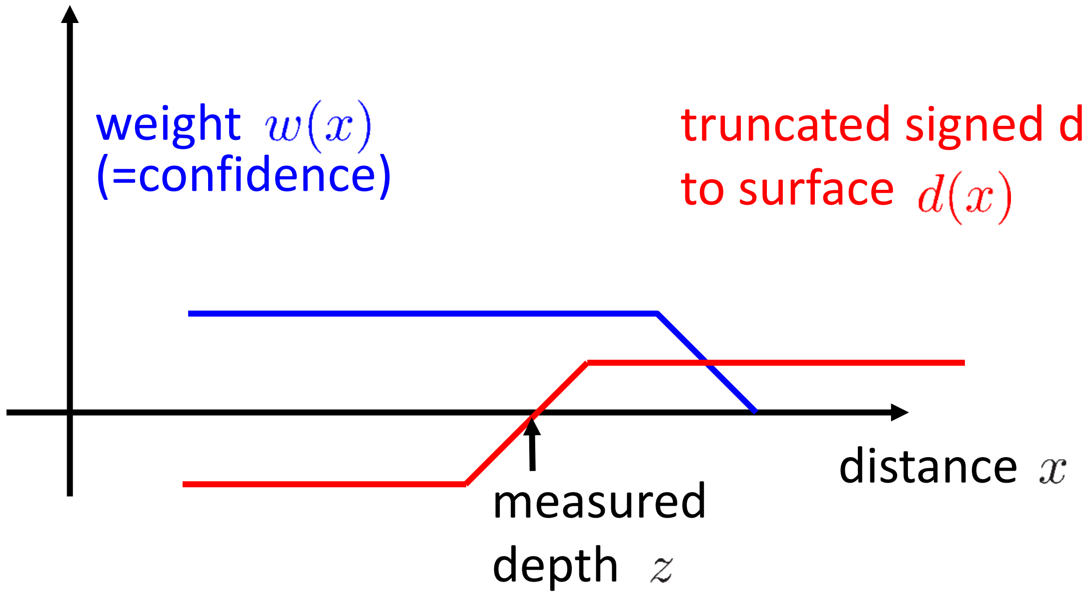
Algorithm:

1. Estimate the signed distance field
2. Extract the surface using interpolation (surface is located at zero-crossing)



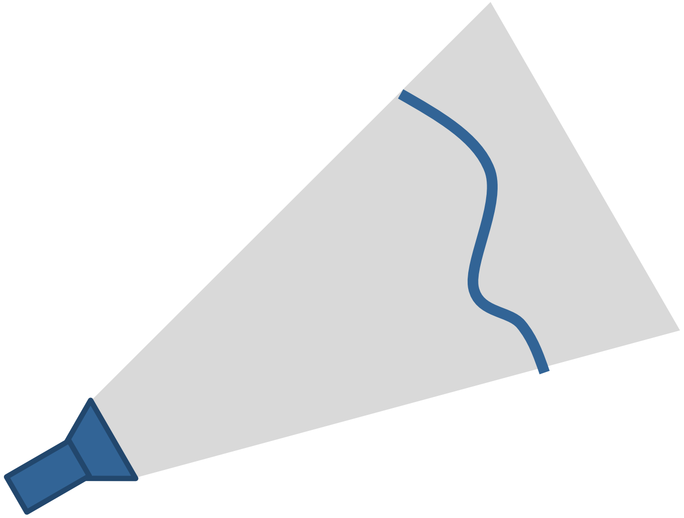
Distance and Weighting Functions

- Weight each observation according to its confidence
- Weight can additionally be influenced by other modalities (reflectance values, …)



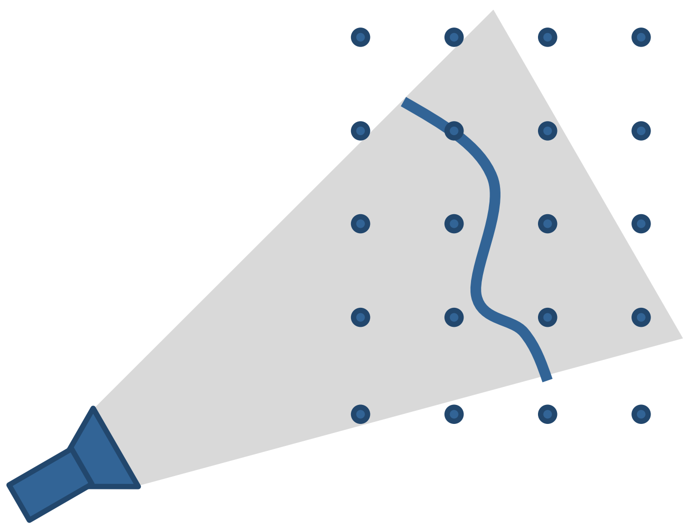
Dense Mapping: 2D Example

- Camera with known pose



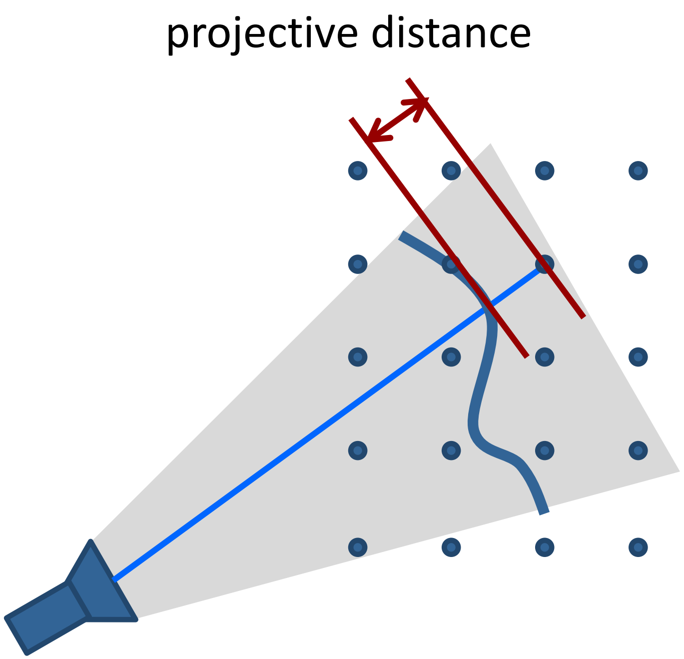
Dense Mapping: 2D Example

- Camera with known pose
- Grid with signed distance function



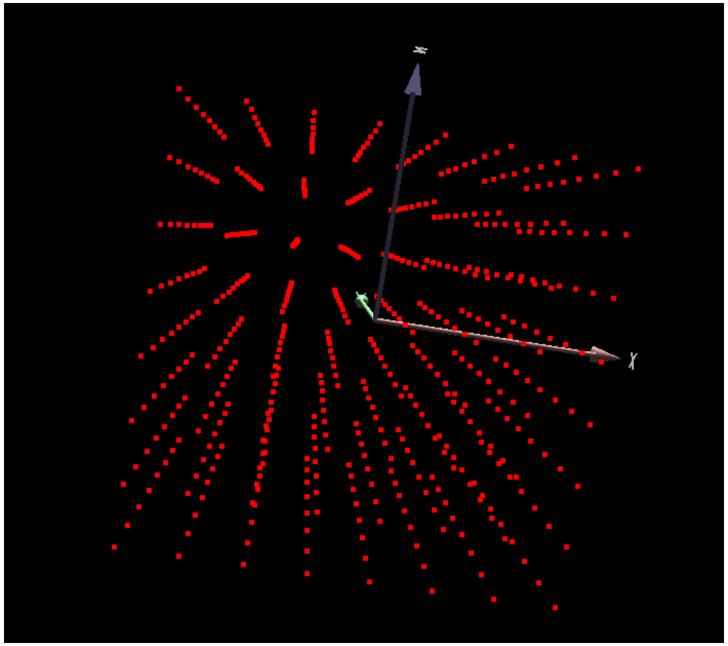
Dense Mapping: 2D Example

- For each grid cell, compute its projective distance to the surface



Dense Mapping: 3D Example

- Generalizes directly to 3D
- But: memory usage is cubic in side length



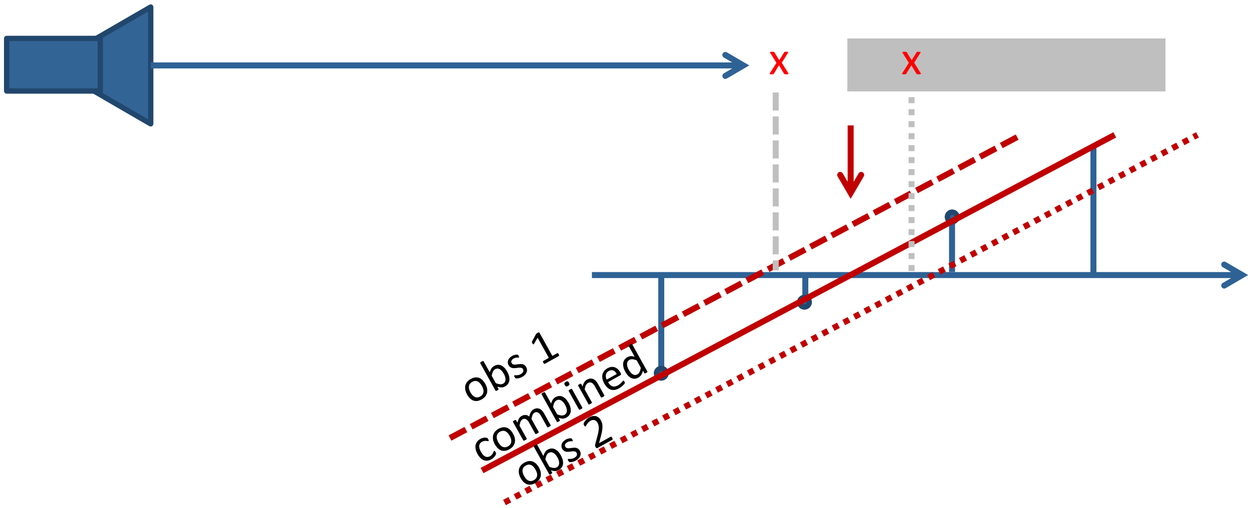
Data Fusion

- **Idea:** Compute weighted average
- Each voxel cell x in the SDF stores two values
  - Weighted sum of signed distances $D_t(\mathbf{x})$
  - Sum of all weights $W_t(\mathbf{x})$
- When new range image arrives, update every voxel cell according to

$$D_{t+1}(\mathbf{x}) = D_t(\mathbf{x}) + w_{t+1}(\mathbf{x}) d_{t+1}(\mathbf{x})$$
$$W_{t+1}(\mathbf{x}) = W_t(\mathbf{x}) + w_{t+1}(\mathbf{x})$$

Two Nice Properties

- Noise cancels out over multiple measurements



- Zero-crossing can be extracted at sub-voxel accuracy (least squares estimate)

1D Example: $$x^* = \frac{\sum D_t(x)x}{\sum W_t(x)x}$$

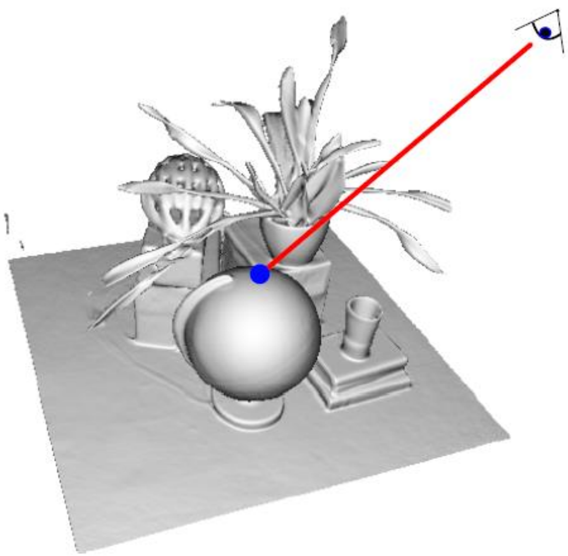
Visualizing Signed Distance Fields

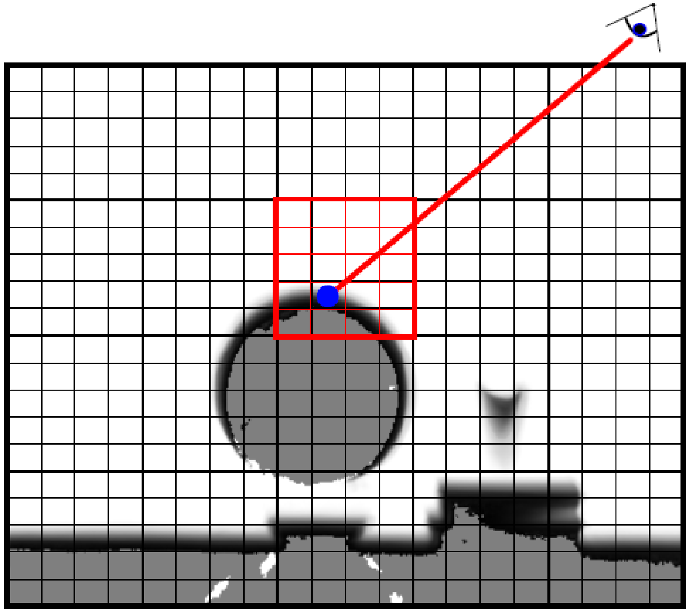
Common approaches to iso surface extraction:

1. Ray casting (GPU, fast)
   For each camera pixel, shoot a ray and search for zero crossing
2. Poligonization (CPU, slow)
   E.g., using the marching cubes algorithm
   Advantage: outputs triangle mesh

Ray Casting

- For each camera pixel, shoot a ray and search for the first zero crossing in the SDF
- Value in the SDF can be used to skip along when far from surface

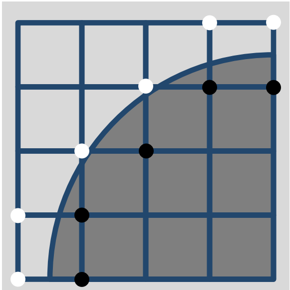


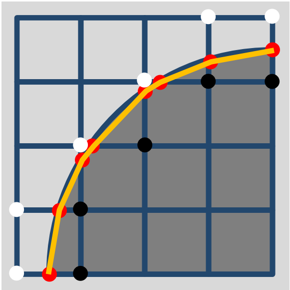


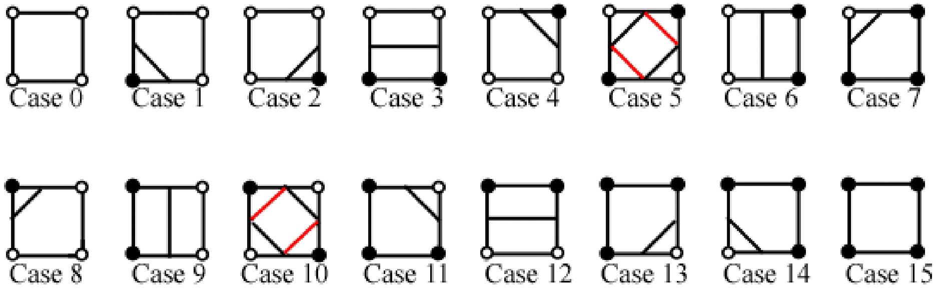
Marching Cubes

First in 2D, **marching squares**:

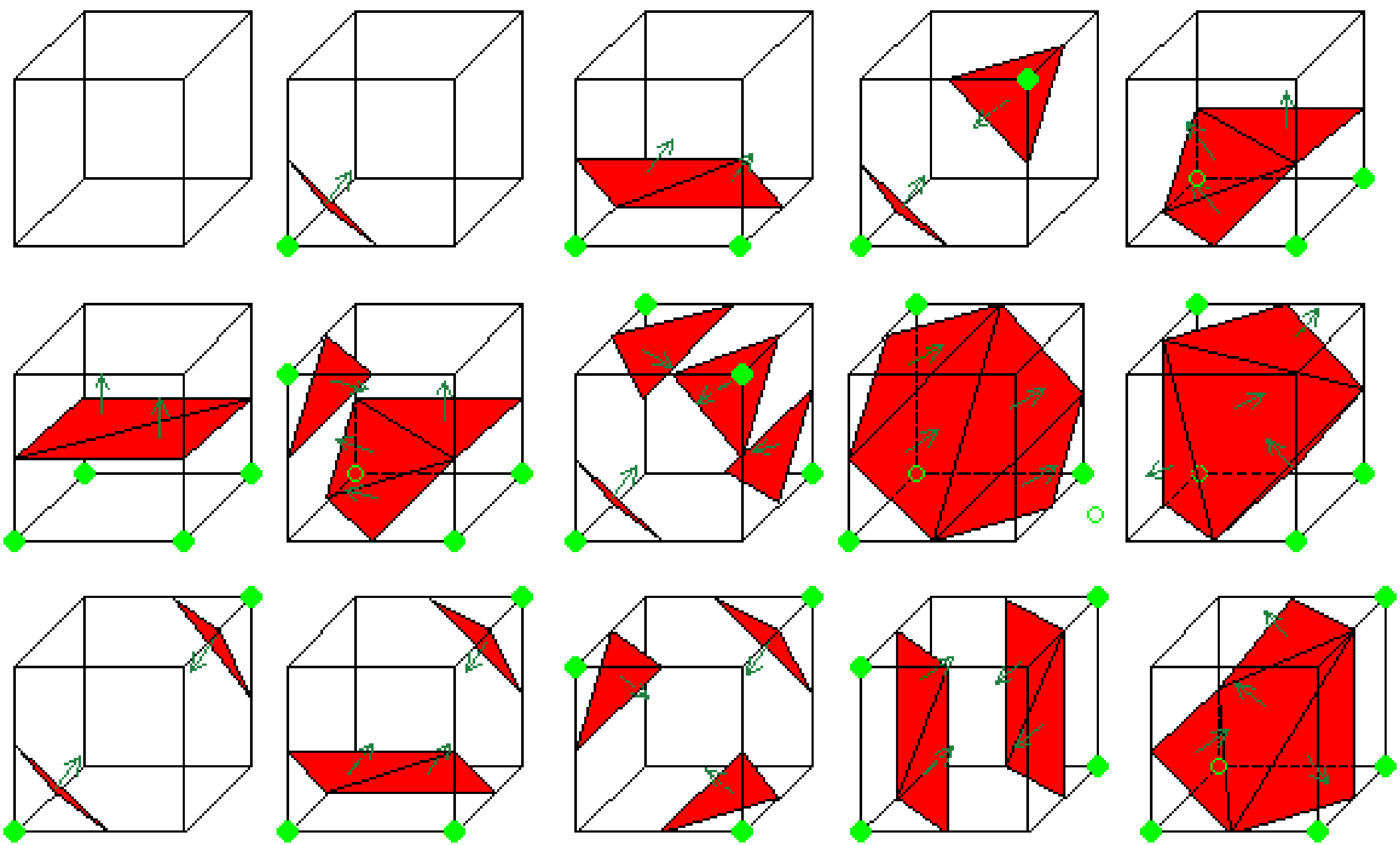
- Evaluate each cell separately
- Check which edges are inside/outside
- Generate triangles according to lookup table
- Locate vertices using least squares







Marching Cubes



KinectFusion

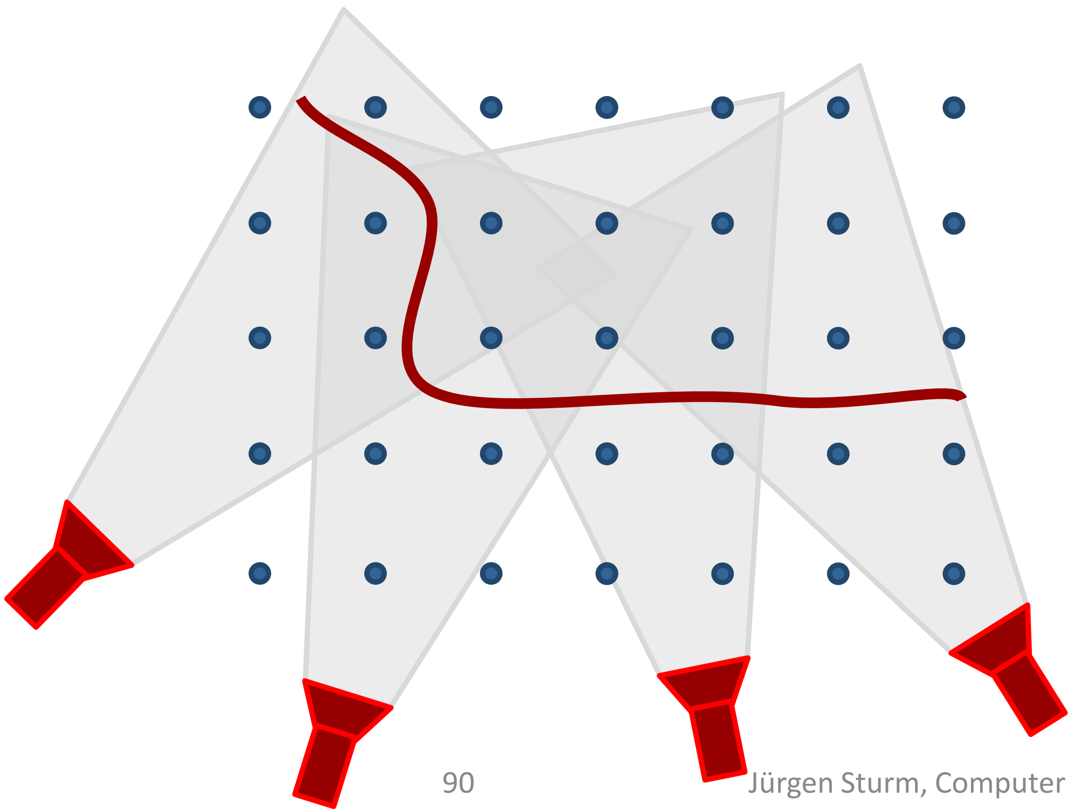
[Newcombe et al., 2011]

- Projective ICP with point-to-plane metric
- Truncated signed distance function (TSDF)
- Ray Casting

Dense Tracking: 2D Example

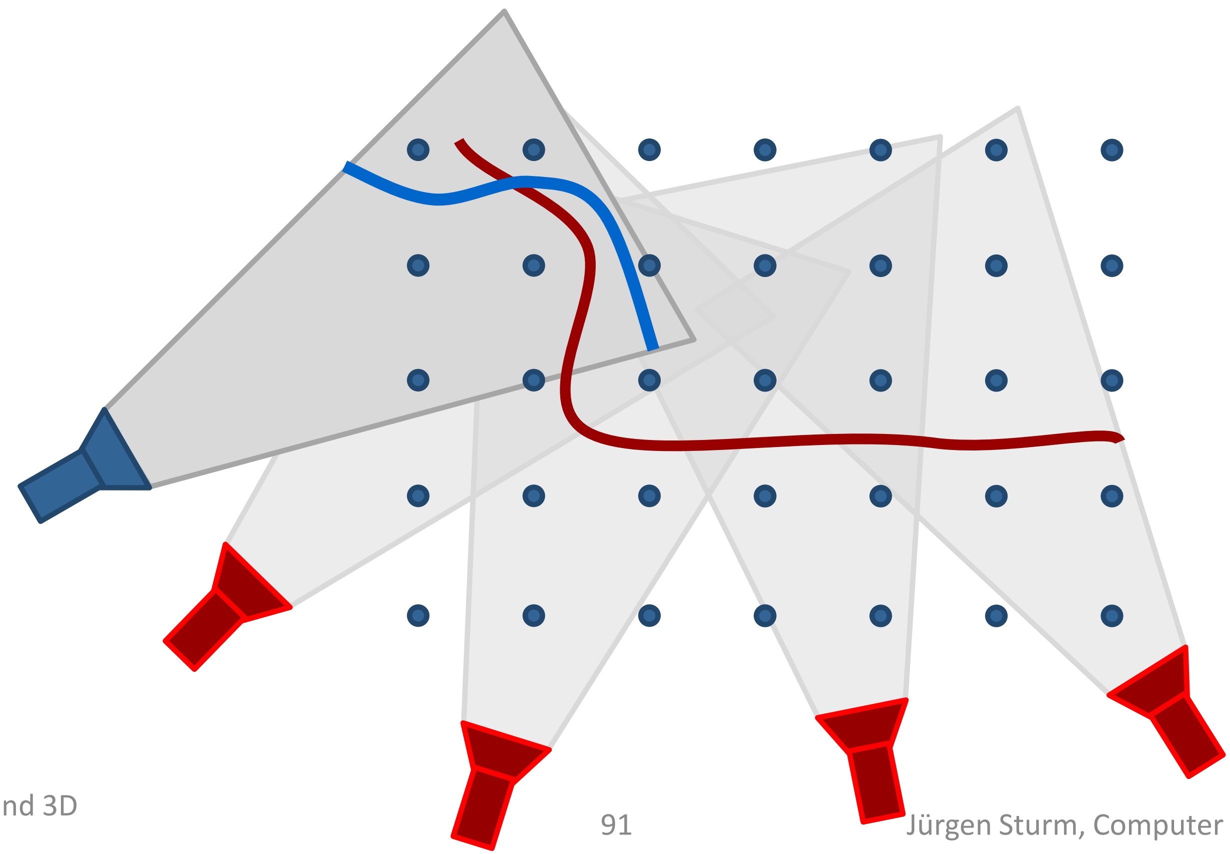
[Bylow et al., RSS 2013]

- 3D model built from the first k frames



Dense Tracking: 2D Example
[Bylow et al., RSS 2013]

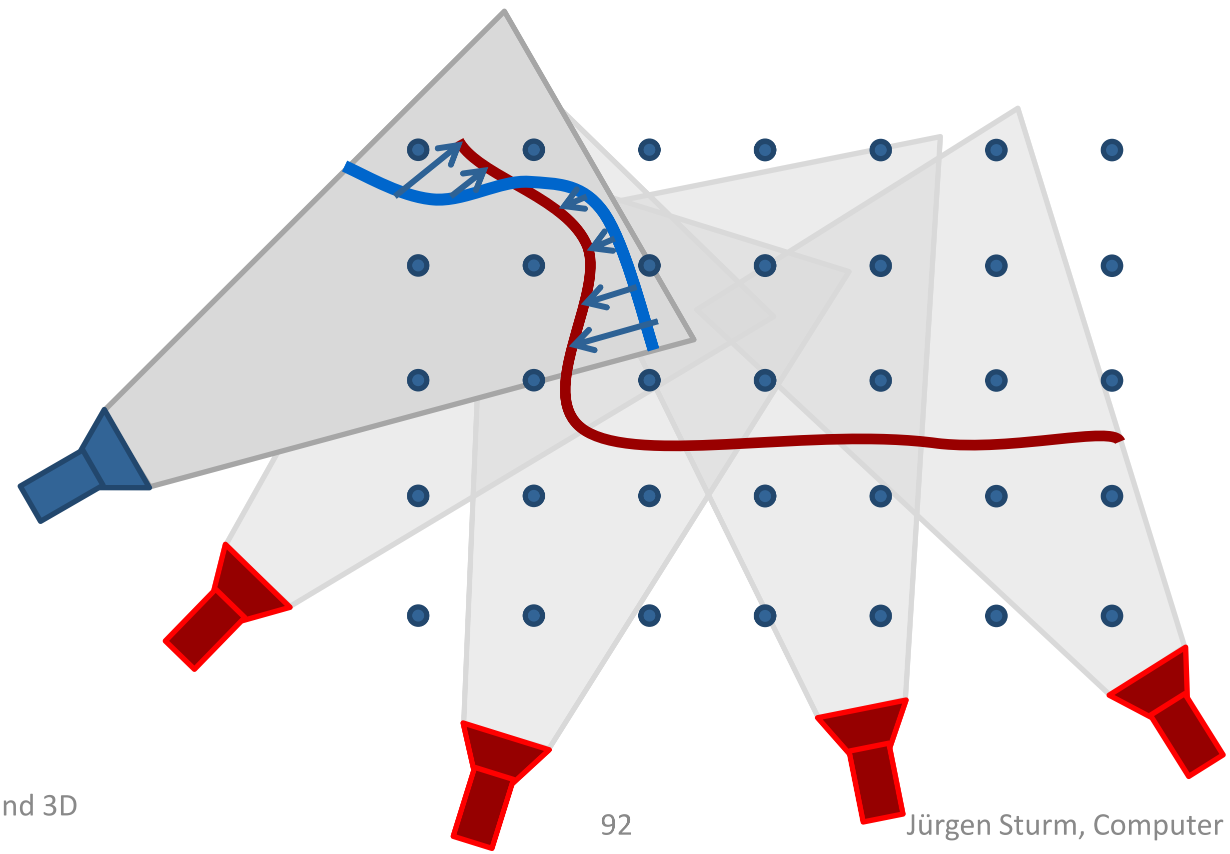
- Minimize distance between depth image and SDF



Dense Tracking: 2D Example

[Bylow et al., RSS 2013]

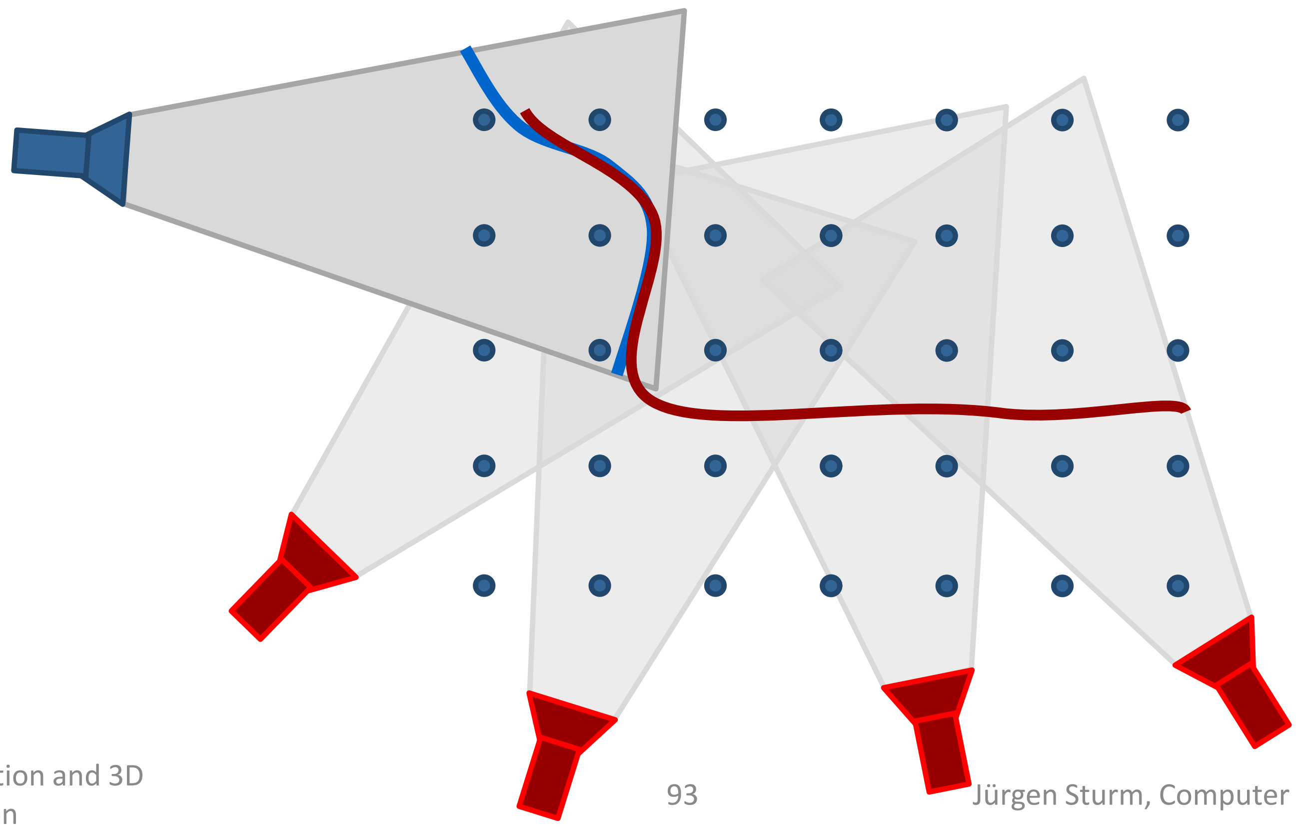
- Minimize distance between depth image and SDF



Dense Tracking: 2D Example

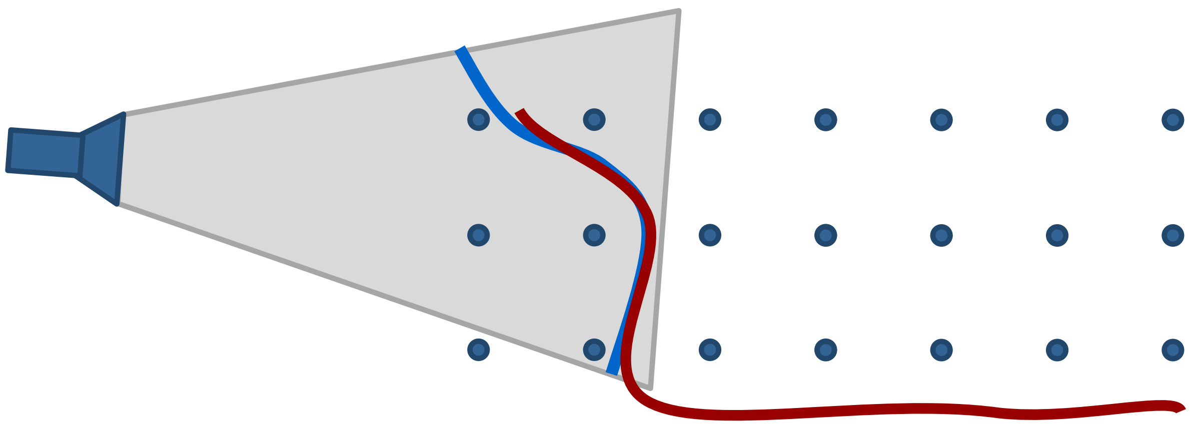
[Bylow et al., RSS 2013]

- Minimize distance between depth image and SDF



Dense Tracking: 2D Example
[Bylow et al., RSS 2013]

- Minimize distance between depth image and SDF



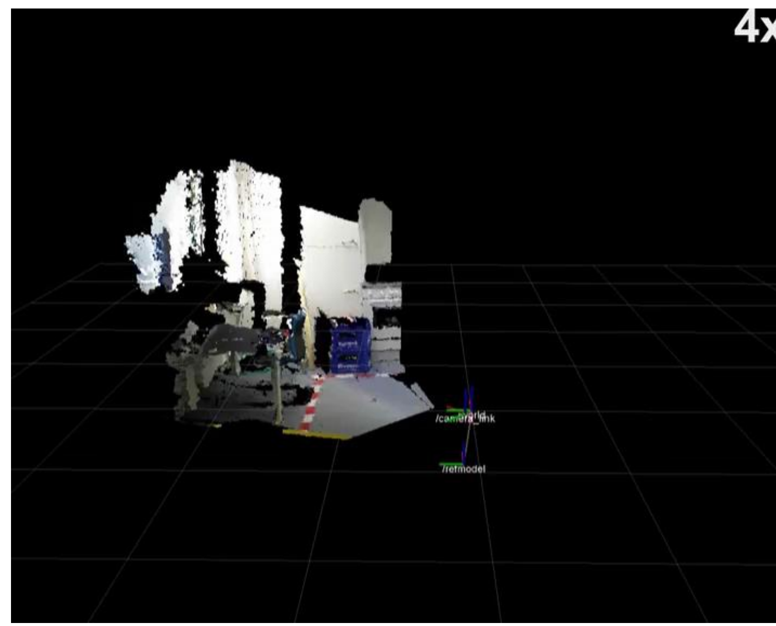
$$\arg\min_{\xi} E(\xi) = \arg\min_{\xi} \frac{1}{M} \sum_{ij} V(X(\xi, (i,j), I_d))^2$$

3D Reconstruction from a Quadrocopter

[Bylow et al., RSS 2013]

- AscTec Pelican quadrocopter
- Real-time 3D reconstruction, position tracking and control

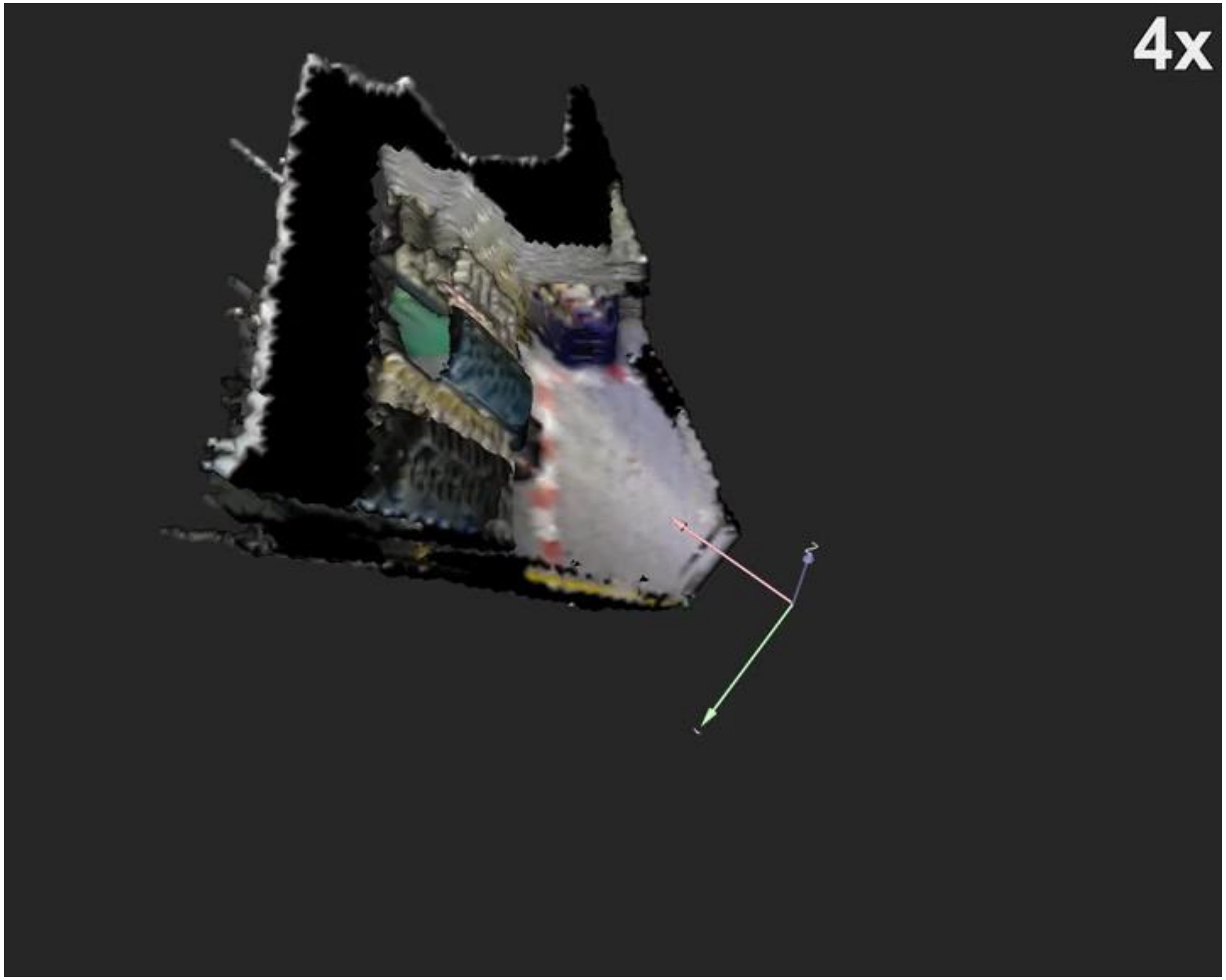
external view



estimated pose

Resulting 3D Model

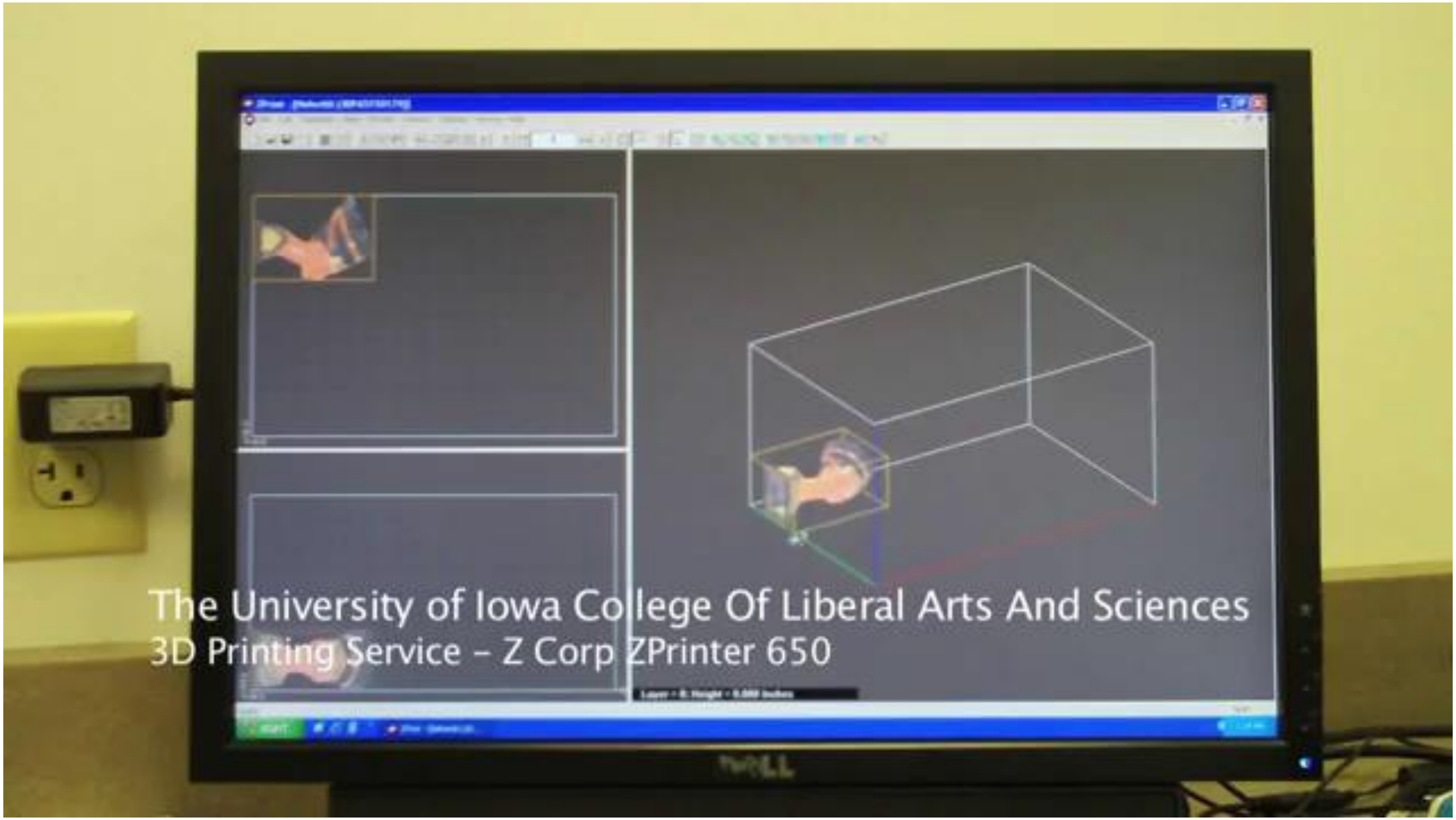
[Bylow et al., RSS 2013]



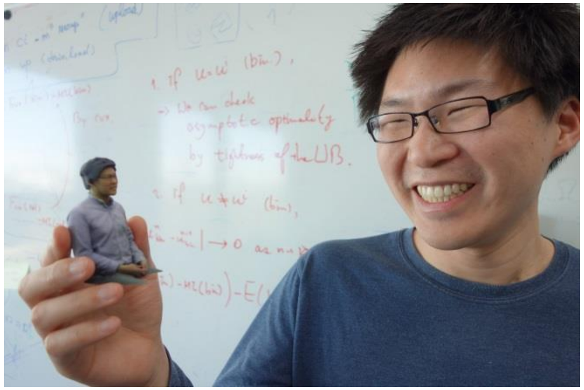
Let's Scan a Person!

[Sturm et al., GCPR 2013]

3D Color Printing



Can We Print These Models in 3D?



- Who wants to get a 3D scan of him/herself?

Lessons Learned Today

- How to estimate the camera poses and 3D points from monocular images using bundle adjustment
- How depth cameras work
- How to estimate occupancy maps
- What signed distance functions are
- How to reconstruct triangle meshes from SDFs