Dense Reconstruction

"SLAM gets Graphic"
or
"To kinfinity and beyond"

STEPHEN MILLER

Outline

Intro	Who am I, What are we going to talk about, How does it fit with what you've learned
Reconstruction Basics	Graphics perspectives on the SLAM problem, "volumetric reconstruction"
Kinect Fusion	Enter roboticists; realtime, online, "frame-to-model registration"
Current / Future	What does it look like in 2013? What easy papers are waiting to be written?

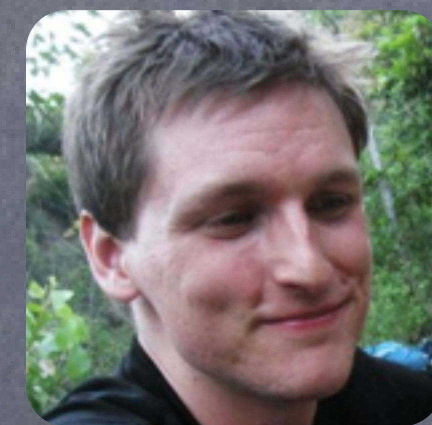
Preliminary thanks

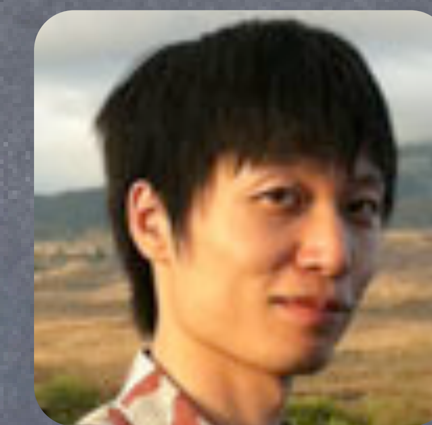
Co-authors

Alex Teichman
Andrej Karpathy
Qianyi Zhou
Fei-Fei Li
Vladlen Koltun
Sebastian Thrun

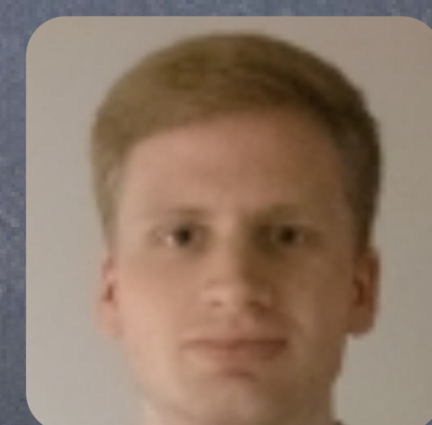
Figure-Donors

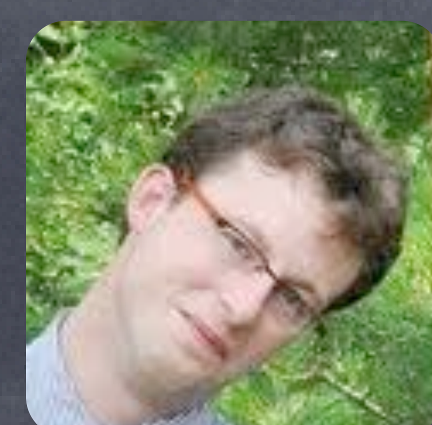
Erik Bylow
Juergen Sturm
Thom Whelan
Peter Henry
Radu Rusu










Intro

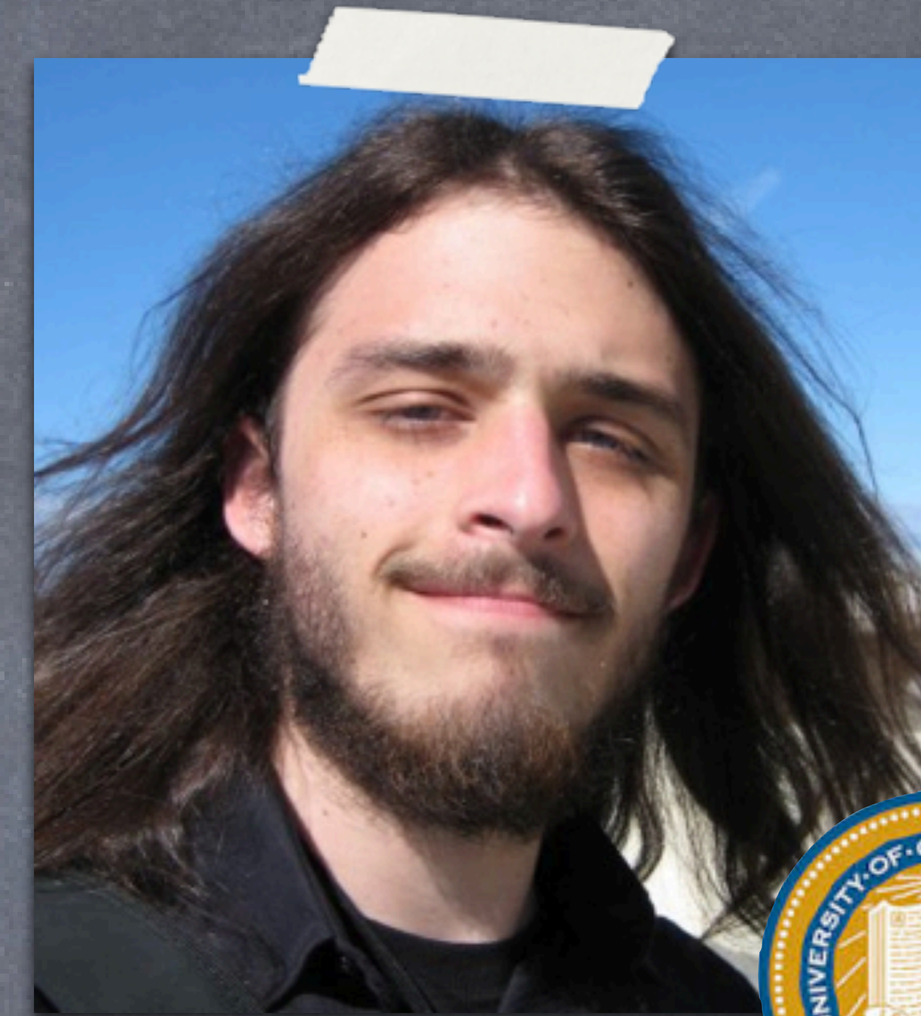
Intro → Reconstruction Basics → KinectFusion → Current/Future

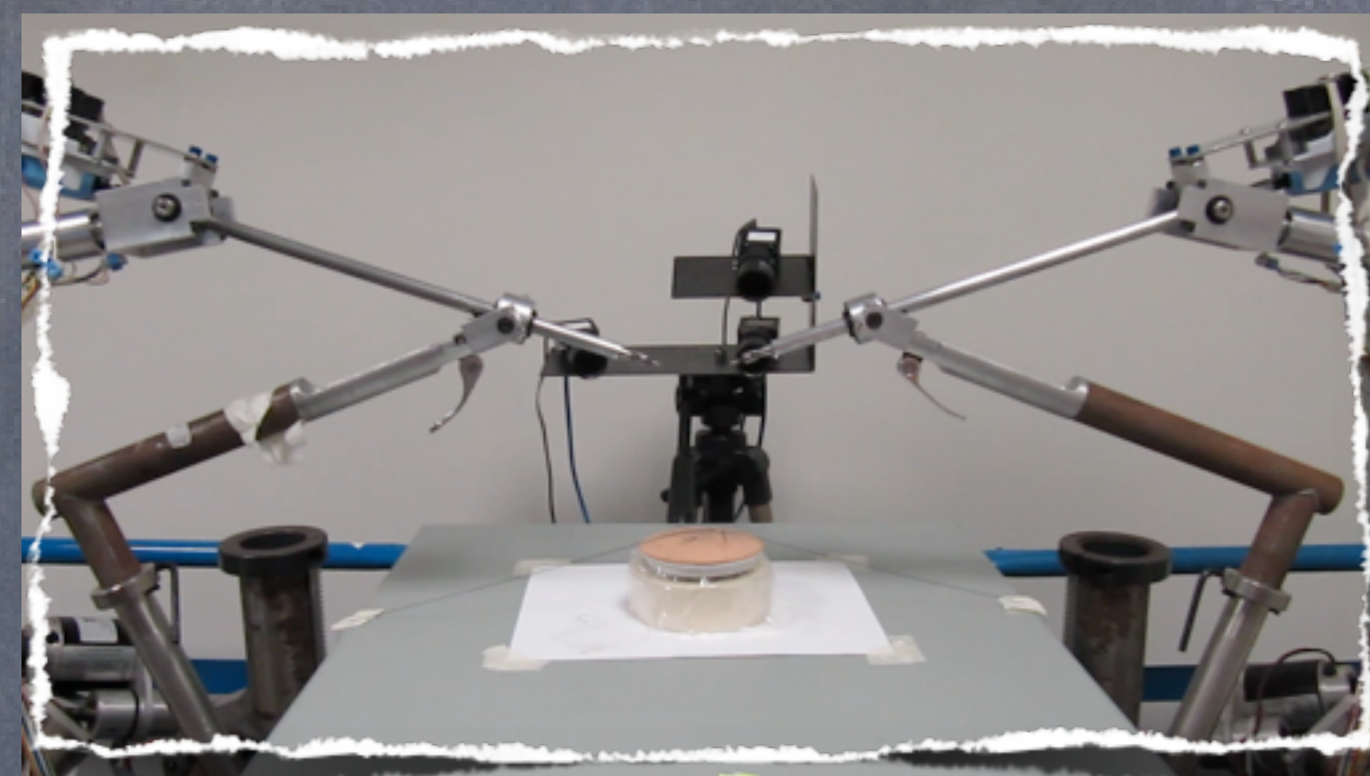
Who am I

- Undergrad: Cal

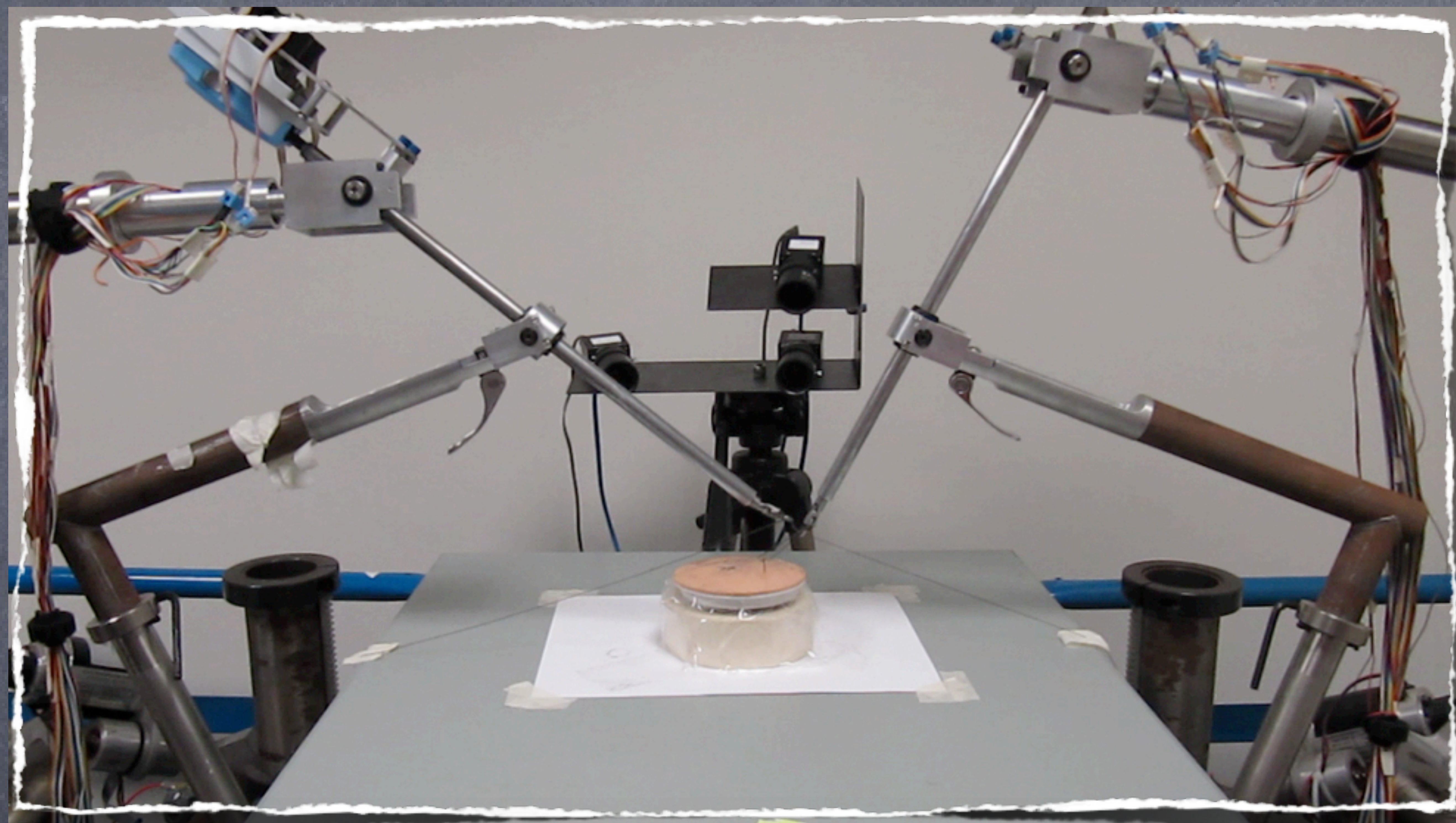
  - (2009)
    Surgical Robotics

  - (2010-2011)
    Personal Robotics





Changing priorities

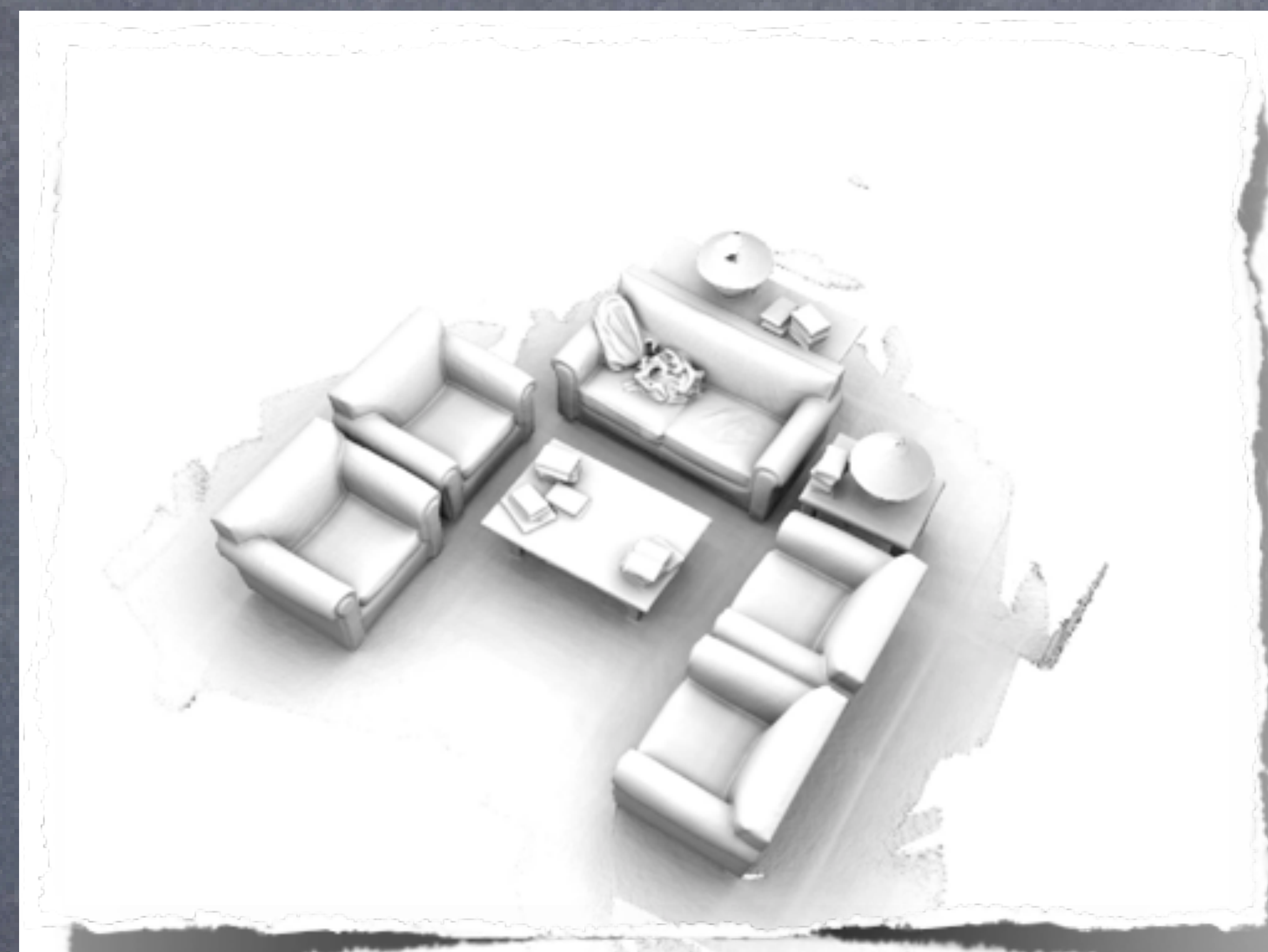


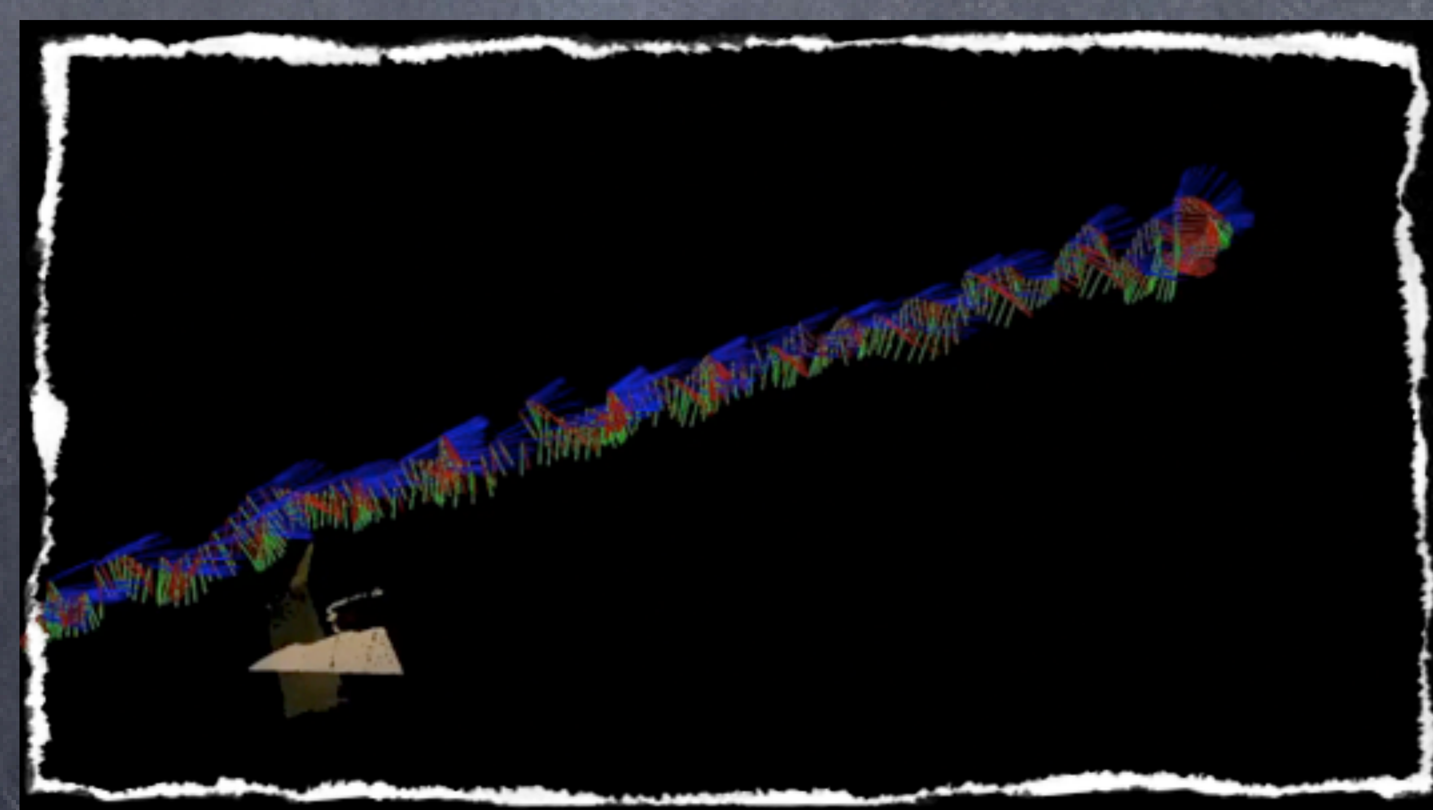
Blind robots are <u>frustrating</u> to work with

Who am I

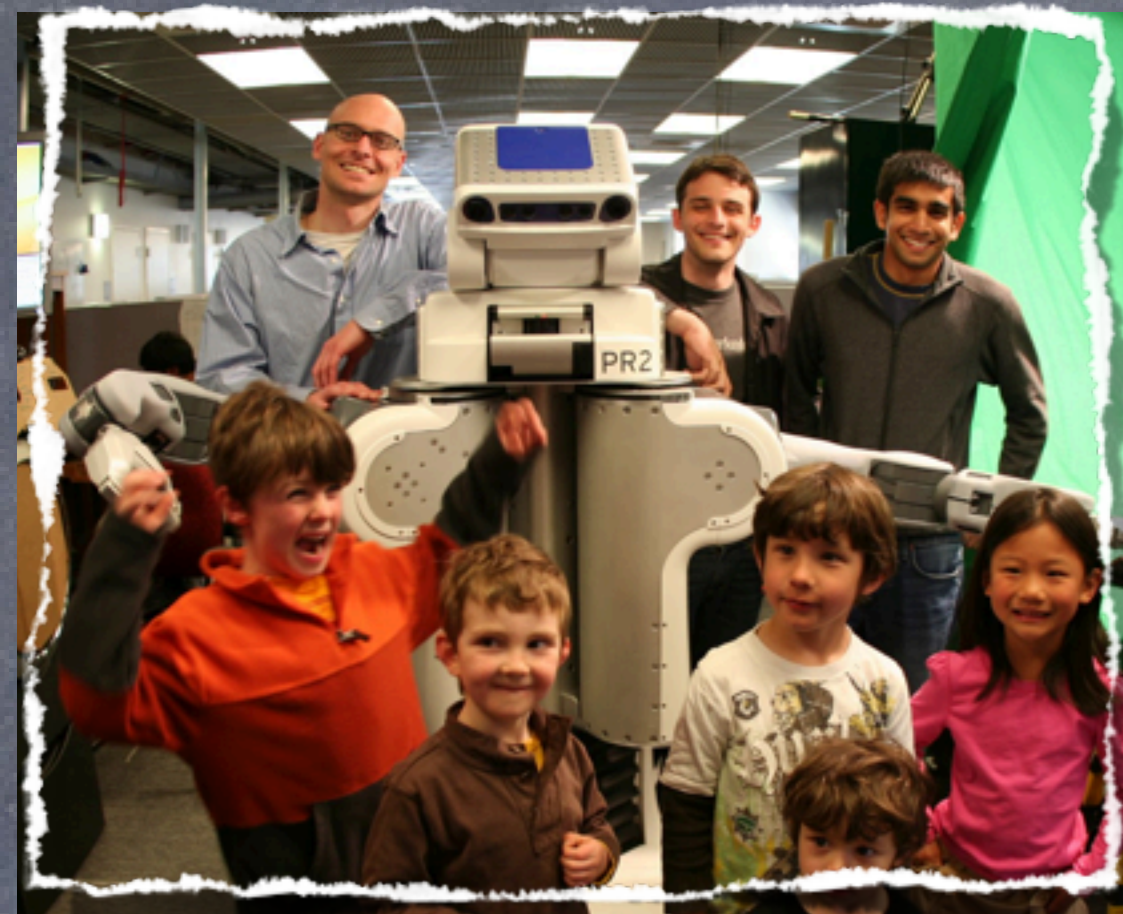
- Grad: Stanford

  - (2011)
  Existential crisis
  - (2012-)
  3D Perception:
  * Calibration
  * Mapping
  * Object
    Detection





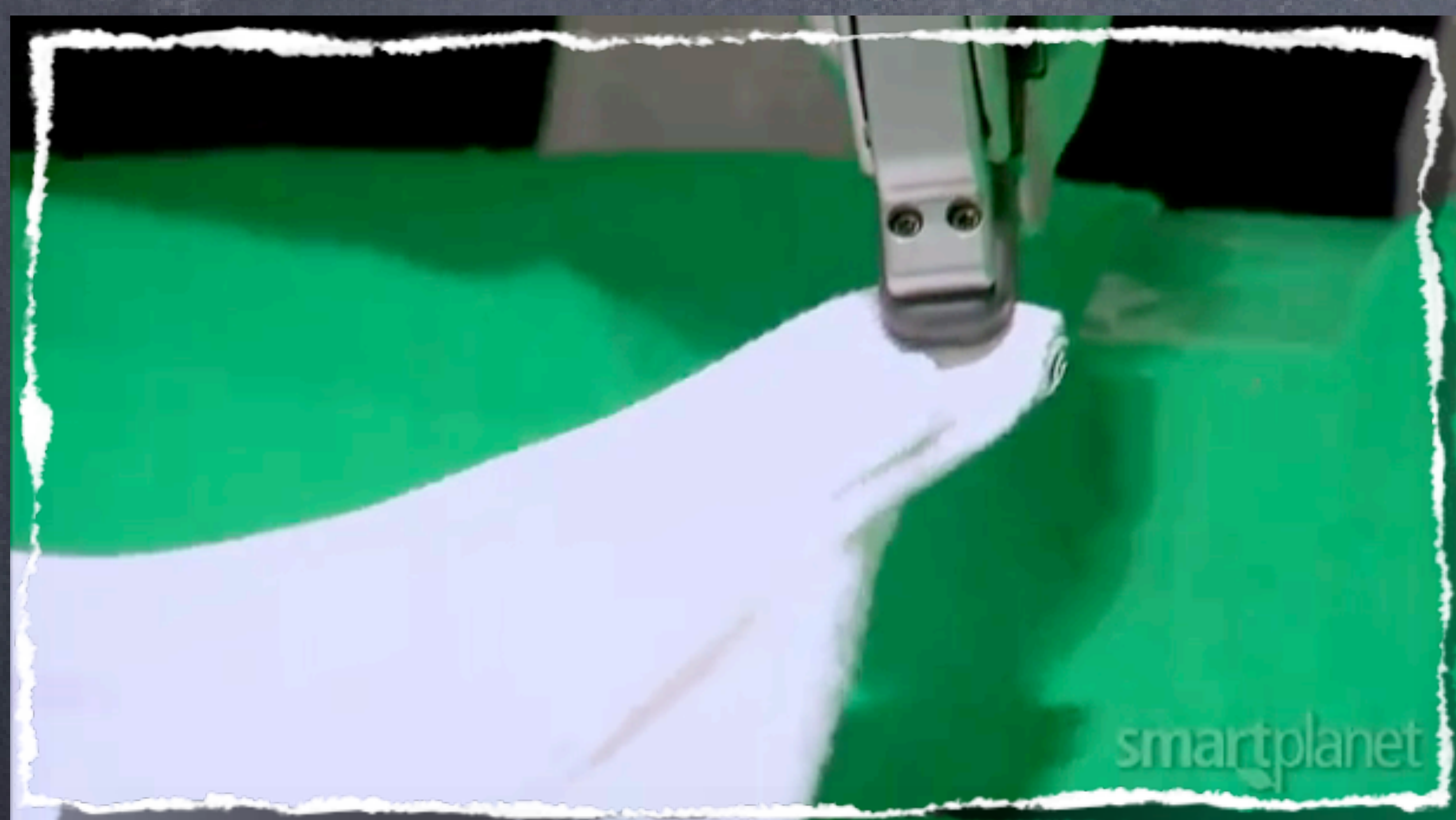
Who I owe it all (>= 95%) to

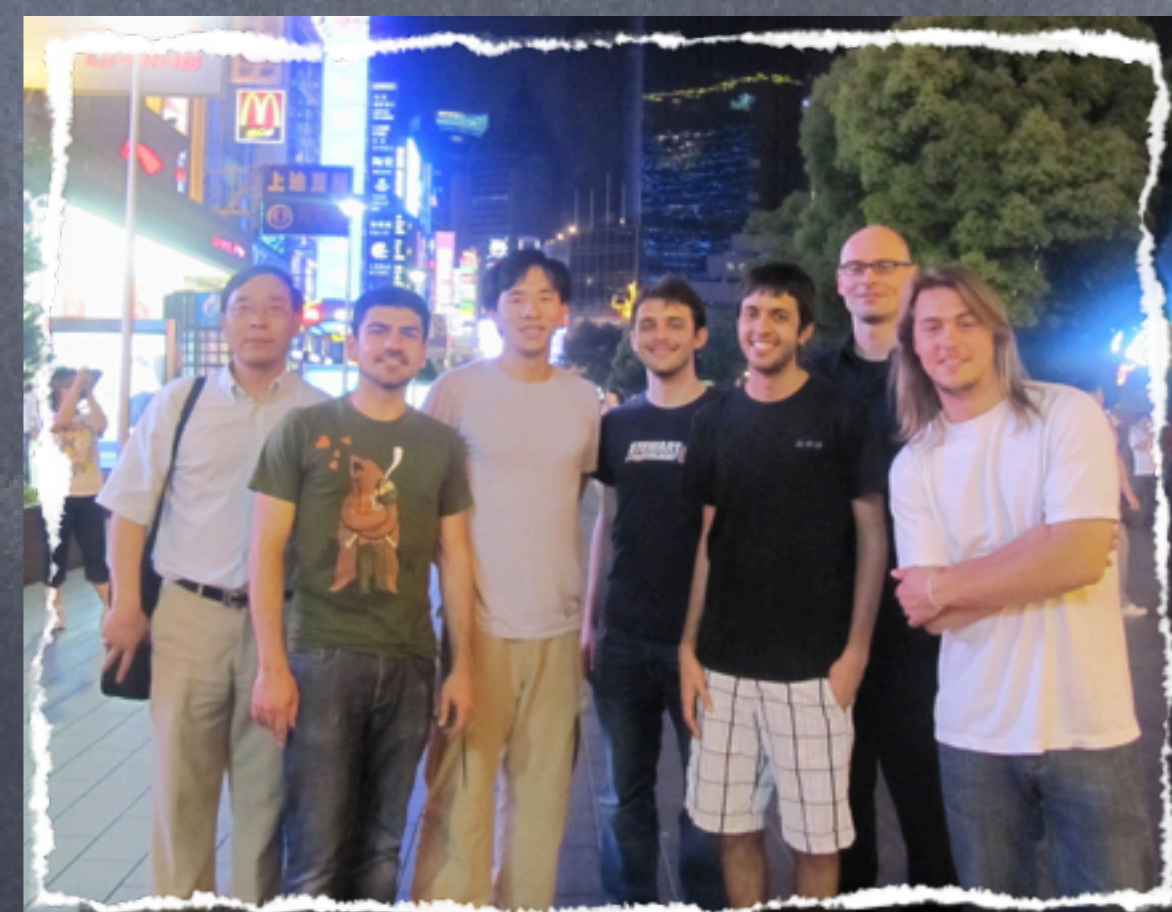


Anchorage, '10

Berkeley, '10

Singapore, '10





CBS Smart Planet, '11

Shanghai, '11

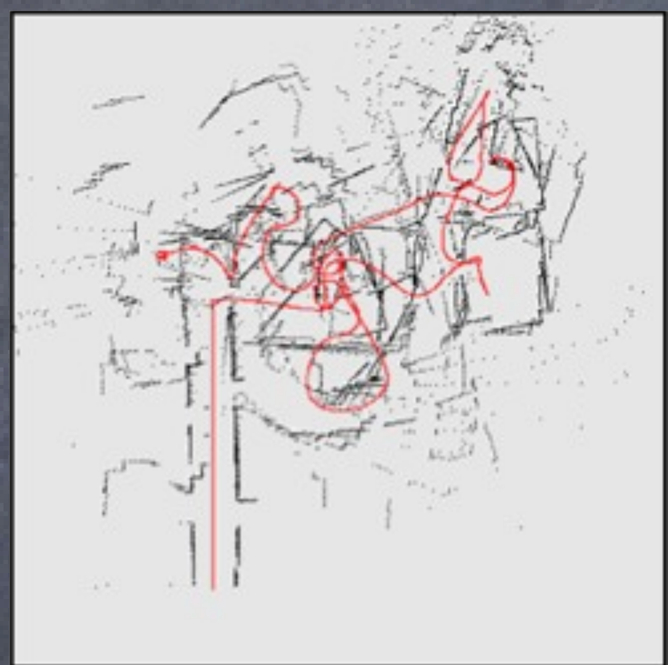
Tokyo, '13

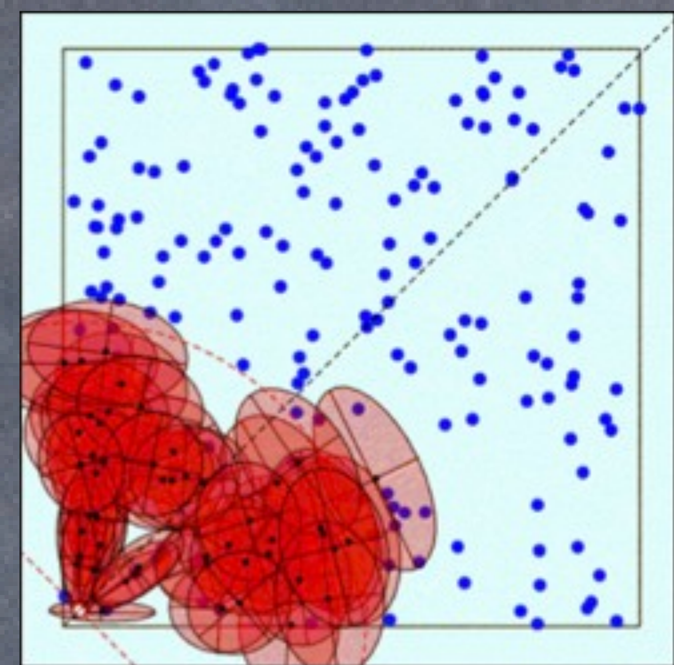
Okay,
back to work

Intro → Reconstruction Basics → KinectFusion → Current/Future

SLAM, as you've learned so far

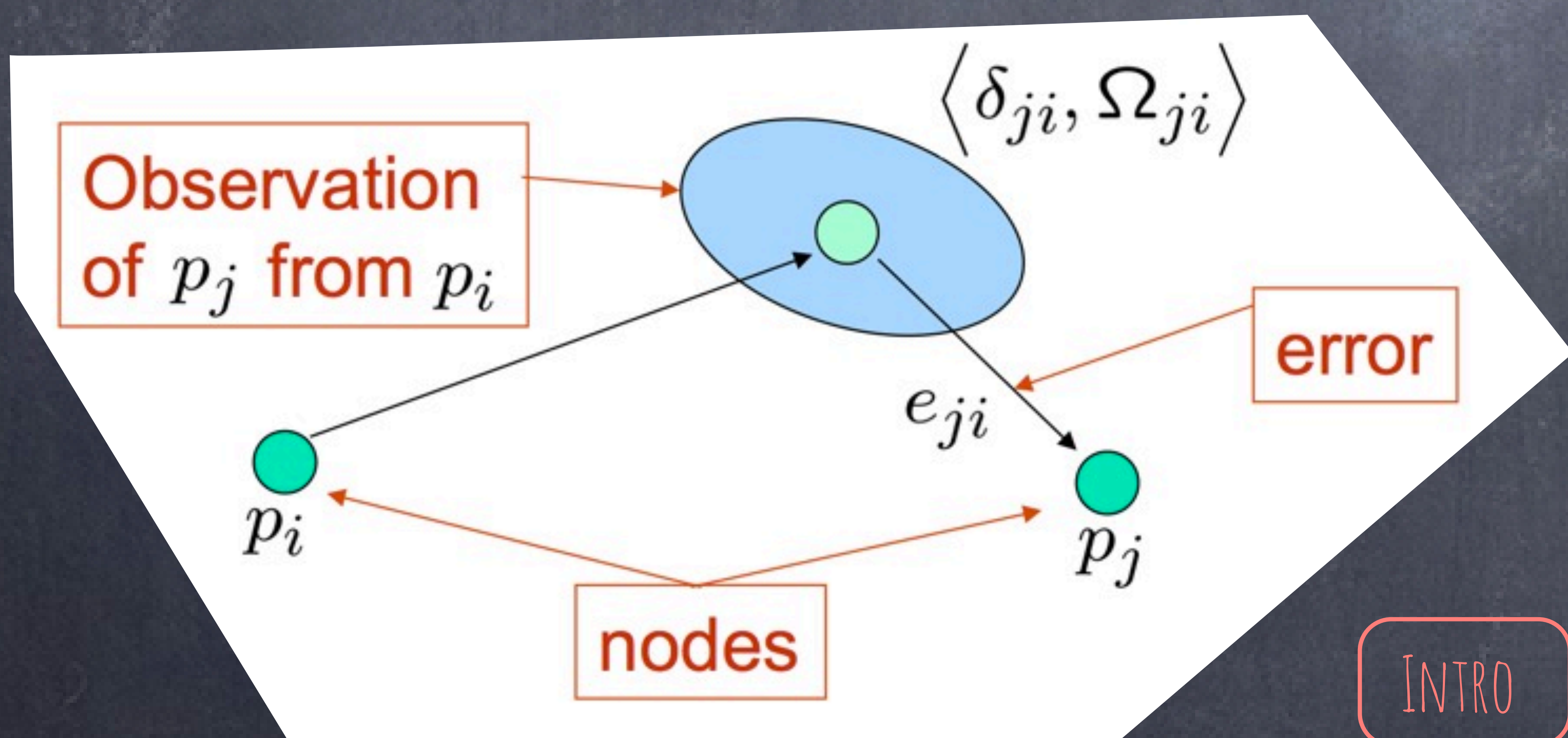
Track sparse features





"Map" in postprocessing

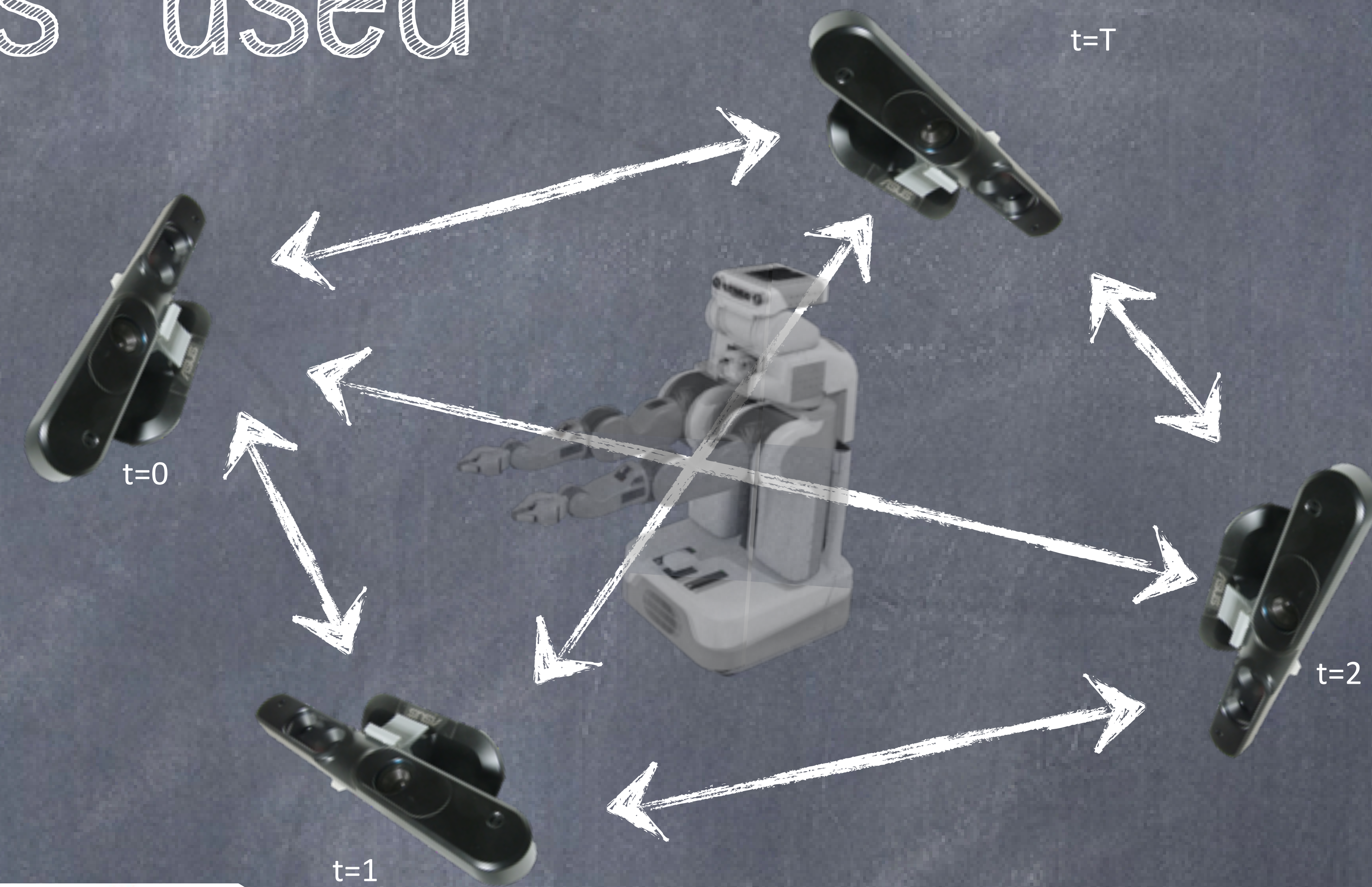
Poses related via Bayes law



Elegant math integrates all observations

$$w_t^i = \frac{p(z_t \mid x_{1:t}^i, z_{1:t-1})\, p(x_t^i \mid x_{1:t-1}^i, u_t, z_{1:t-1})}{\pi(x_t^i \mid x_{t-1}^i, u_t, z_t)}$$

SLAM, as it's used



- One graph: poses are related by edges
  - node(i): sensor pose at time i
  - edge(i,j): how does pose i relate to pose j?
  - weights: covariance (pssst, usually identity is fine)
  - basically GMapping

$$\max_{x,v,w} \log P(v,w)$$
$$\text{s.t.}\quad \forall t \quad x_{t+1} = f(x_t, u_t) + w_t$$
$$z_t = g(x_t) + v_t$$

- We still need the theory

- But it tends to be a black box

Software :: FABMAP

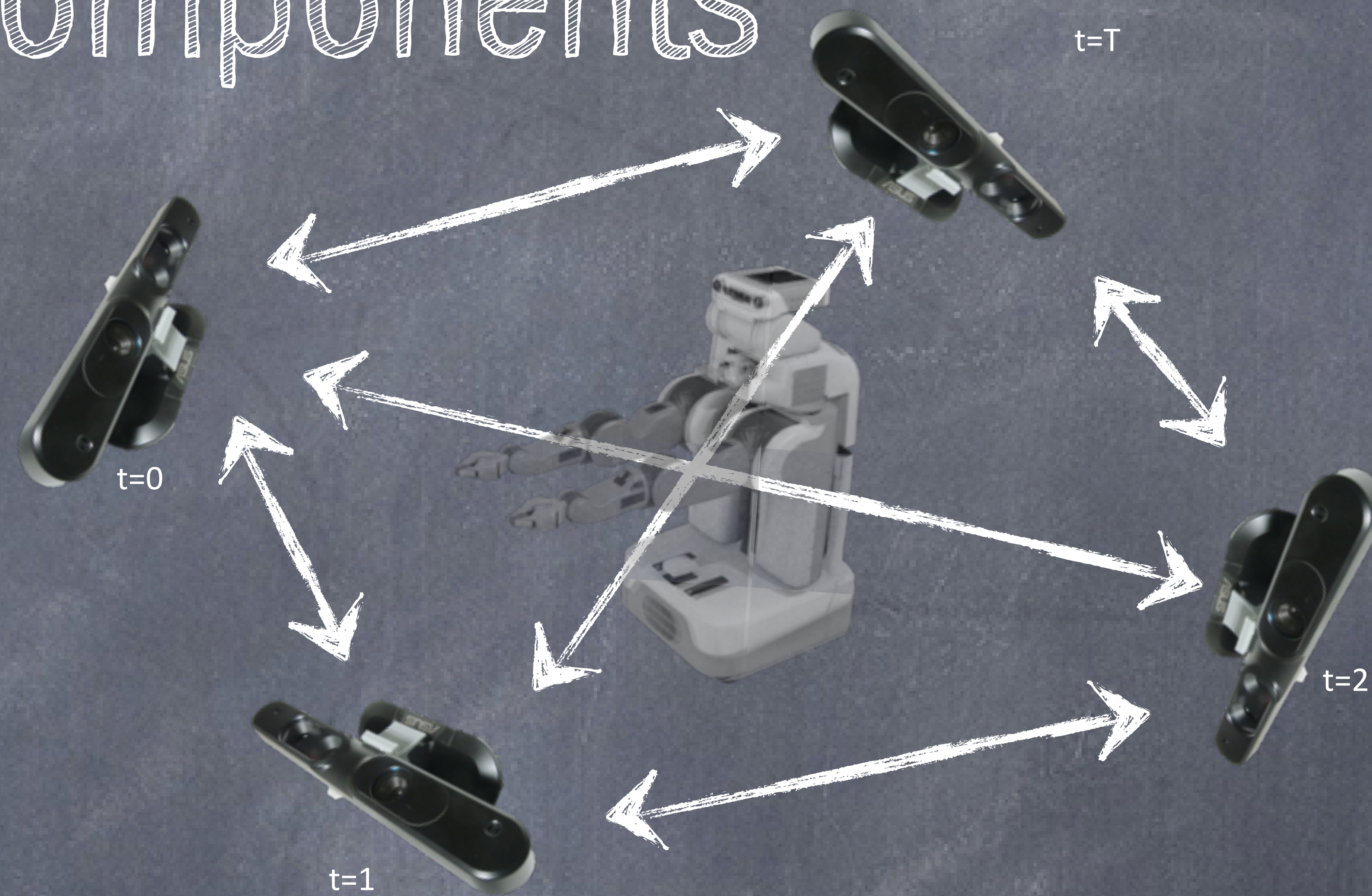
g2o: A General Framework for Graph Optimization

iSAM (incremental Smoothing and Mapping)

```
$ sudo apt-get install SLAM
```

SLAM: primary components

- **Odometry**: how does time T relate to time T+1?
- **Loop Closure**: how can I adjust my belief to handle new information?
- **Map**: What does the world look like when I put all observations together?



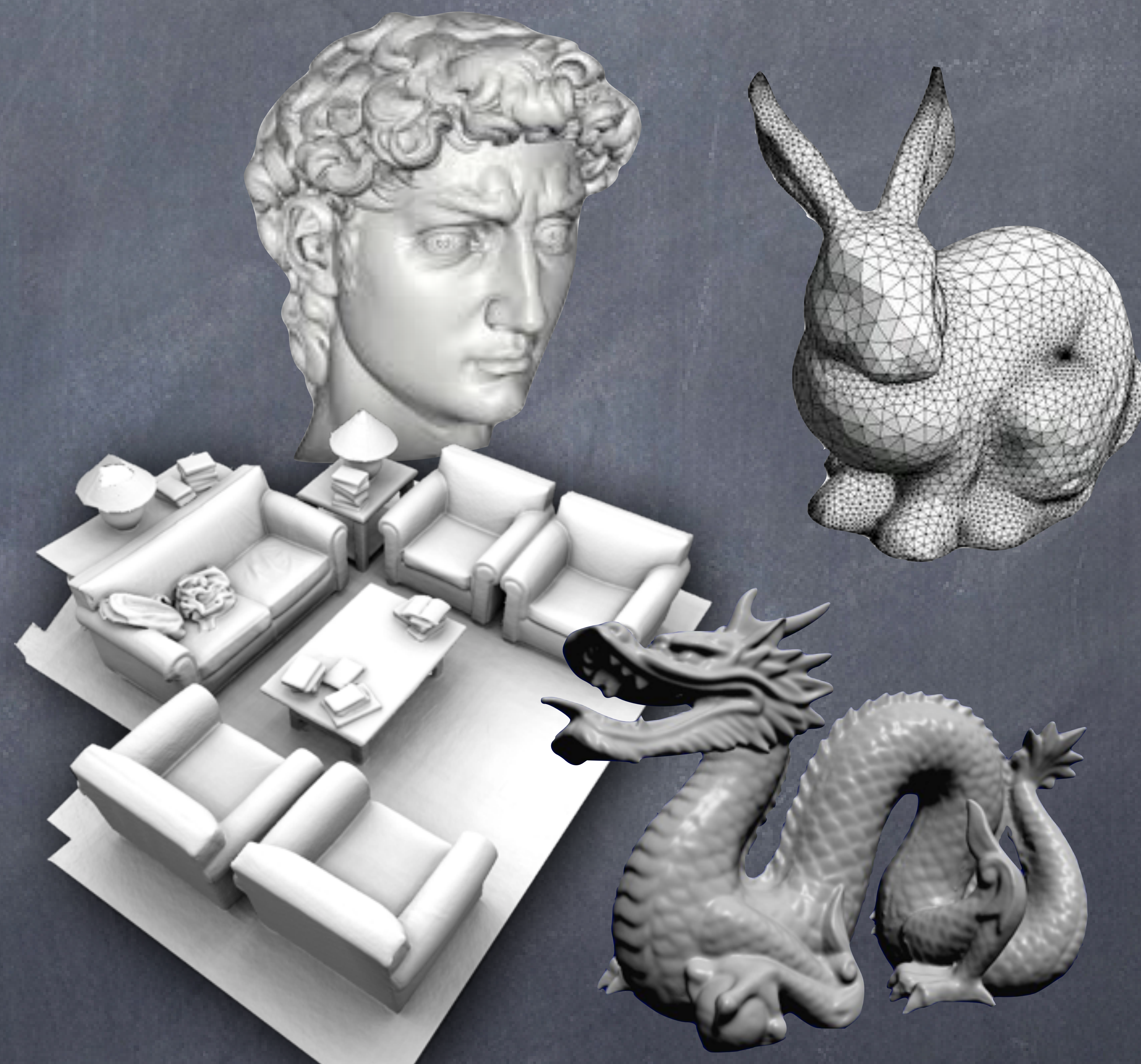
Reconstruction Basics

Robotics

Graphics

- SLAM

- Surface Reconstruction



INTRO → RECONSTRUCTION BASICS → KINECTFUSION → CURRENT/FUTURE

Robotics

- SLAM
- Map is a means to an end (localization)
- Pose unknown; priors given by IMU/GPS/etc
- Fairly precise sensors
- Need a pose estimate at every timestep (use everything you can!)

Graphics

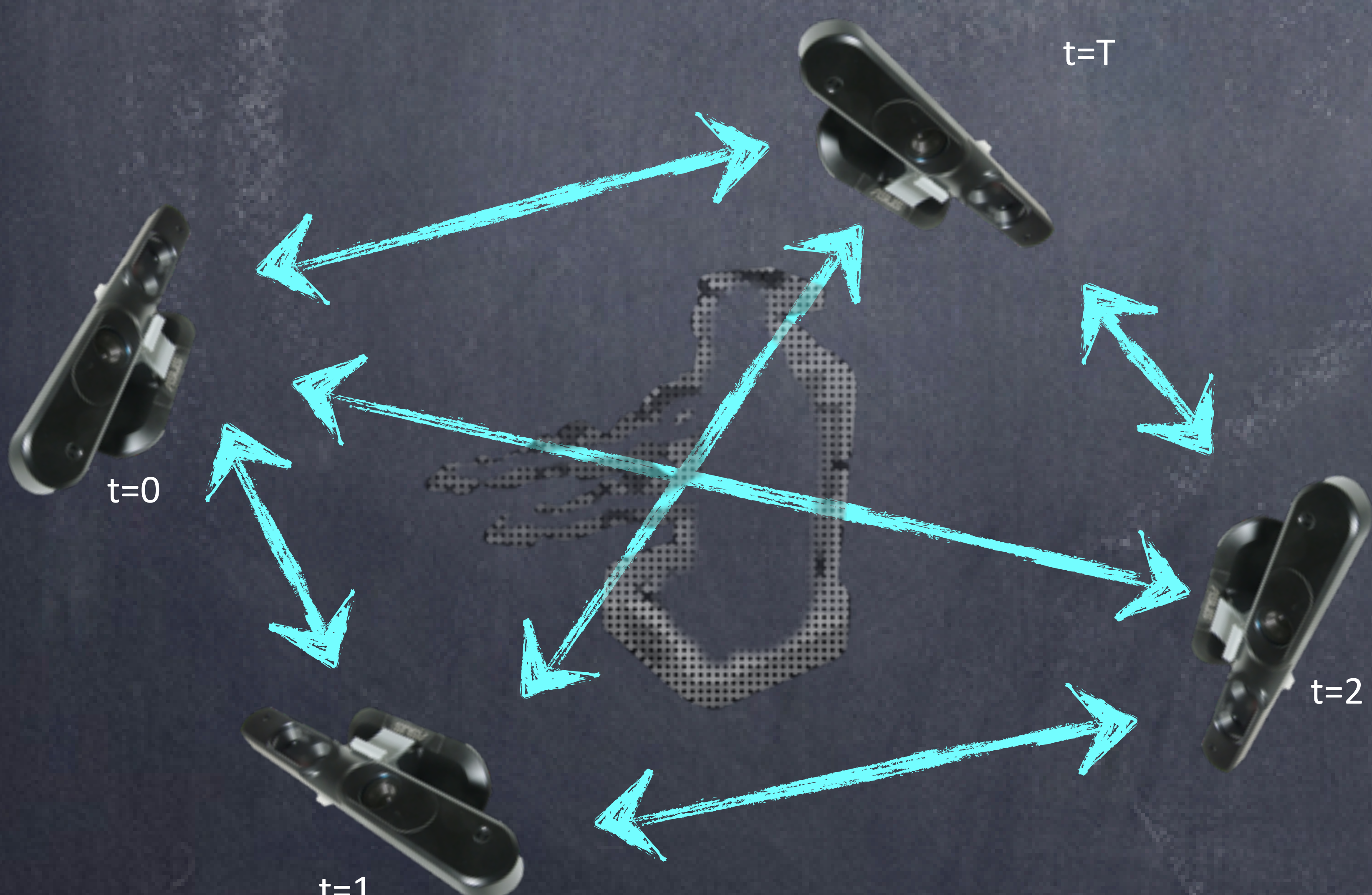
- Surface Reconstruction
- Model (“Map”) is the goal
- Pose is [roughly] given; shape used to refine
- Fairly precise sensors
- Would much rather throw out bad data than use it

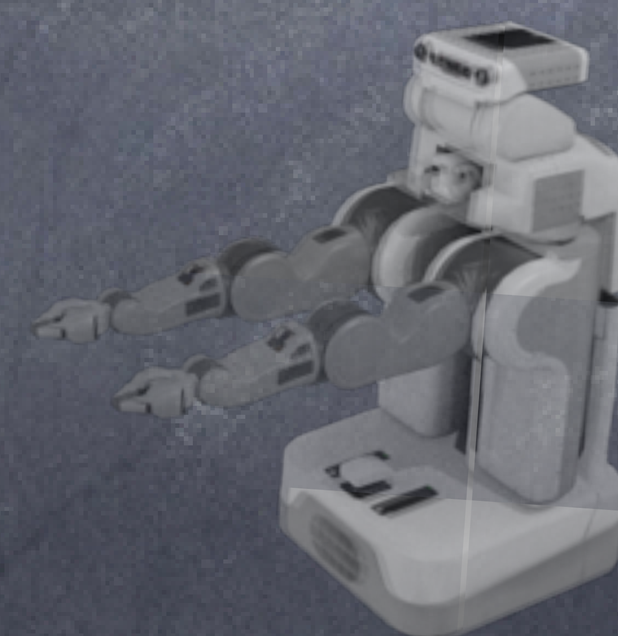
A tale of two modules

Registration

Reconstruction

- How to scan lines align?
- Should I use this data or not?
- What surface would have created these scan lines?
- Average out noise

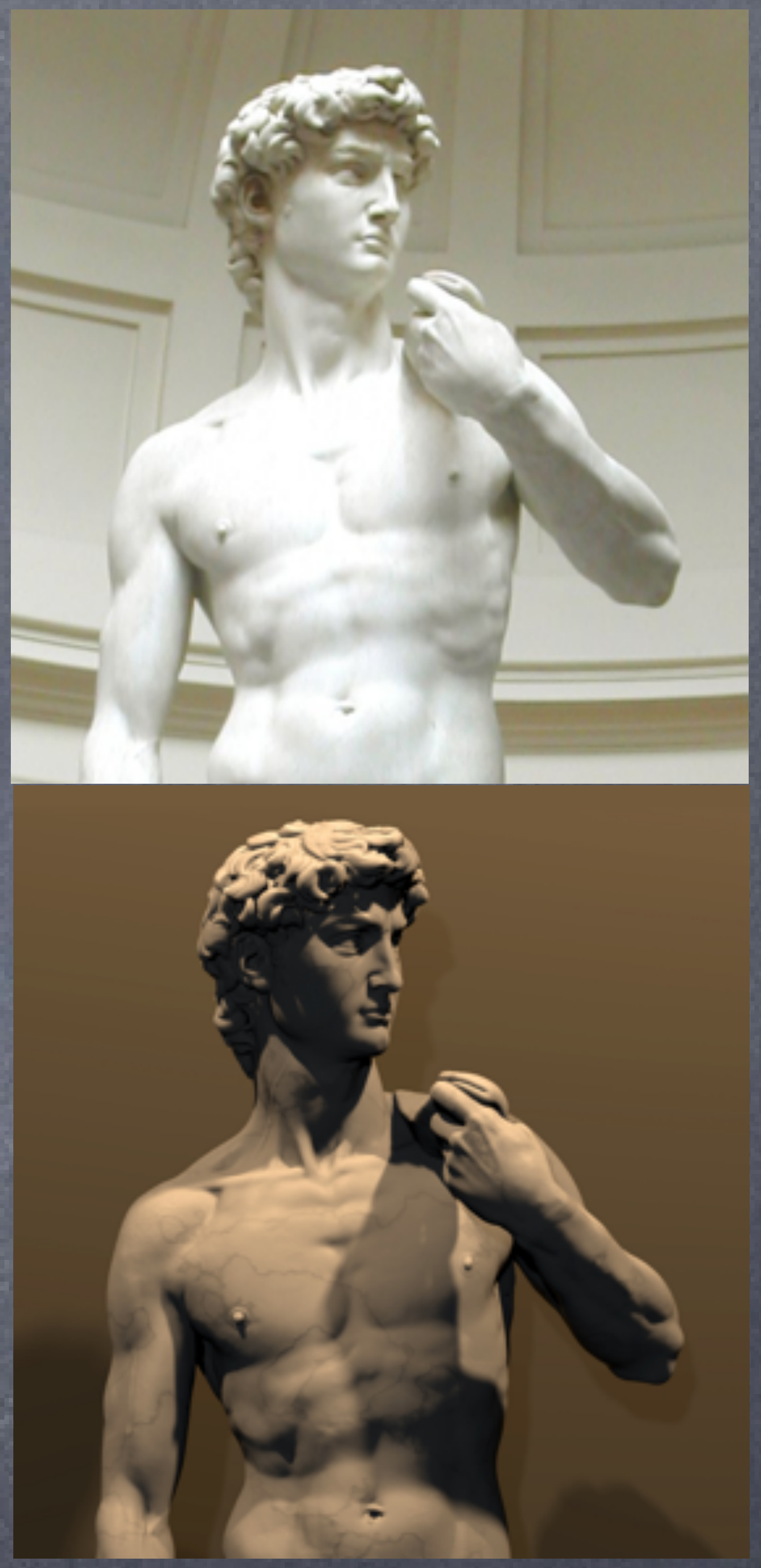




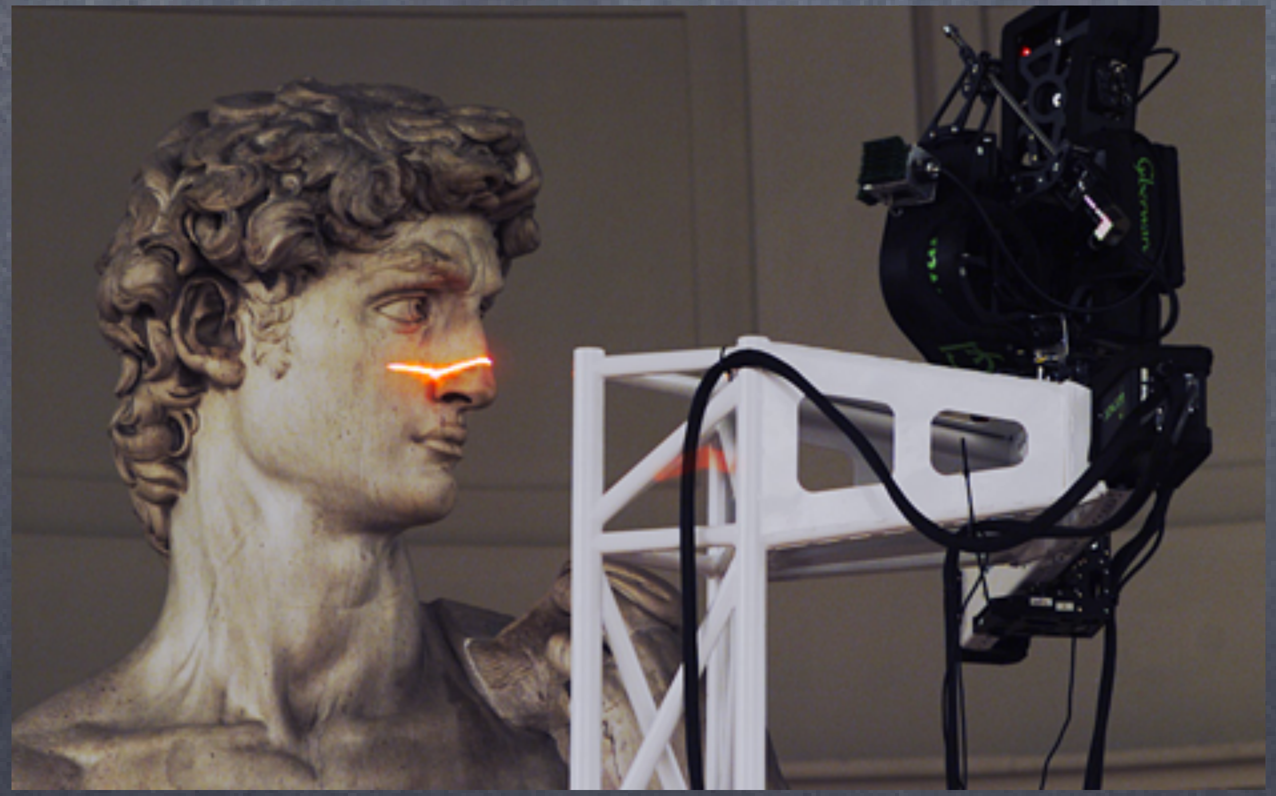
The Michelangelo Project

- Input: Absolutely any non-intrusive sensor
- Output: Digital, high-fidelity surface model
- We roboticists often assume this step is easy; it isn't!

reality, Florence

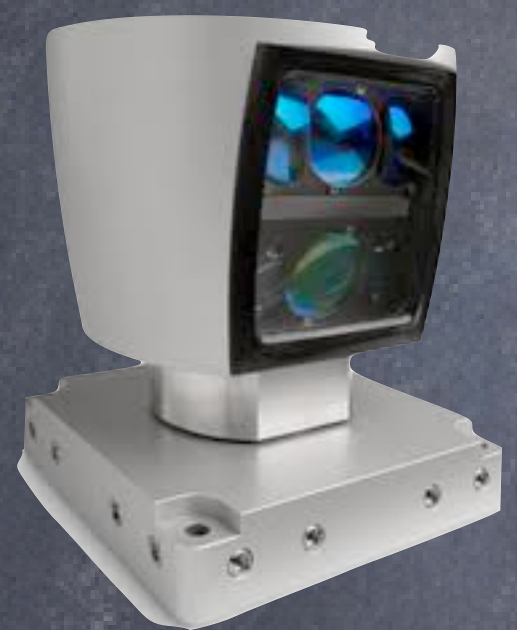


digital, Levoy's website



Why is this hard?
Lots of noisy Data

Velodyne HDL-64E
1.3M points per second

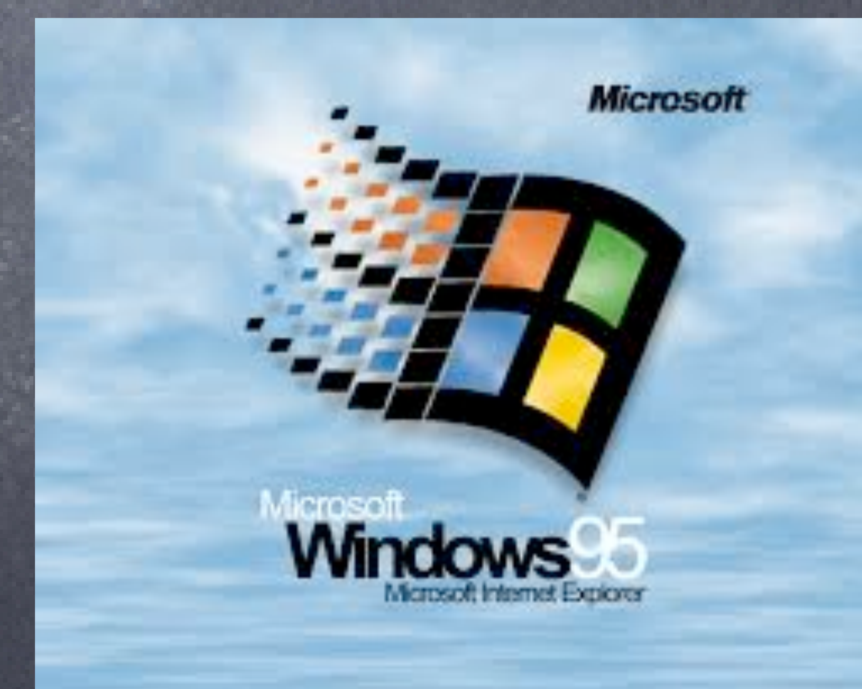


Kinect
9.2M points per second

Slowly scan every crevice
>= 30 minutes

X

>= 26 GB (HDL)
>= 62 GB (Kinect)



Intro → Reconstruction Basics → KinectFusion → Current/Future

Curless & Levoy, 1996

- Popularized an efficient data structure for keeping track of visibility information (the TSDF)
- Gave online method for building this data structure from (basically) any depth sensor at known pose
- If depth readings are Gaussian, approximation is lossless (assuming we need to cram it into this data structure)

A Volumetric Method for Building Complex Models from Range Images

Brian Curless and Marc Levoy
Stanford University

Abstract

A number of techniques have been developed for reconstructing surfaces by integrating groups of aligned range images. A desirable

images into a single description of the surface. A set of desirable properties for such a surface reconstruction algorithm includes:

- *Representation of range uncertainty.* The data in range images

Surface Representation?

- How should we represent a continuous surface?

- Discrete triangles don't quite capture it

- Ideally, it's a curve; a function

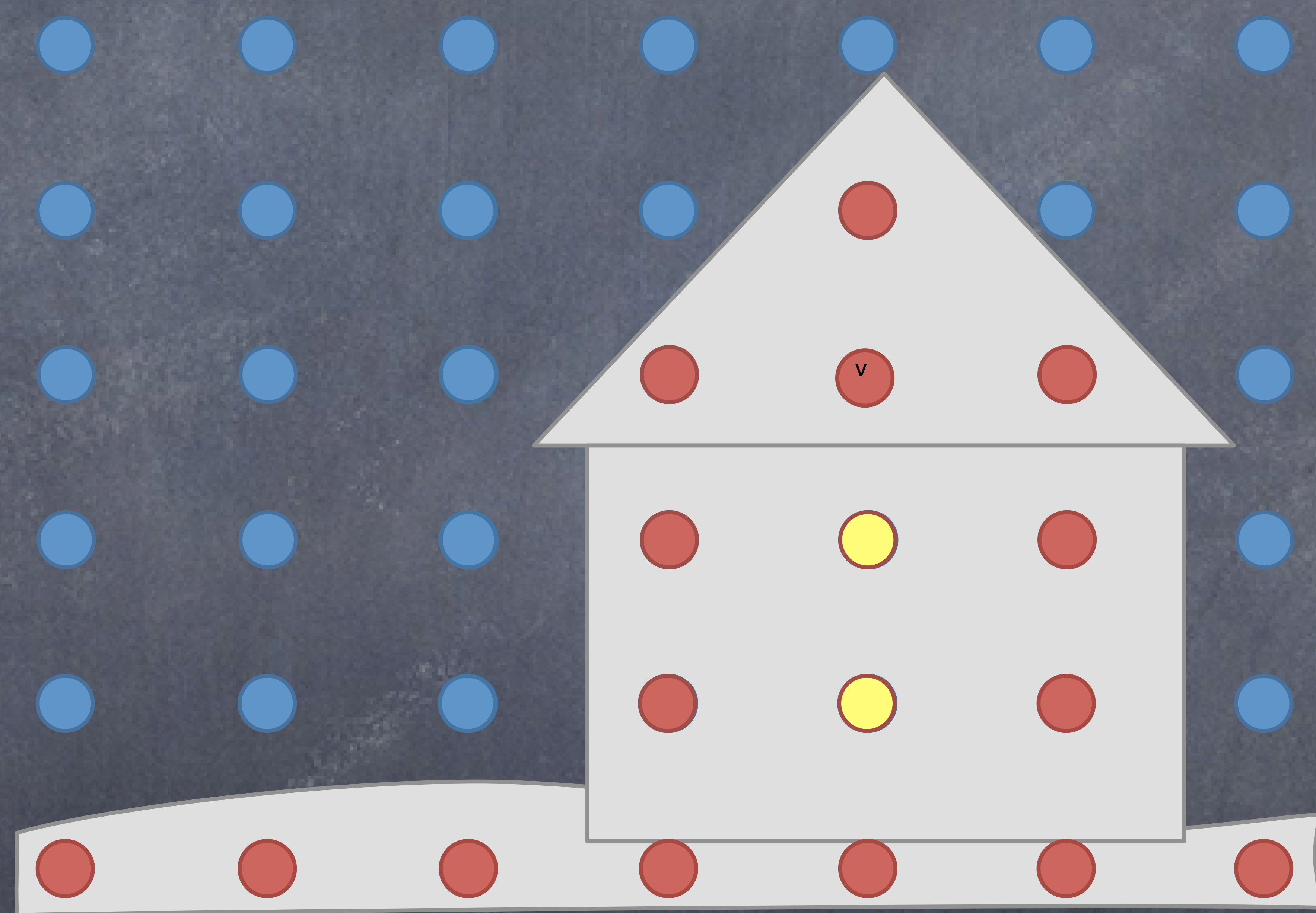
$$f(\vec{x}) \in \{\mathbb{R}^3 \to \mathbb{R}\}$$

- Must be parametrized to be learned

Recap: Occupancy Grid

- Binary: where is safe to move? (Optional: where haven't I seen?)

$$f(\vec{x}) \in \{0, 1, ?\}$$



0: Unoccupied

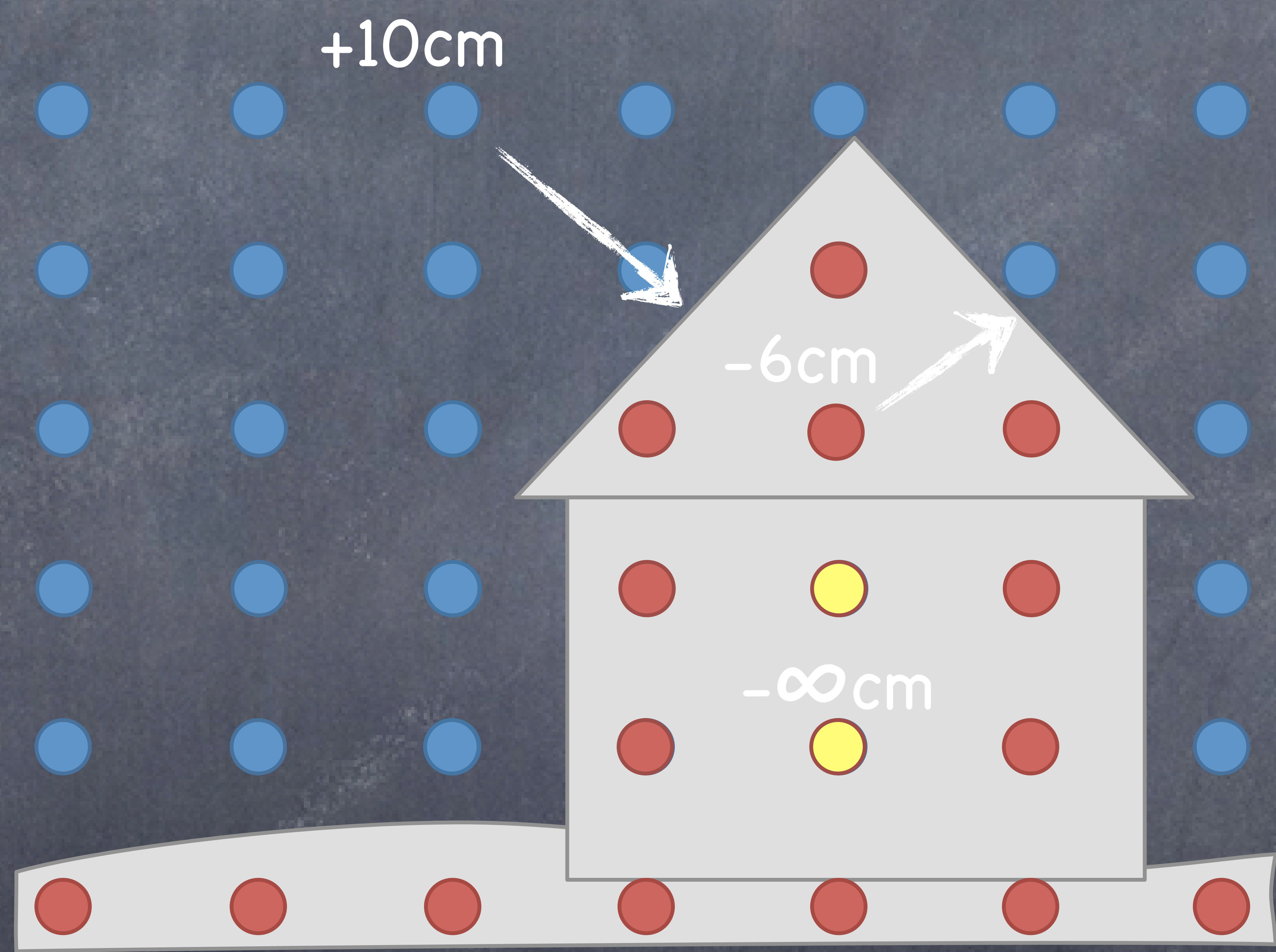
1: Occupied

?: Unseen

Signed Distance Field

- Floating point: how far am I from the surface?

$$f(\vec{x}) \in (-\infty, \infty)$$

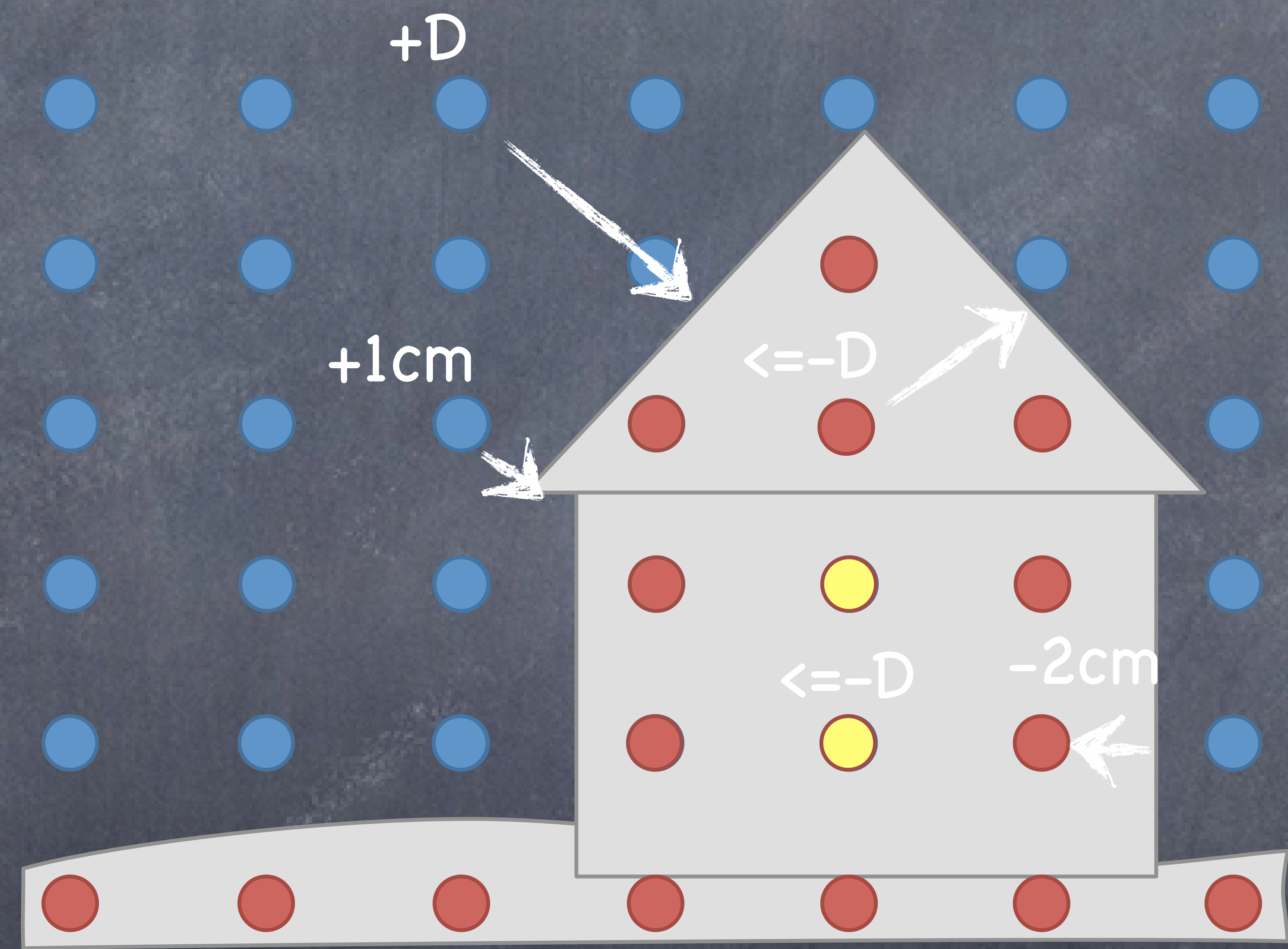


- >0: Outside
- <0: Inside
- $-\infty$: Unseen

Truncated Signed Distance Field

- Floating point: how far am I from the surface if I'm close

$$f(\vec{x}) \in [-D, +D]$$



0<T<=D: Outside

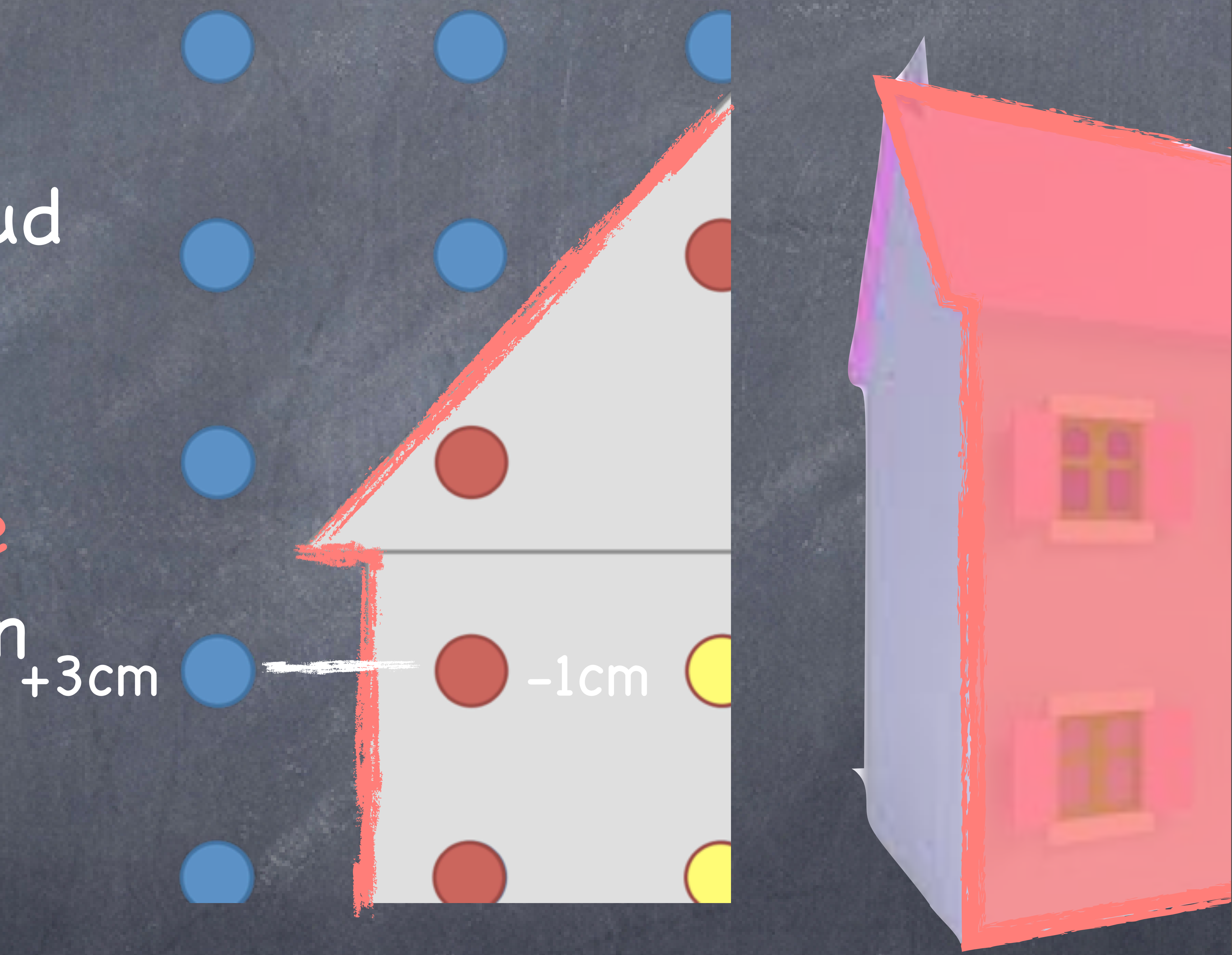
-D<T<0: Inside

-D: Unseen

- Truncation limit D should approximate sensor noise

TSDF --> Mesh

- $f(\vec{x}) \in \{\mathbb{R}^3 \rightarrow \mathbb{R}\}$ is called an "implicit surface function"
- Can easily turn into a mesh or cloud
  - Surface = $\{\vec{x} \in \mathbb{R}^3 | f(\vec{x}) = 0\}$ (where is my distance 0?)
  - Called 0 crossing, or 0 isosurface
  - Estimate by trilinear interpolation
- Or add padding
  - iso level d : $\{\vec{x} \in \mathbb{R}^3 | f(\vec{x}) = d\}$

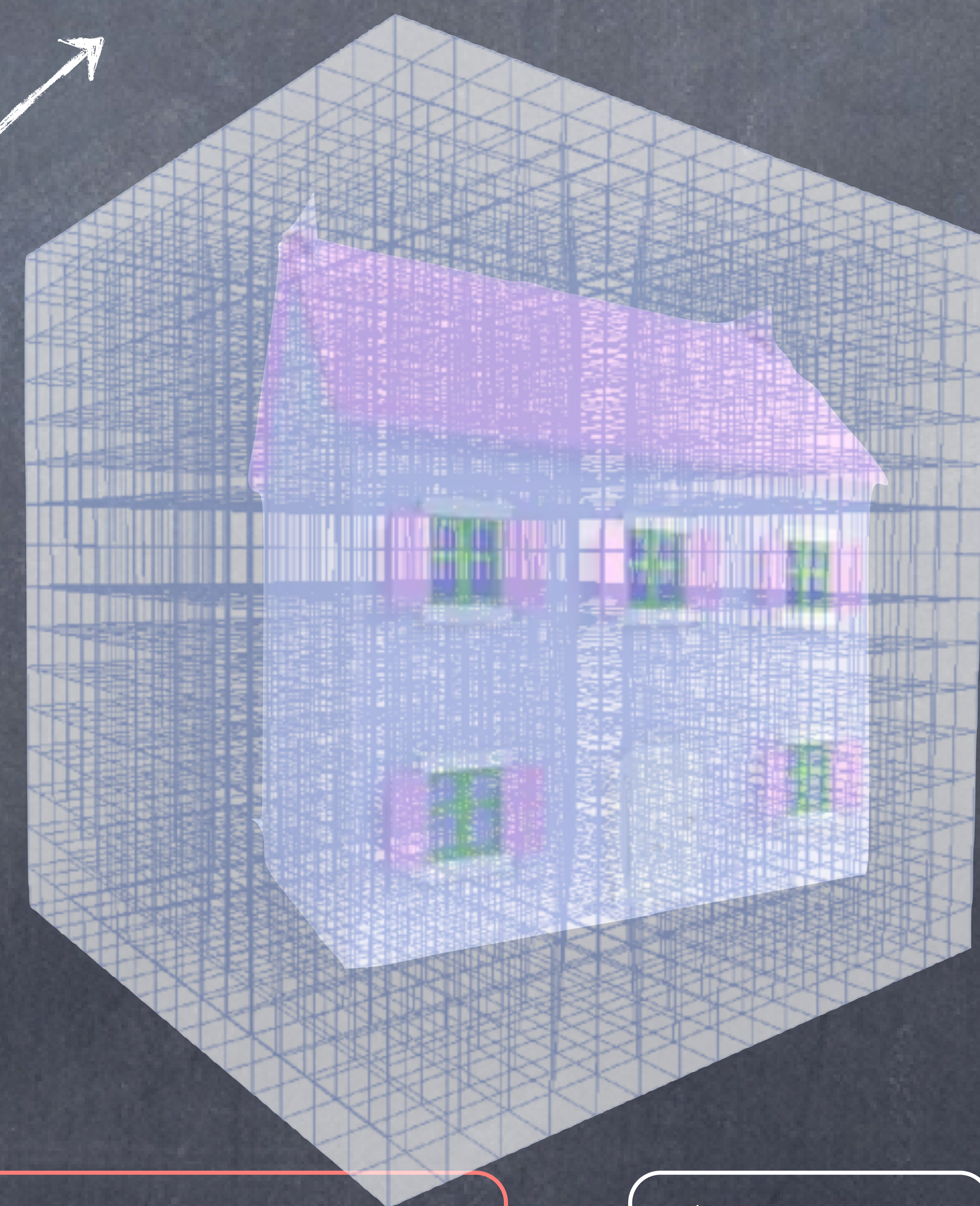


TSDF > Mesh

Mesh: only stores observed surface

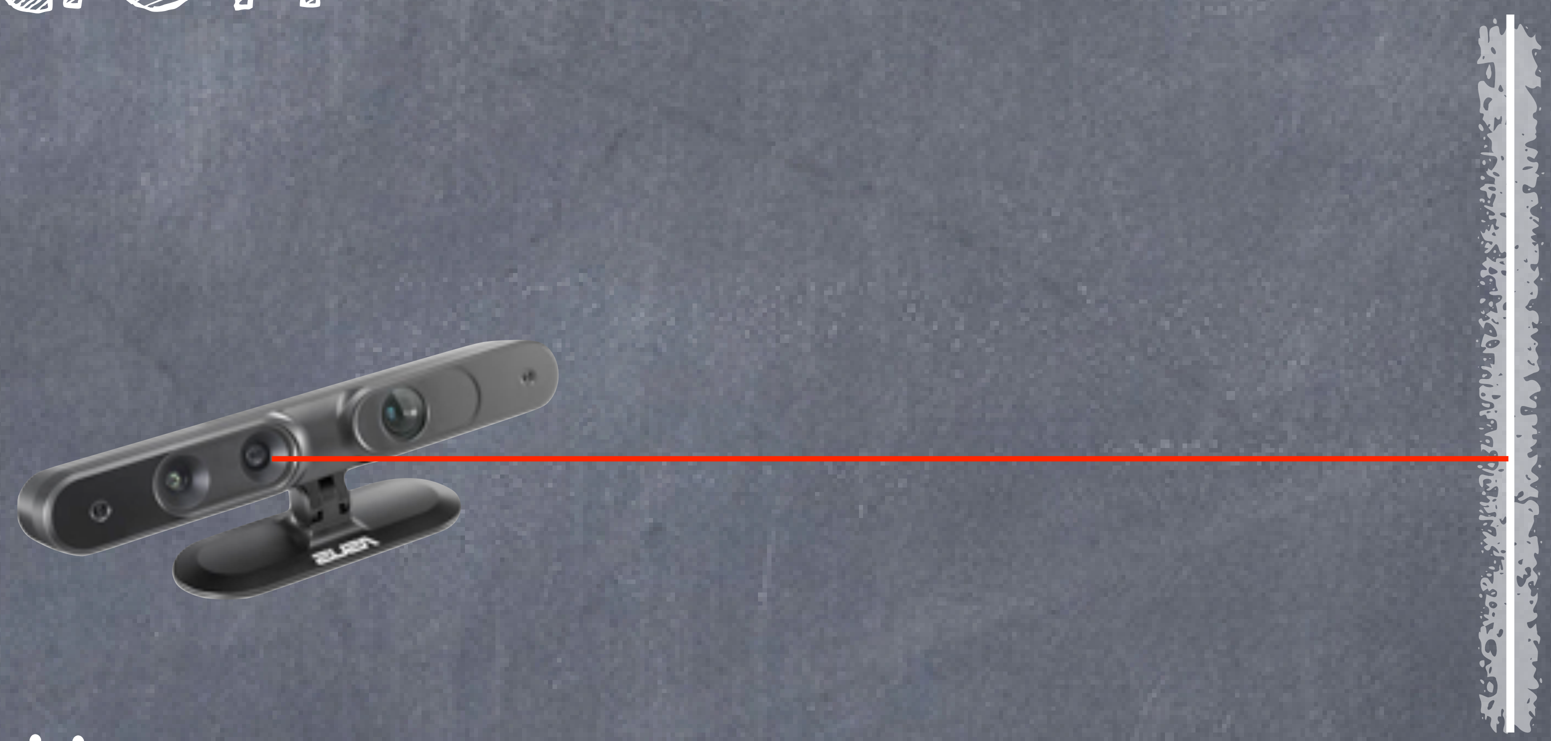
TSDF: also stores observed non-surface

Useful for registration!



SDF Estimation

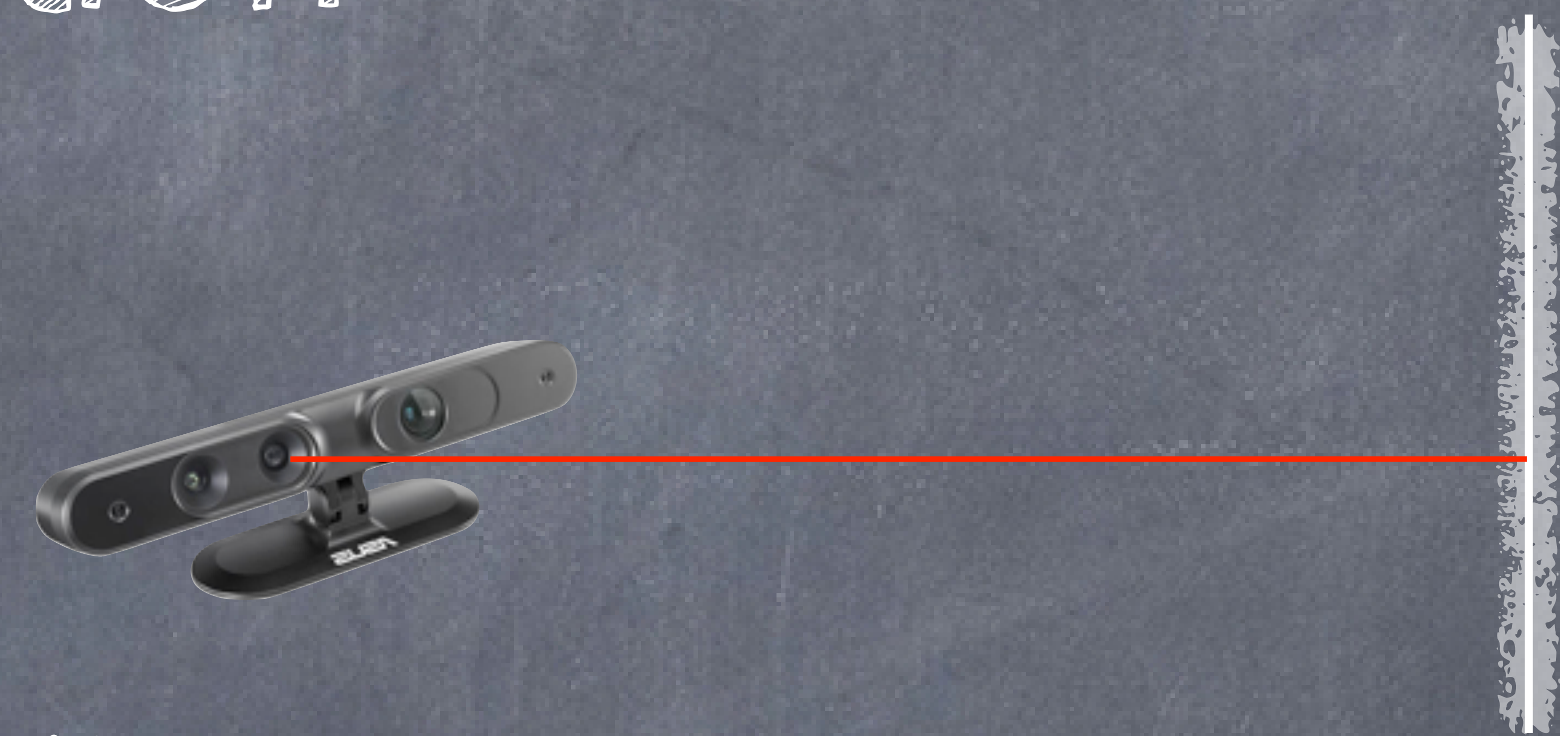
- Simple example: stationary sensor



- How far is the wall ("d_w")?
  - Assuming Gaussian noise, it will be the average

$$d_w = \frac{1}{T}\sum_t z_t$$

SDF Estimation



- Simple example: stationary sensor
- How far is the wall ("d_w")?
  - Assuming Gaussian noise, it will be the average

$$d_w = \frac{1}{T}\sum_t z_t$$

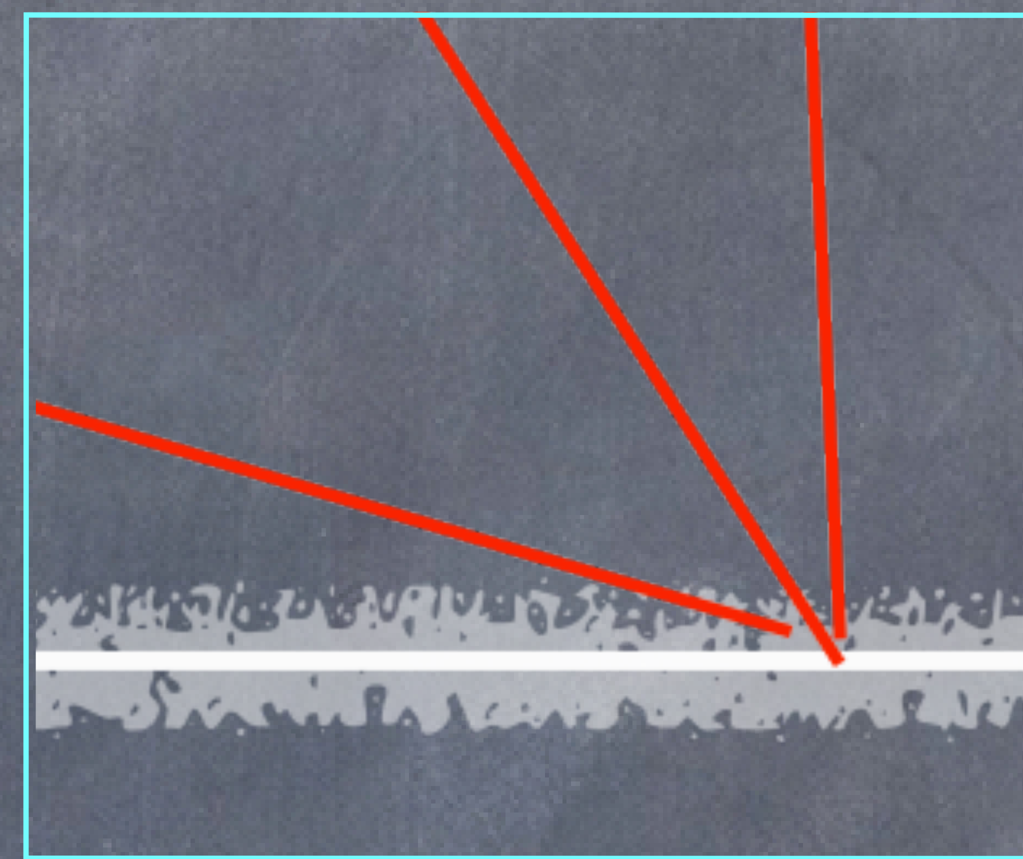
- How far is some point ("$\vec{x}$") from the wall?

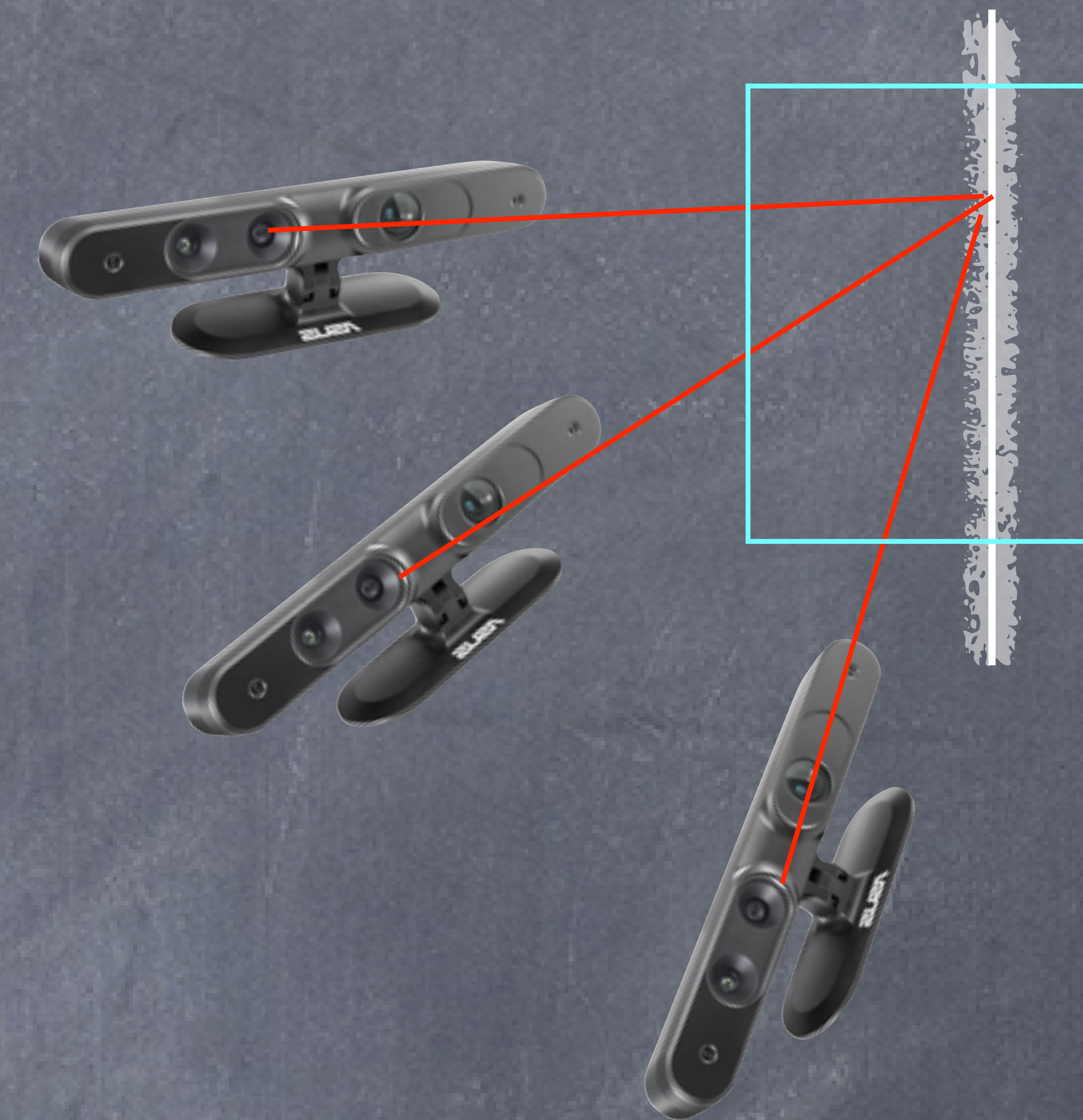
$$d_{\vec{x}} = \vec{x} - d_w = \vec{x} - (\frac{1}{T}\sum_t z_t) = \frac{1}{T}\sum_t (\vec{x} - z_t)$$

AVERAGE ALL READINGS WHICH PASS THROUGH $\vec{x}$

SDF Estimation

- What if we angle it?
- Readings are less trustworthy at an angle
- Simple averaging is not sufficient; need to weight samples





$$d_{\vec{x}} = \frac{1}{\sum_t w_t} \sum_t w_t (\vec{x} - z_t)$$

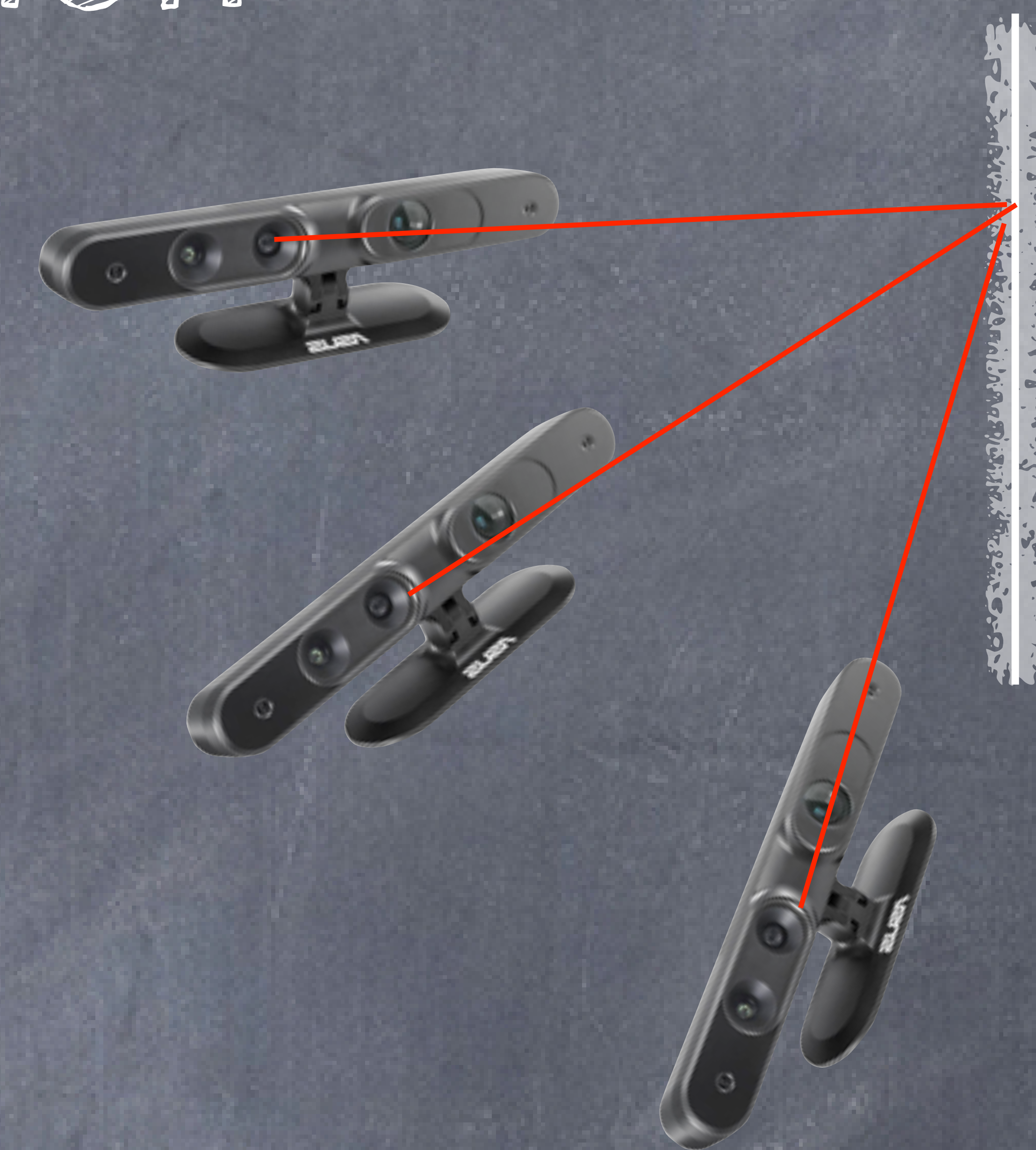
JUST A COMMON CHOICE; COULD BE FANCIER

$$w_t \propto \underline{r_t} \cdot n_t$$

DIRECTION OF RAY

Online SDF Estimation

$$d_{\vec{x}} = \frac{1}{\sum_t w_t} \sum_t w_t(\vec{x} - z_t)$$



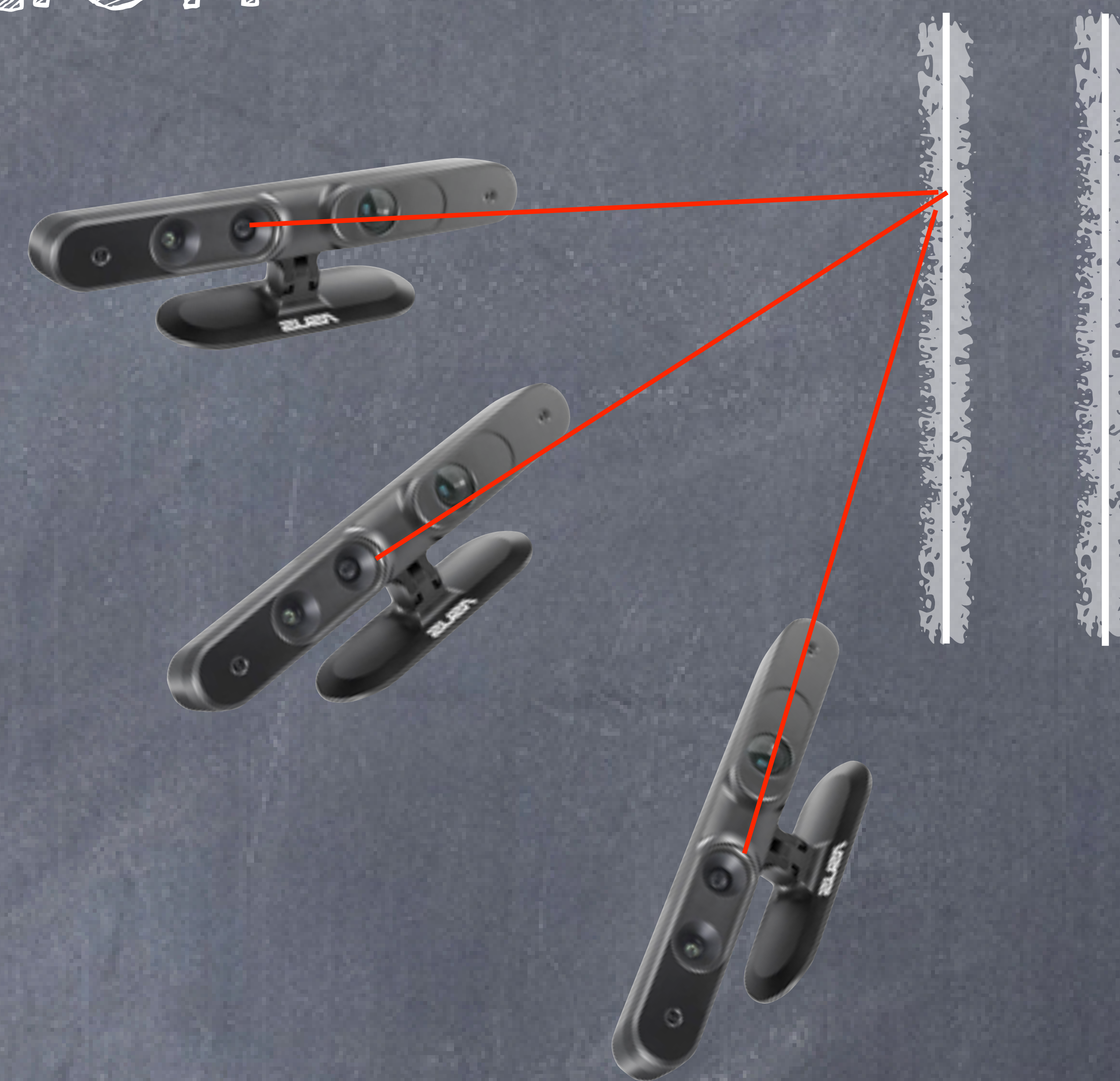
- Weighted averaging can be done online

$$d_{\vec{x}} \leftarrow \frac{w_t(\vec{x} - z_t) + w_{\vec{x}} d_{\vec{x}}}{w_t + w_{\vec{x}}}$$

$$w_{\vec{x}} \leftarrow w_t + w_{\vec{x}}$$

Online TSDF Estimation

$$d_{\vec{x}} = \frac{1}{\sum_t w_t} \sum_t w_t h(\vec{x} - z_t)$$



- Weighted averaging can be done online

$$d_{\vec{x}} \leftarrow \frac{w_t h(\vec{x} - z_t) + w_{\vec{x}} d_{\vec{x}}}{w_t + w_{\vec{x}}}$$

$$w_{\vec{x}} \leftarrow w_t + w_{\vec{x}}$$

- Truncate so surfaces don't interfere $h(x) = \begin{cases} +D & x > D \\ x & -D \leq x \leq +D \\ -D & x < -D \end{cases}$

- Initialize as unknown: $d_{\vec{x}} \leftarrow 0, w_{\vec{x}} \leftarrow 0$

Kinect Fusion

Intro → Reconstruction Basics → KinectFusion → Current/Future

1996 to 2011: Theory

Object Recognition from Local Scale-Invariant Features

David G. Lowe
Computer Science Department
University of British Columbia
Vancouver, B.C., V6T 1Z4, Canada
lowe@cs.ubc.ca

SIFT '99

The GraphSLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures

Sebastian Thrun
Michael Montemerlo
Stanford AI Lab
Stanford University
{thrun,mmde}@stanford.edu

GRAPHSLAM '06

FAB-MAP: Probabilistic Localization and Mapping in the Space of Appearance

Mark Cummins and Paul Newman
The International Journal of Robotics Research 2008 27: 647
DOI: 10.1177/0278364908090961

The online version of this article can be found at:
http://ijr.sagepub.com/content/27/6/647

Published by:
SAGE
http://www.sagepublications.com

On behalf of:
ijrr
Multimedia Archives

FAB-MAP '08

UKF '97

A New Extension of the Kalman Filter to Nonlinear Systems

Simon J. Julier — sijulier@robots.ox.ac.uk
Jeffrey K. Uhlmann — uhlmann@robots.ox.ac.uk
The Robotics Research Group, Department of Engineering Science, The University of Oxford
Oxford, OX1 3PJ, UK, Phone: +44-1865-282180, Fax: +44-1865-273908

FASTSLAM '02

FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem

Michael Montemerlo and Sebastian Thrun
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
mmde@cs.cmu.edu, thrun@cs.cmu.edu

Daphne Koller and Ben Wegbreit
Computer Science Department
Stanford University
Stanford, CA 94305-9010
koller@cs.stanford.edu, ben@wegbreit.com

ISAM '07

iSAM: Fast Incremental Smoothing and Mapping with Efficient Data Association

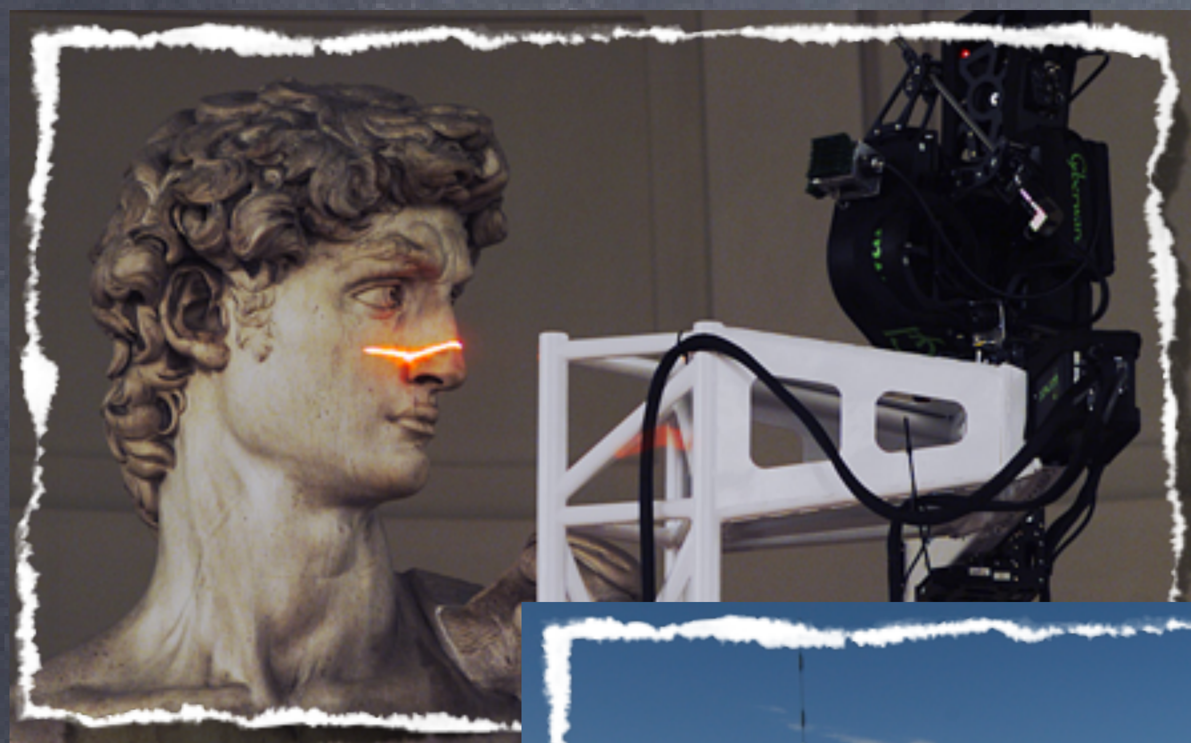
Michael Kaess, Ananth Ranganathan, and Frank Dellaert
Center for Robotics and Intelligent Machines, College of Computing
Georgia Institute of Technology, Atlanta, GA 30332
{kaess,ananth,dellaert}@cc.gatech.edu

RGBD-SLAM '11

Real-time 3D visual SLAM with a hand-held RGB-D camera

Nikolas Engelhard[a] Felix Endres[a] Jürgen Hess[a] Jürgen Sturm[b] Wolfram Burgard[a]

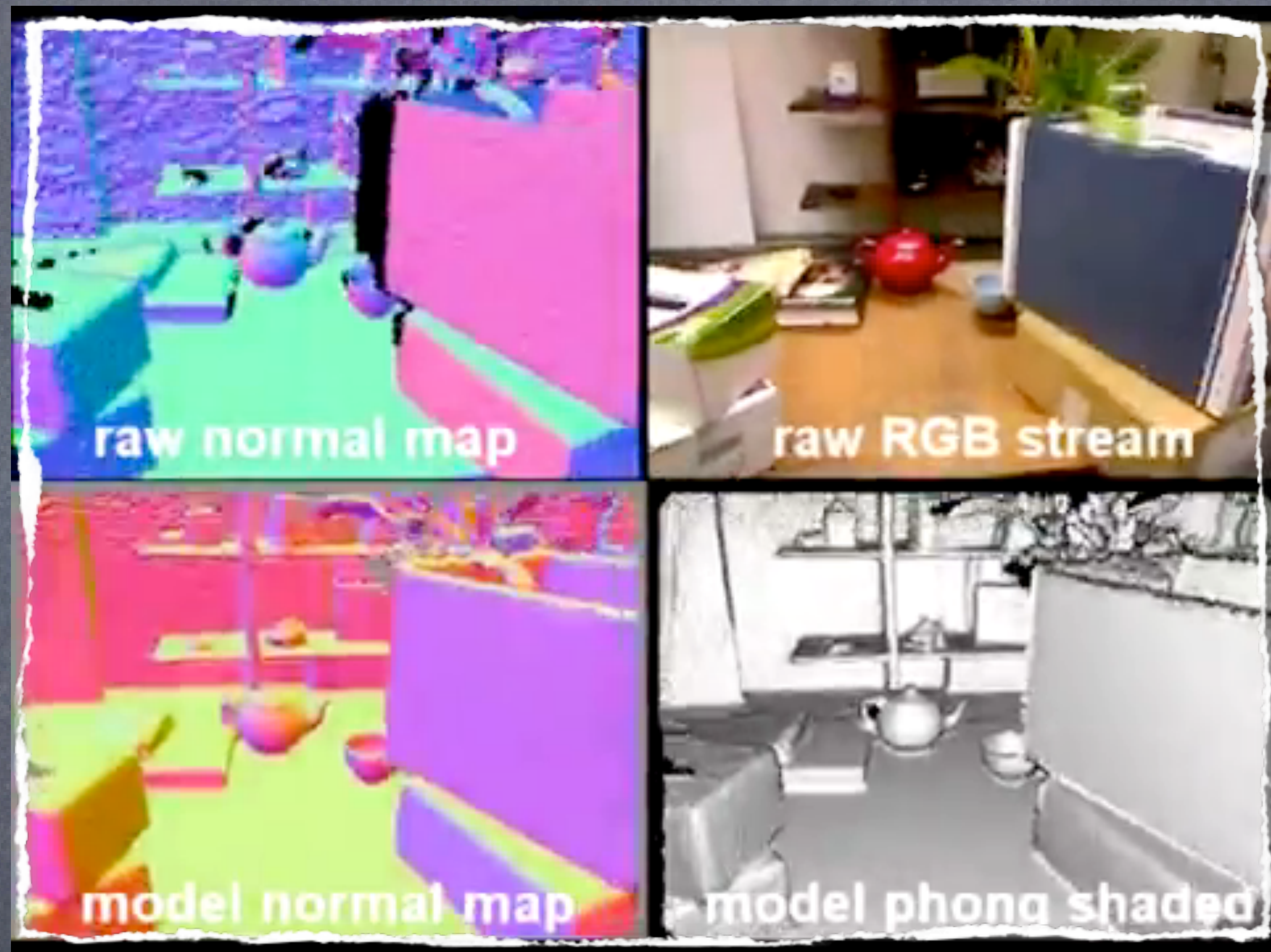
1996 to 2011: Technology



Inexpensive, noisy data

Powerful Processing

Kinect Fusion



Newcomb et al, ISMAR '11

Izadi et al, SIGGRAPH '11

How it works

(Align current cloud to rendered cloud)
Register

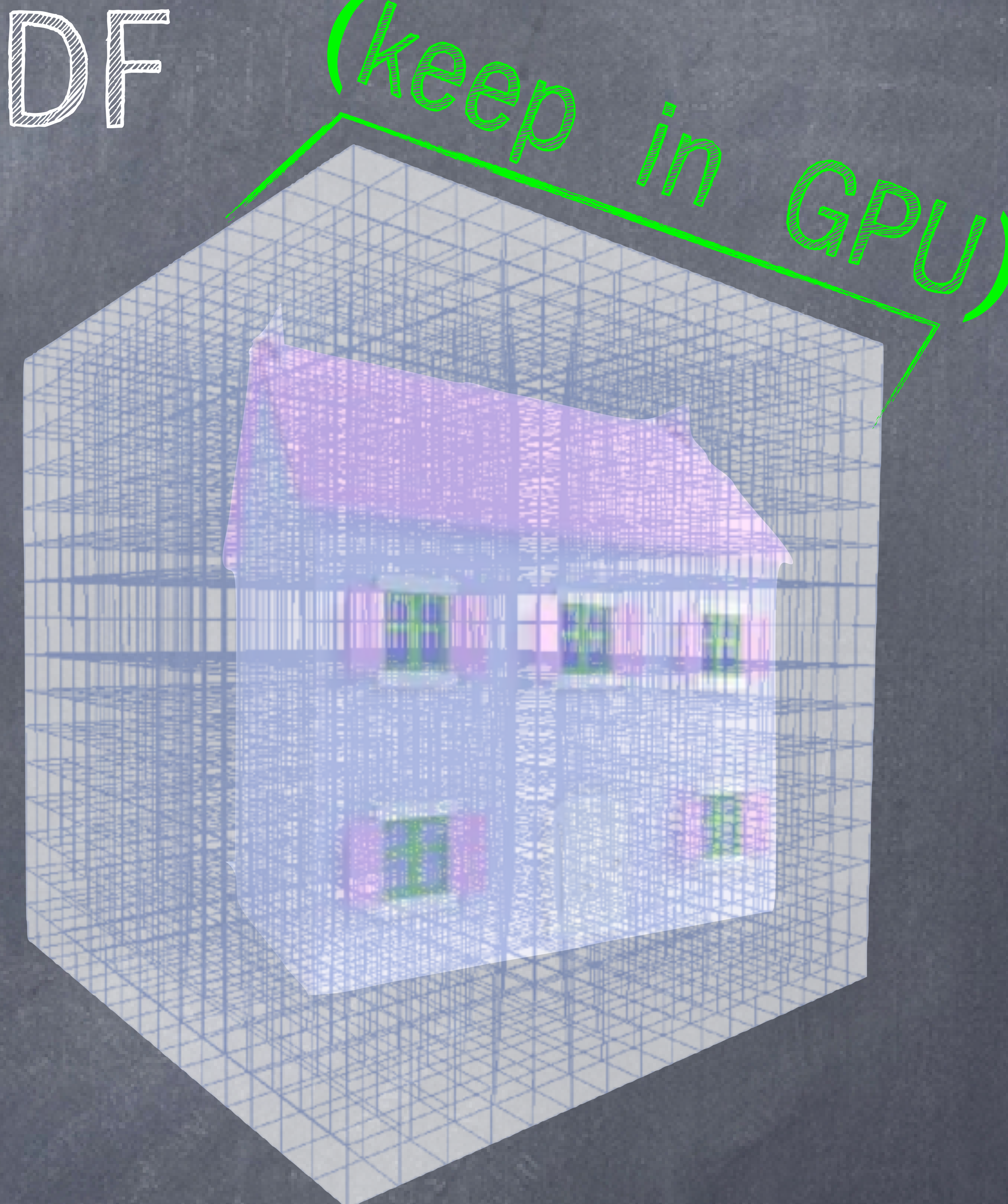
"frame to model registration"

Integrate
(Induct new cloud into model)

(Output triangle mesh)
Extract

Render
(Build synthetic depth map from model)

Integration: TSDF



- Curless & Levoy:

$$d_{\vec{x}} \leftarrow \frac{w_t h(\vec{x} - z_t) + w_{\vec{x}} d_{\vec{x}}}{w_t + w_{\vec{x}}}$$

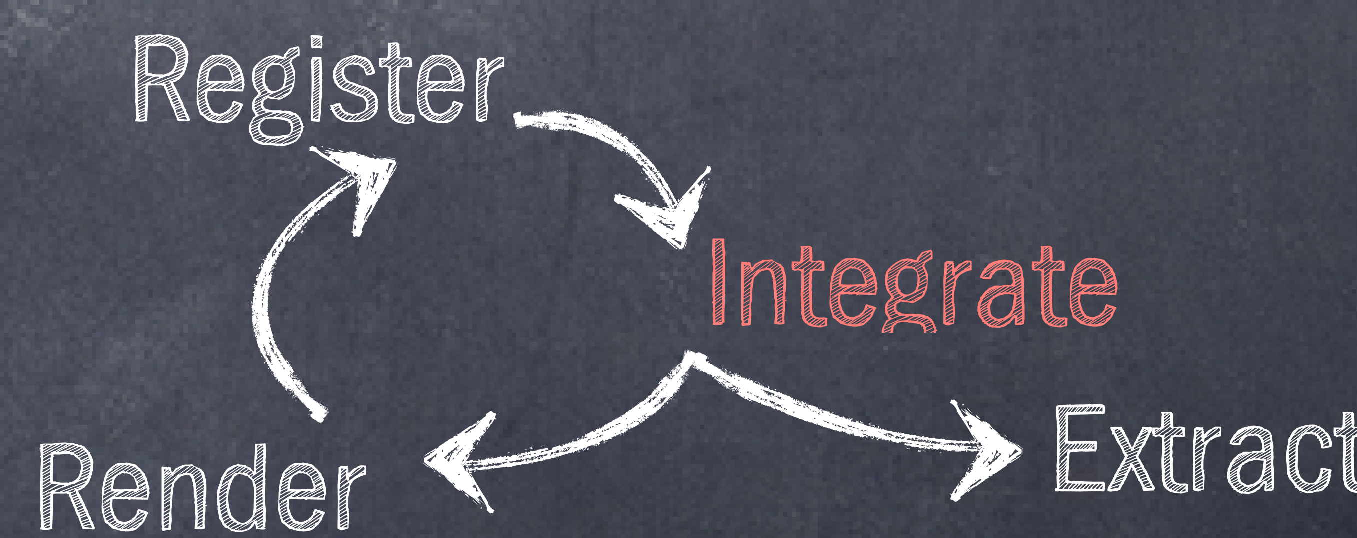
$$w_{\vec{x}} \leftarrow w_t + w_{\vec{x}}$$

- "Simple works well enough"

$$w_t = 1$$

$$D_{max} = D_{min} = 3cm$$

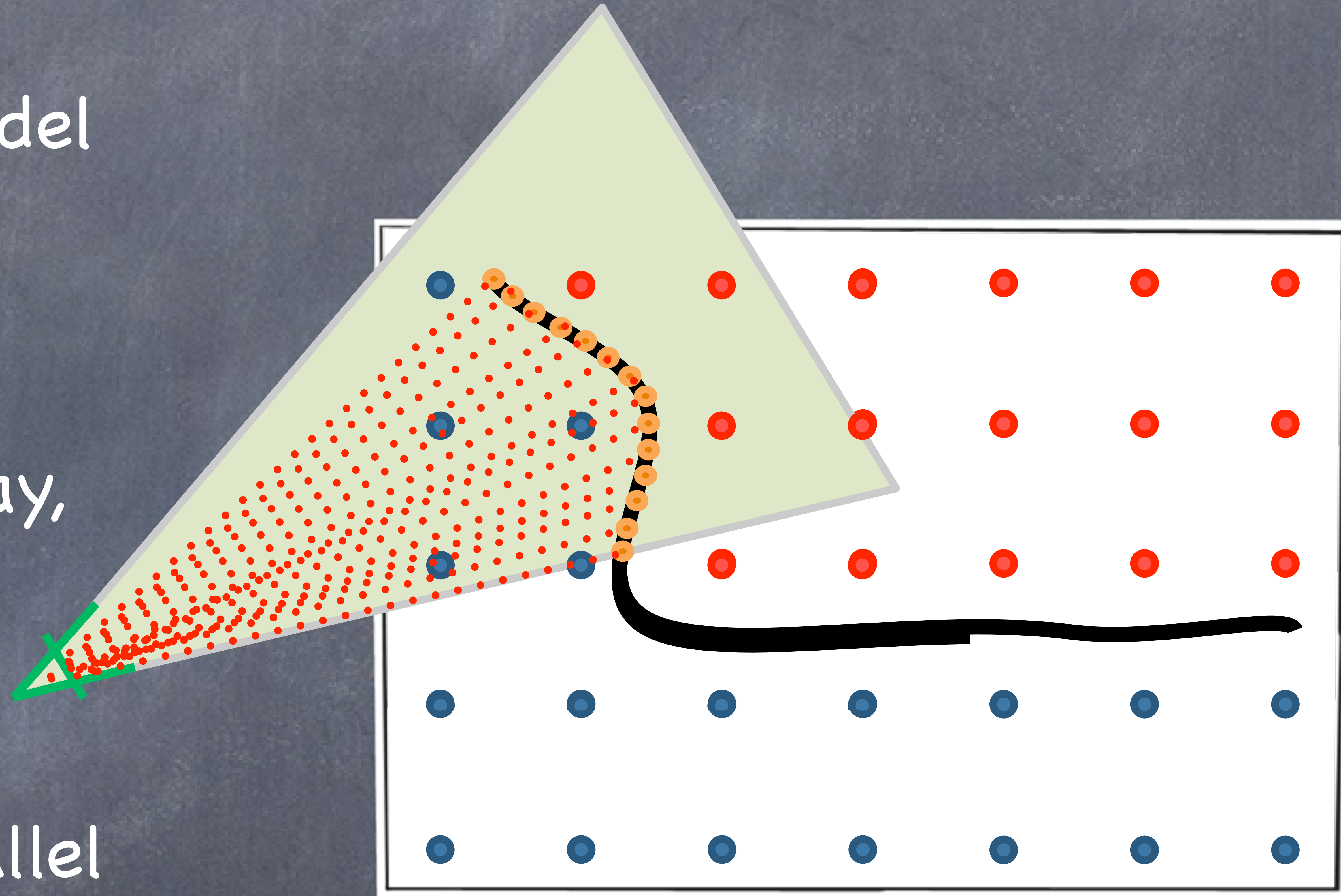
- March over all voxels in parallel; project center to depth map pixels

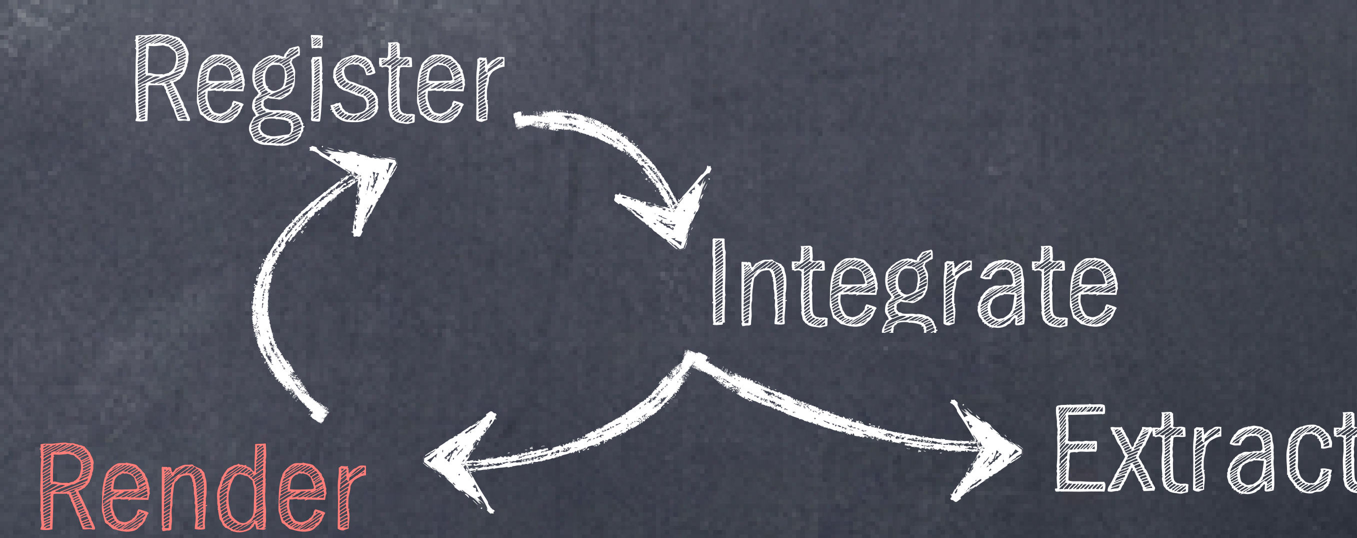


Rendering: Raycasting

- See what the current model <u>should</u> look like from the current vantage point
- March along each pixel ray, stop when TSDF changes signs
- All pixels checked in parallel

(run in GPU)



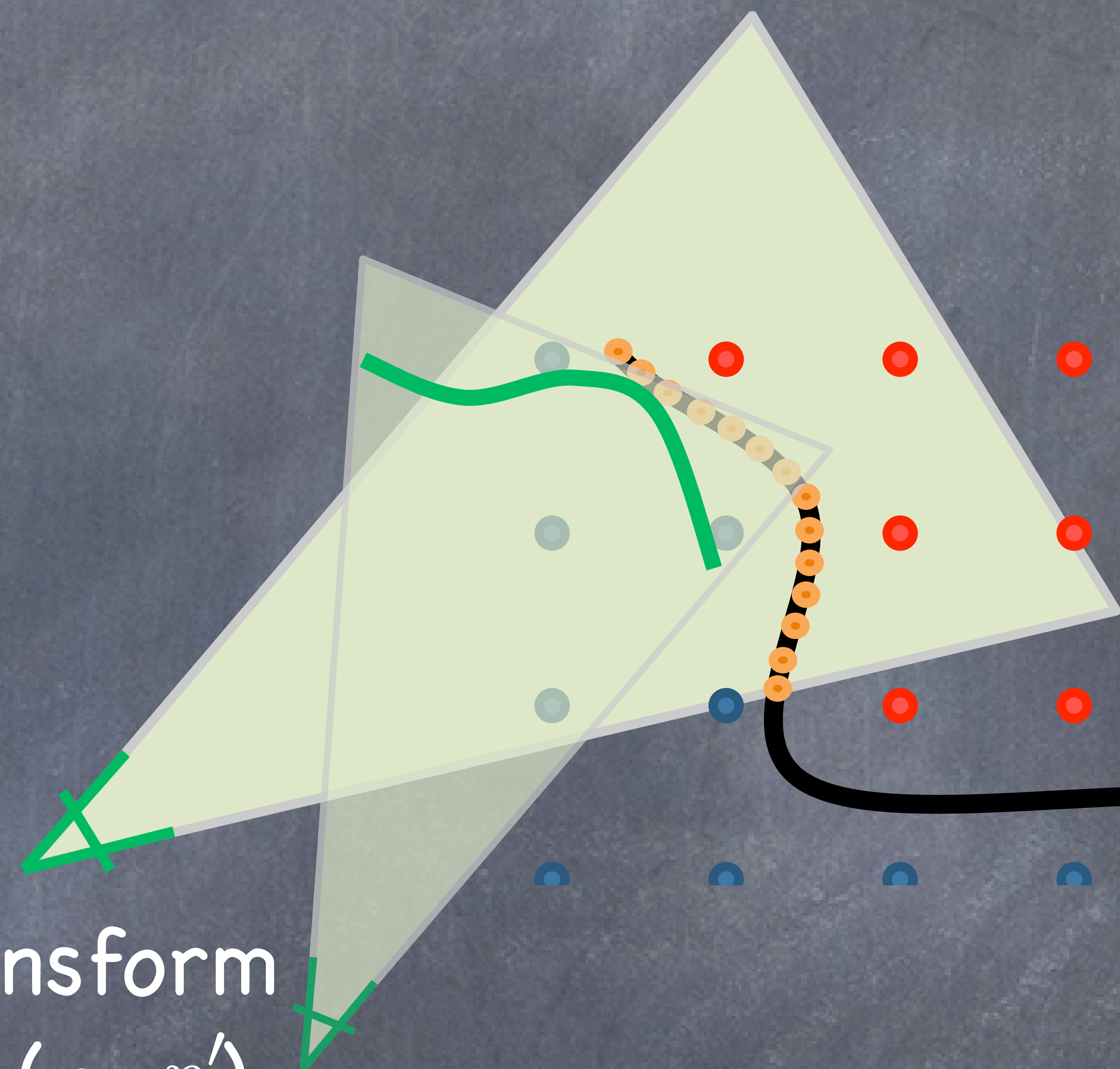


Registering: Point-to-Plane ICP

- Approximates:

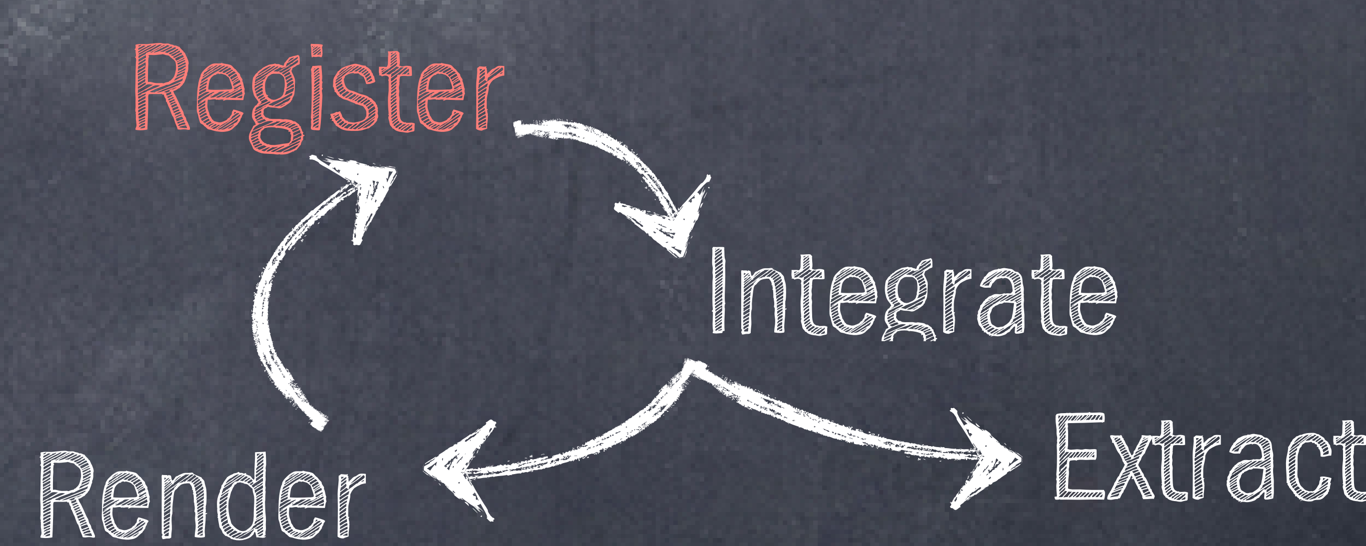
$$\min_T \sum_p (\min_{p'} ||Tp - p'||_2^2)$$

for p' drawn from continuous surface



- Alternatively optimize Transform (T) and Correspondences ($\approx p'$)

- Distance from surface $\approx$ distance from nearest point along its normal direction ($|(p-p')\cdot n'|$)



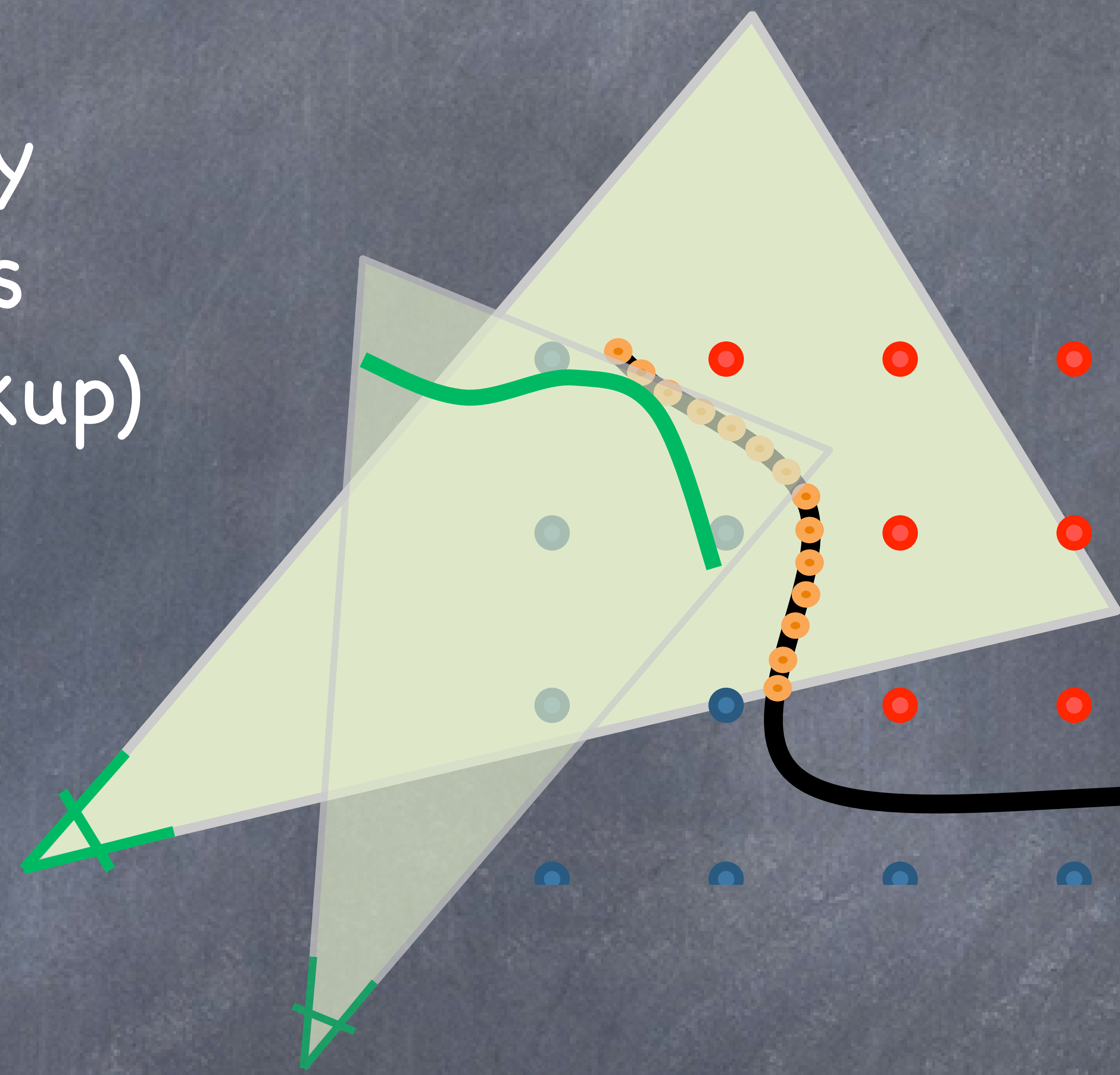
Registering: Point-to-Plane ICP

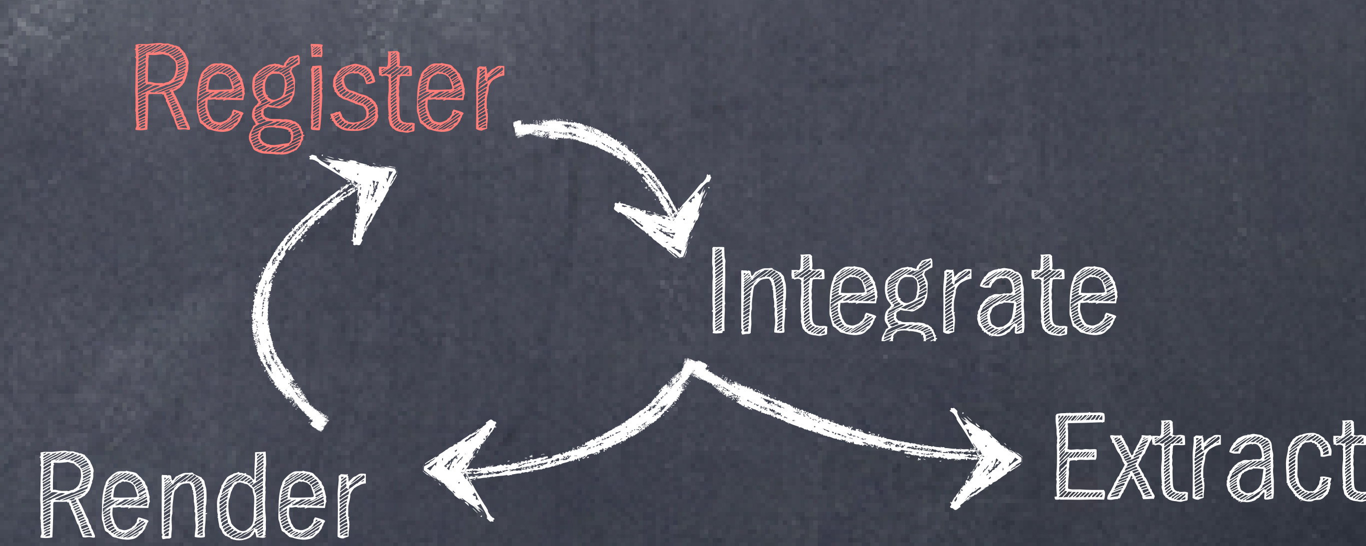
- Correspondences found by reprojecting depth images (no Nearest Neighbor lookup)

  (GPU-friendly)

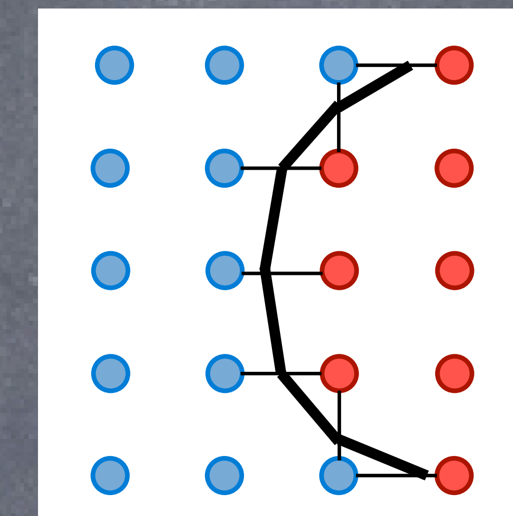
- Transform found by linearizing rotation matrix and solving linear system

  (GPU-friendly)

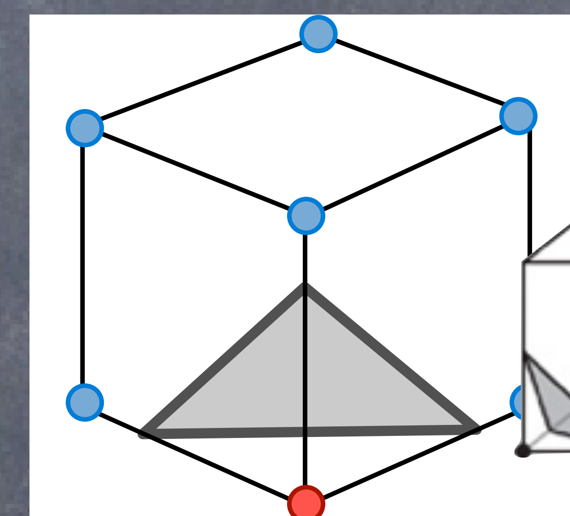


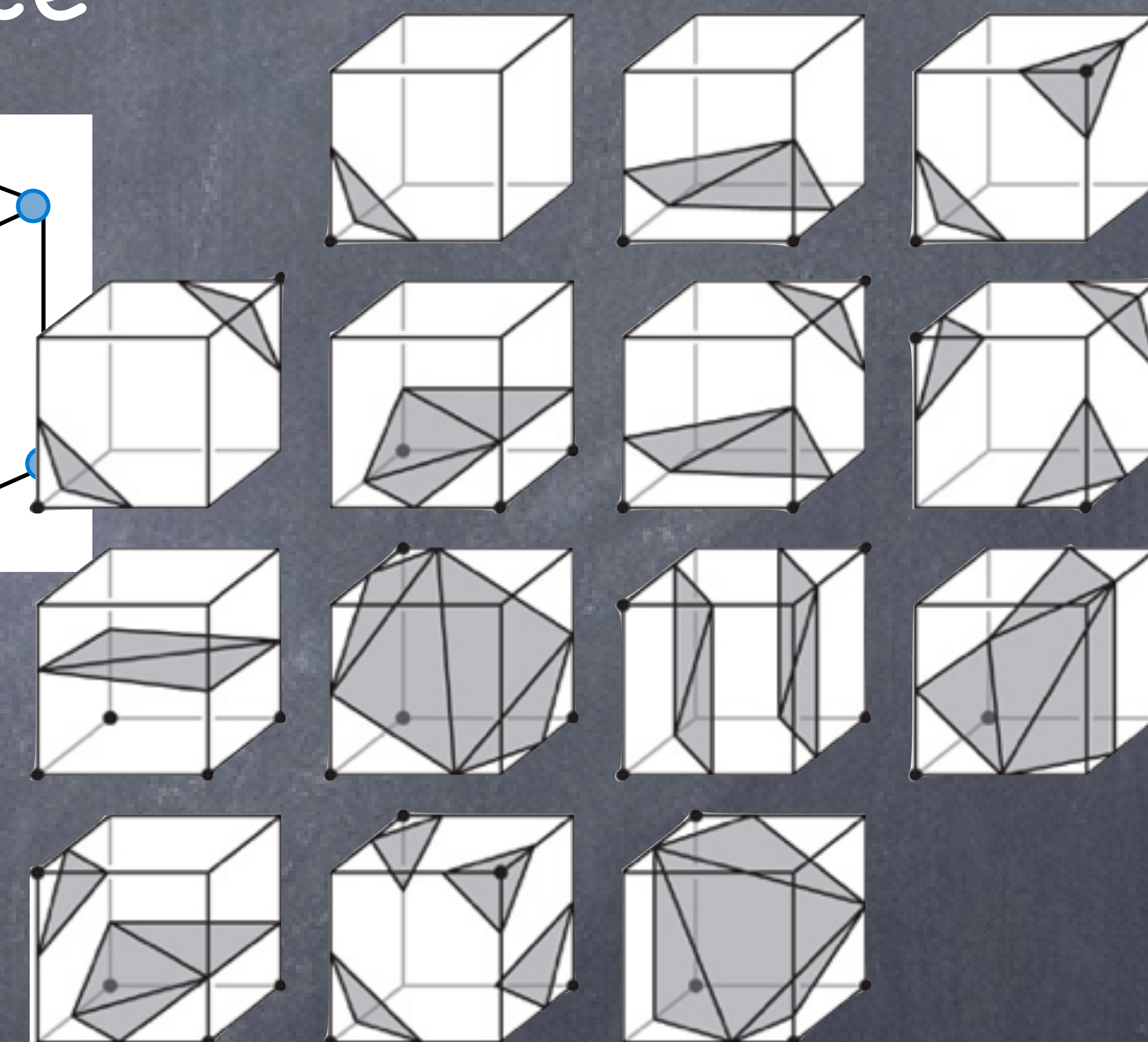


Extraction: Marching Cubes



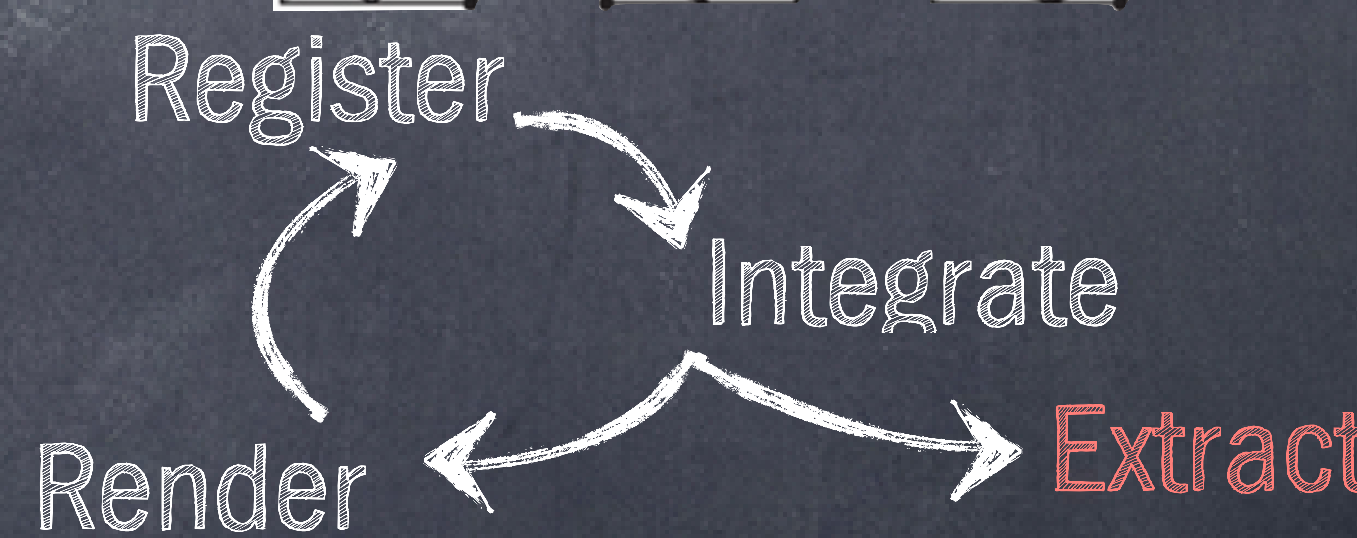
- Classic method for finding 0 crossing
- Voxel centers have positive or negative distance
- 8 neighboring voxels makes a "cube"





- Lookup table for all (2^8) possibilities,
  use actual distance values to refine position
- All cubes done in parallel

(GPU-friendly)



Overview

(Point-to-plane ICP with projective correspondences)

Register

Simple, known technique

All on GPU

=

Realtime

Simple, known technique

Integrate

(Online TSDF integration)

(Run marching cubes)

Extract

Render

(Raycast from the TSDF)

Simple, known technique

Current / Future

Three Big Takeaways from "KinFu"

Abundance of data > Sophistication of technique

If every component is simple, we can afford to use all data all the time

"Frame-to-model" > "Frame-to-frame"

For noisy pointclouds, it's important to have something clean to align to

Volumetric > Sparse pointclouds

To get clean surfaces, don't throw out information. Empty space is an observation.

Three Big Problems with "KinFu"

SCALABILITY

- TSDF had to fit in GPU
  Max 3mx3mx3m for most of us
- Process all voxels at all timesteps

SOLVABLE!

PRECISION

- ICP has flaws (local minima, small angle assumption...)
- No use of color or free space

IMPROVABLE!

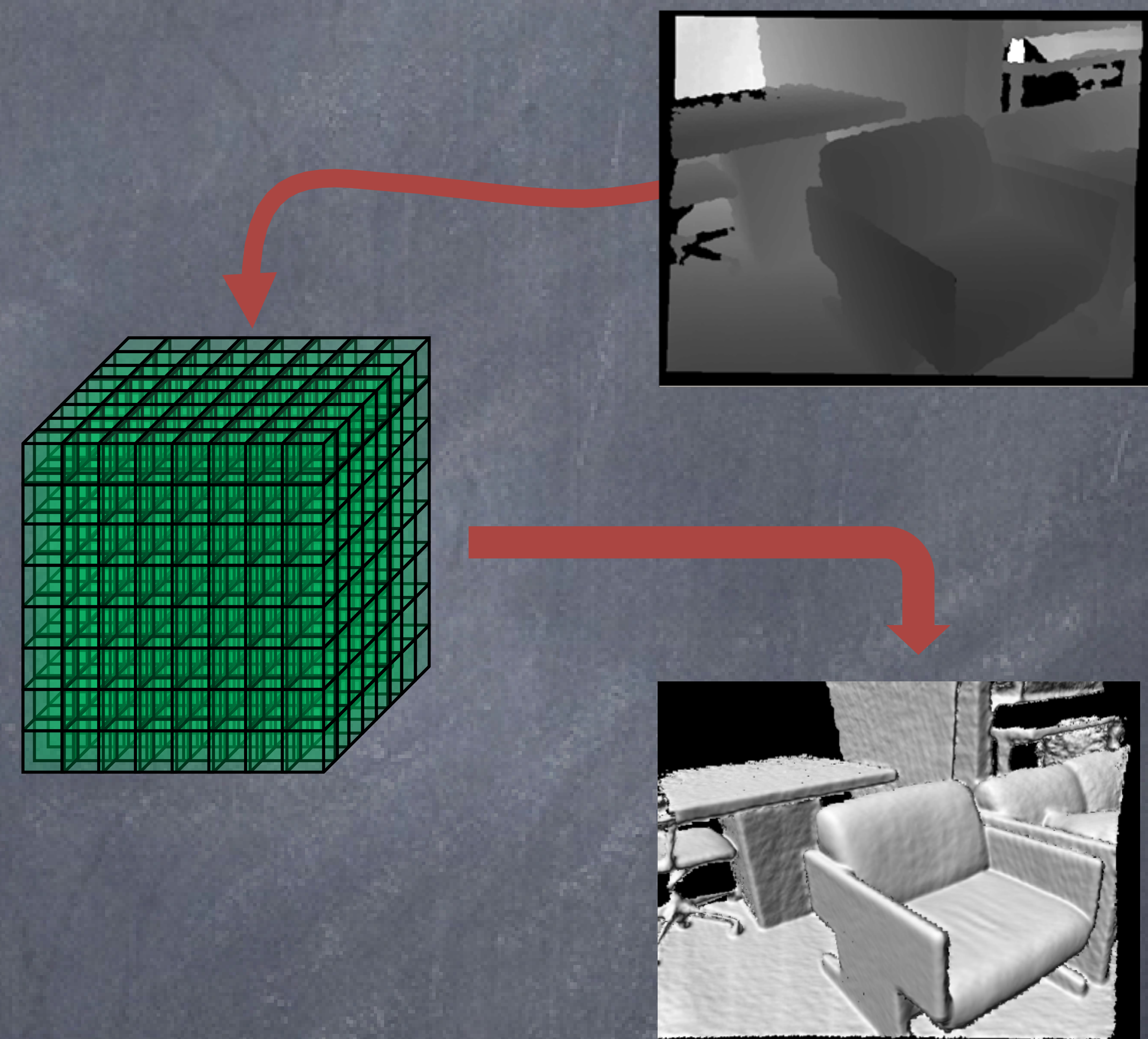
DRIFT

- No way to correct prior mistakes
- Model always assumed correct

VERY UNCLEAR!

Scalability: Intuition

- Kinect Fusion averages over
  all space, all the time
- GPU Memory ~= 1GB
- (1GB) / (sizeof(float d)+sizeof(float w))
  = 512 x 512 x 512 voxels
  = 3m x 3m x 3m (for 6mm voxels)

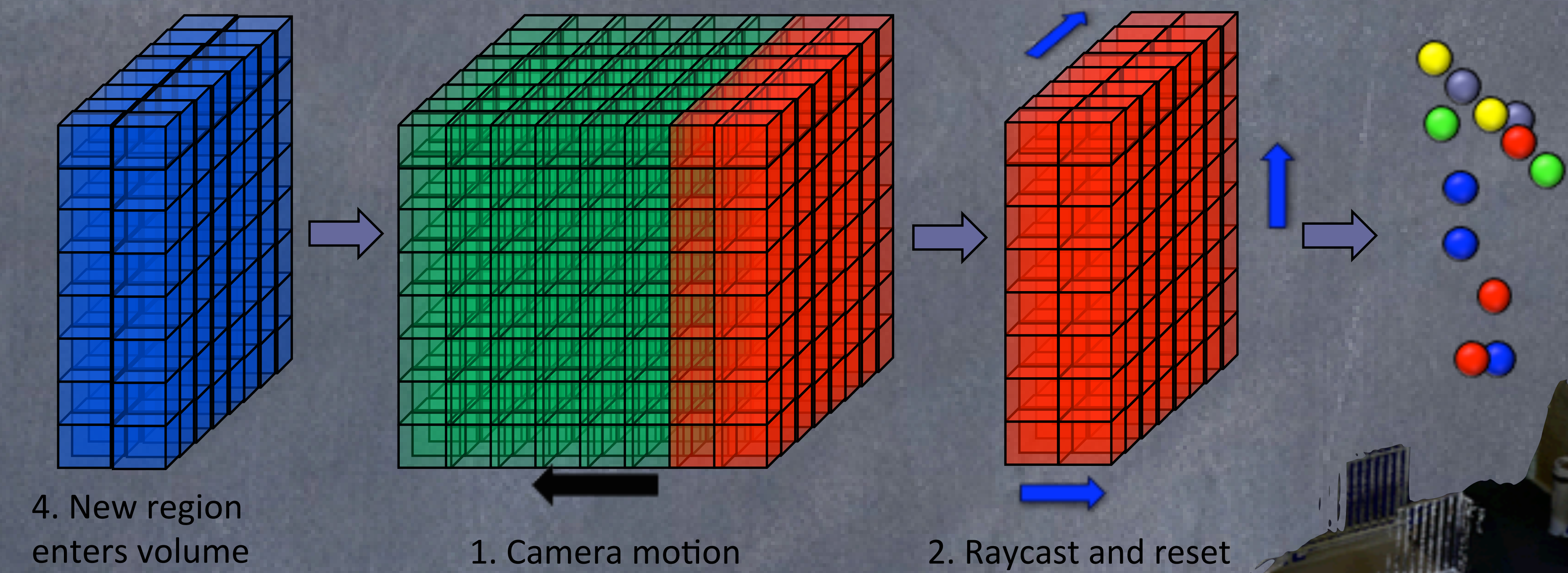


- Space goes with length^3. Moore's law won't solve this any time soon.

Scalability: Kintinuous 1.x

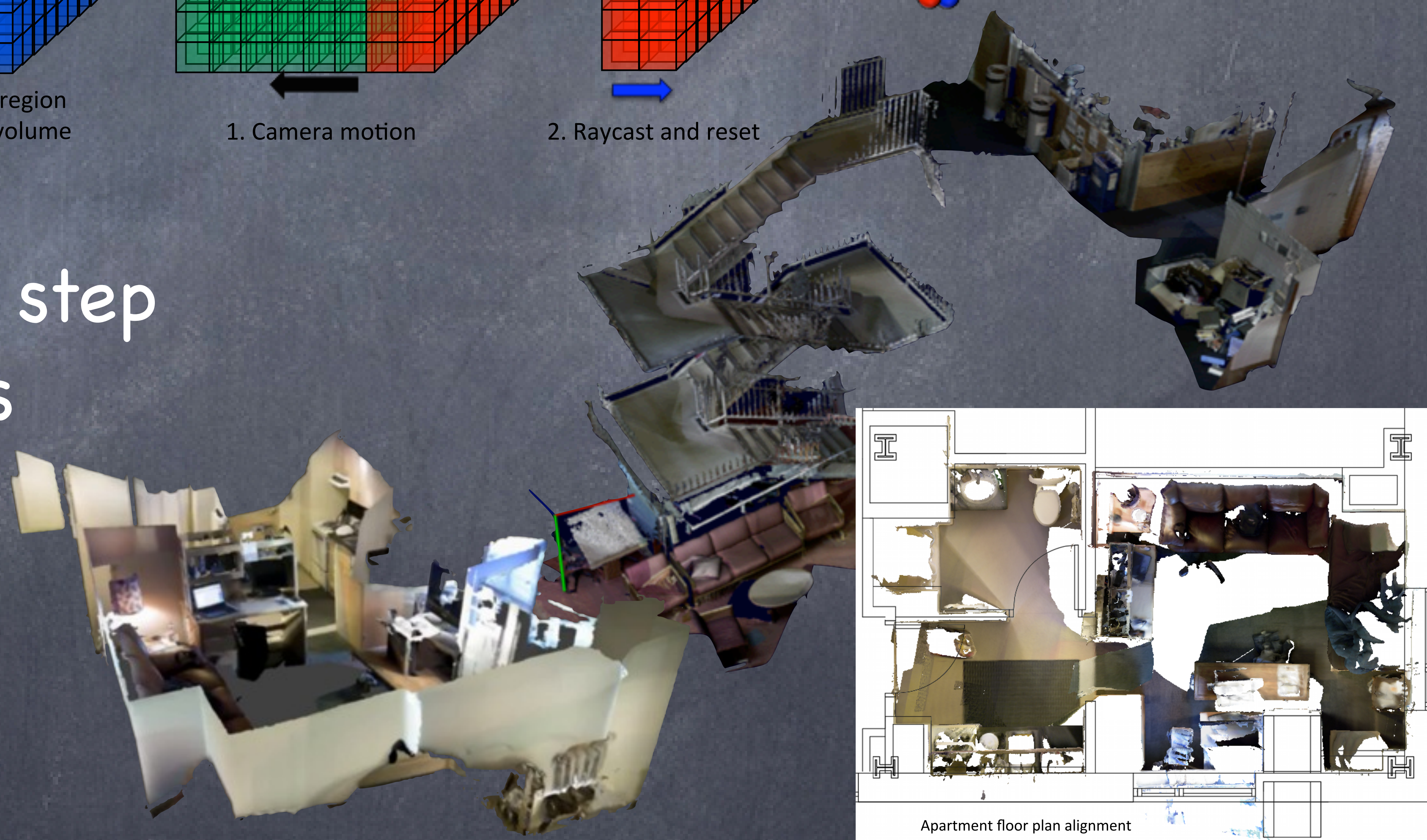
Whelan et al, RSS '12 (workshop) and ICRA '13 (Conference)

- Insight: treat TSDF as cyclical buffer
  - As camera moves, slices pop out and are meshed



- Added color to registration step
  - Helps with <u>precision</u> issues

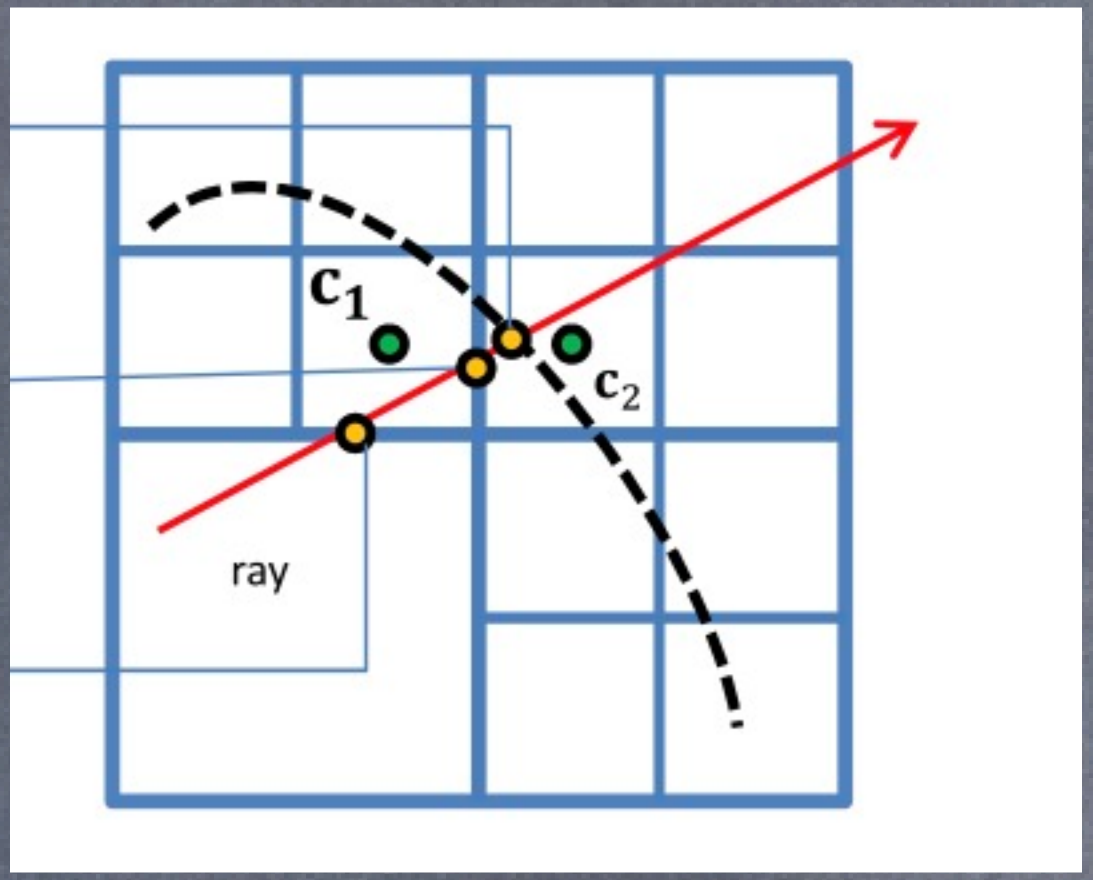
- Similar concepts developed independently in pcl (kinfu_large_scale_app)

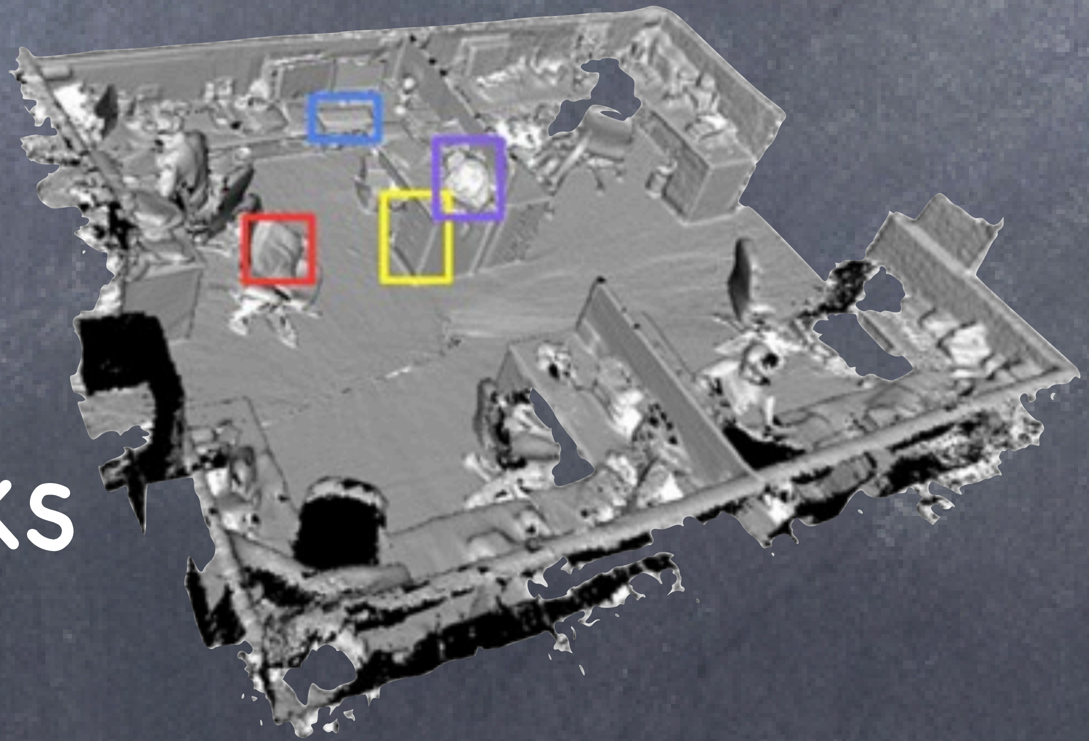


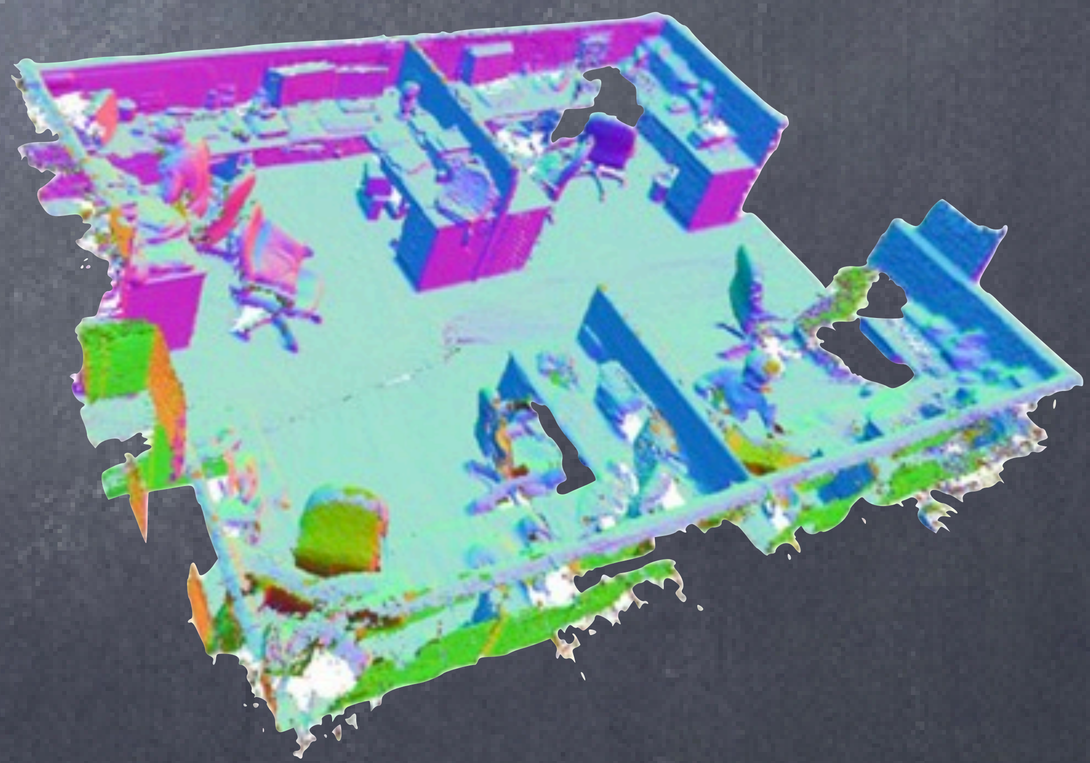
Scalability: Octrees

Zheng et al, CVM '12

- Dense TSDF is wasteful
- Insight: Empty space should have much bigger bins
- Ported KinFu to use an octree
- My independent (unpublished) version:
  - Hierarchical visibility checks
  - ~1/5th realtime on CPU
  - http://github.com/sdmiller/cpu_tsdf.git



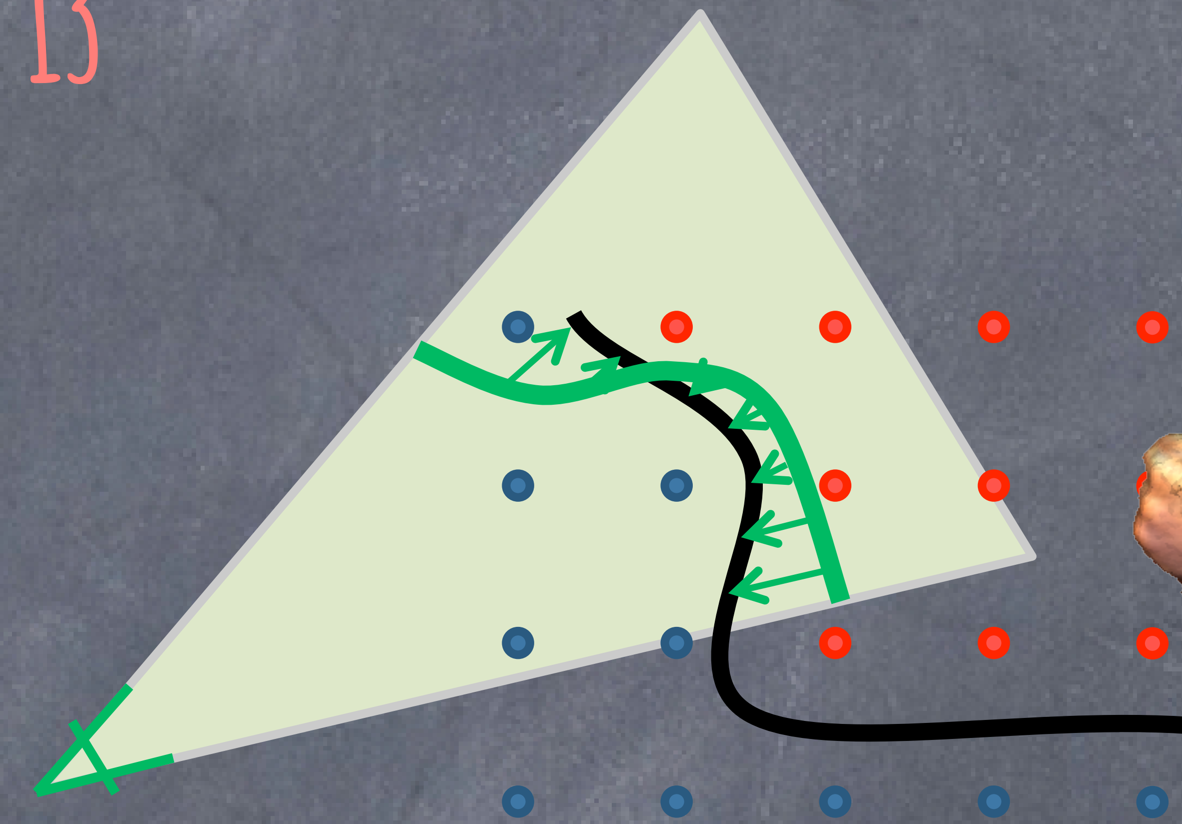




Precision: Volumetric Registration

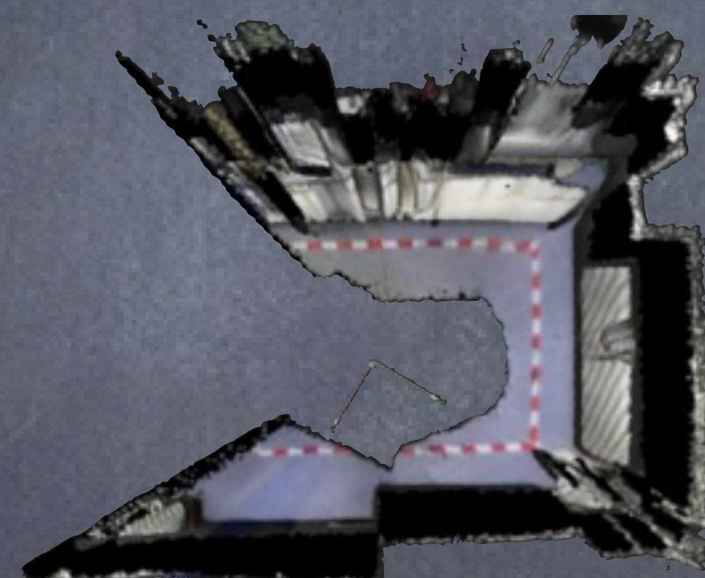
Bylow et al, RSS '13

- TSDF stores how far everything is to surface; why bother with synthetic clouds or ICP?



- Insight: minimize Signed Distance ($f(x)$) directly!

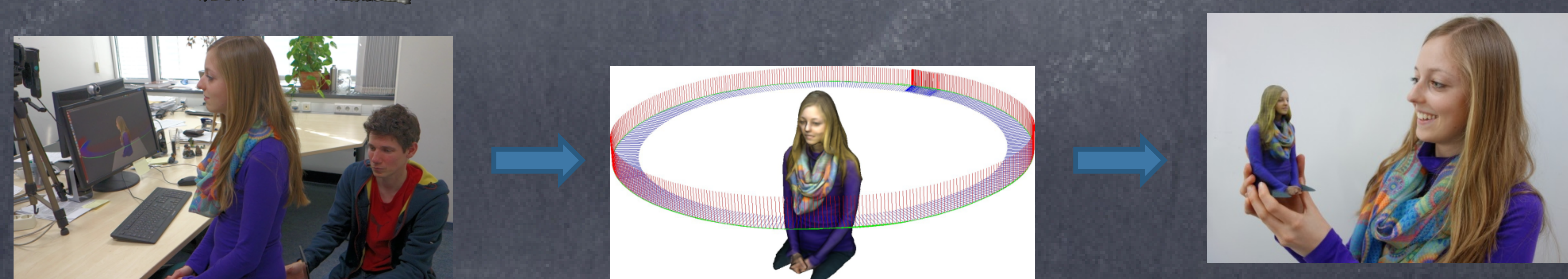
$$T = \arg\min_T \sum_p f(Tp)^2$$



Algorithm	Resolution	Teddy (RMSE)	Desk (RMSE)	Plant (RM
KinFu	256	0.156 m	0.057m	0.598 m
KinFu	512	0.337 m	0.068 m	0.281 m
Our	256	0.086 m	0.038 m	0.047 m
Our	512	**0.080 m**	**0.035 m**	**0.043 m**

More accurate, just as fast

- Newton's method on analytic gradient + Hessian

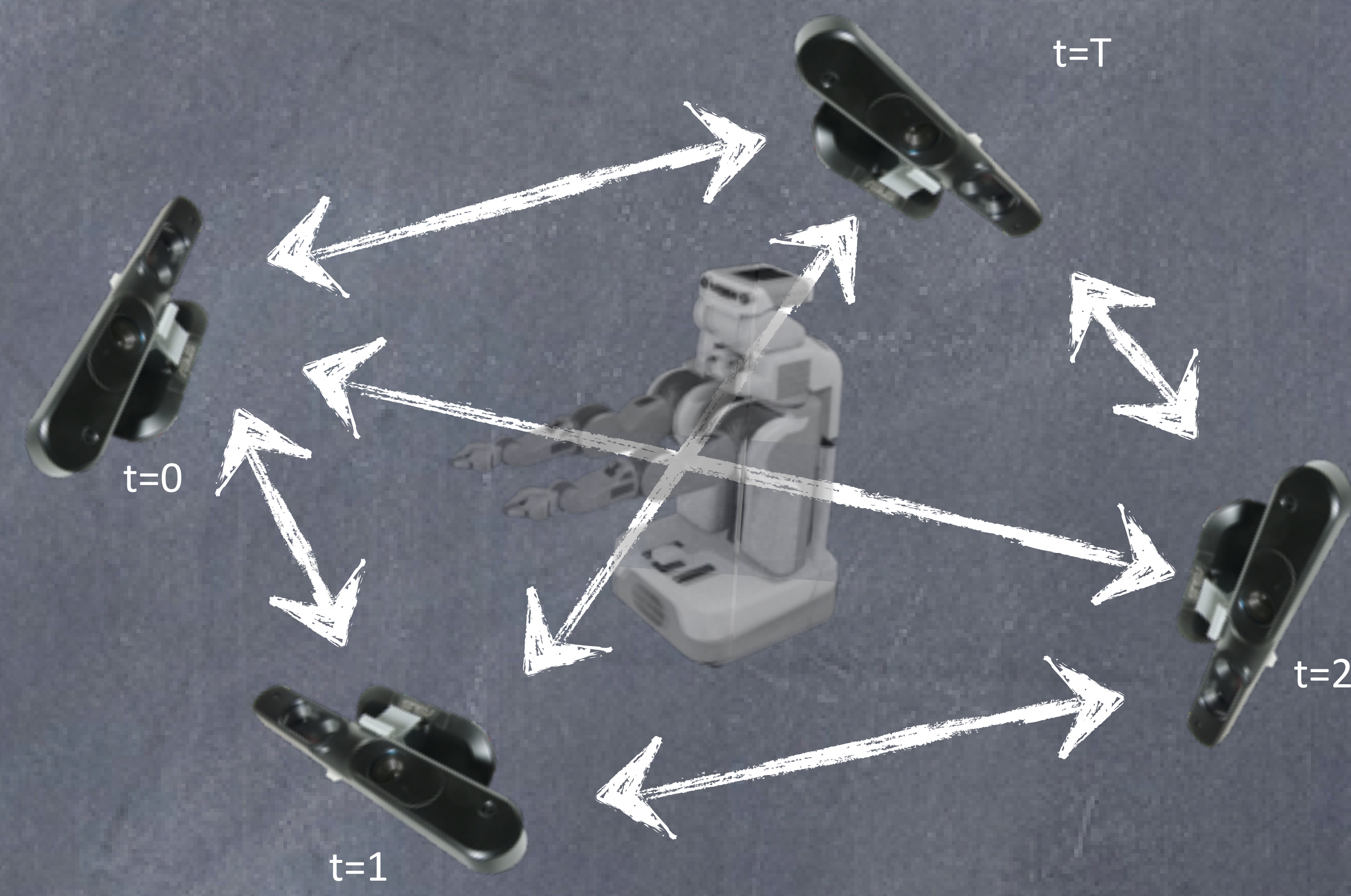


Good enough for watertight models

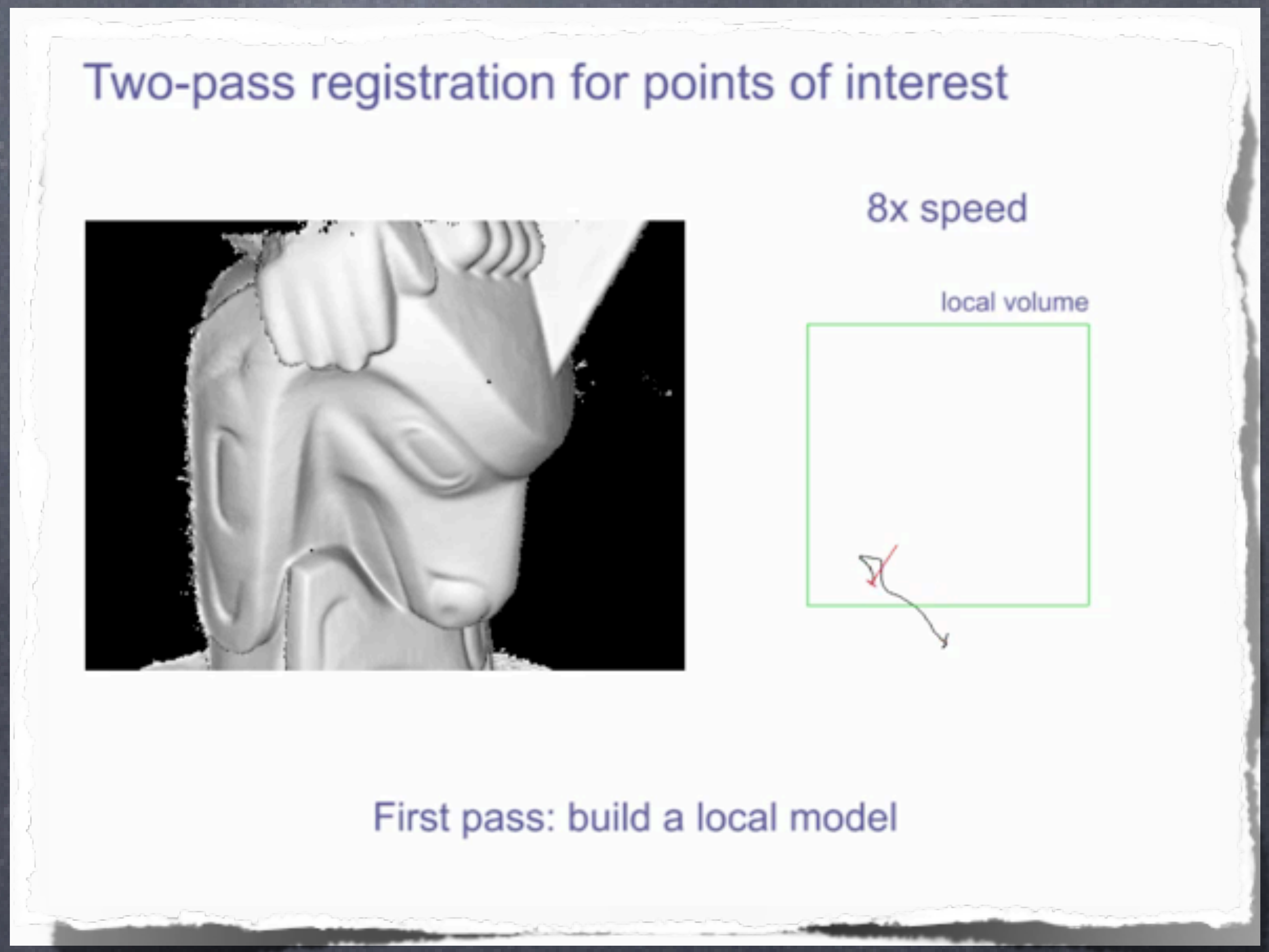
Drift: Adding SLAM (Offline)

Zhou et al, Siggraph '13

- Idea: Combine SLAM and KinFu
  - -Run KinFu over and over
  - -SLAM to align meshes + add loop closure
  - -Re-integrate with the new poses



- "Points of interest": give more weight to parts of the scene we've seen more



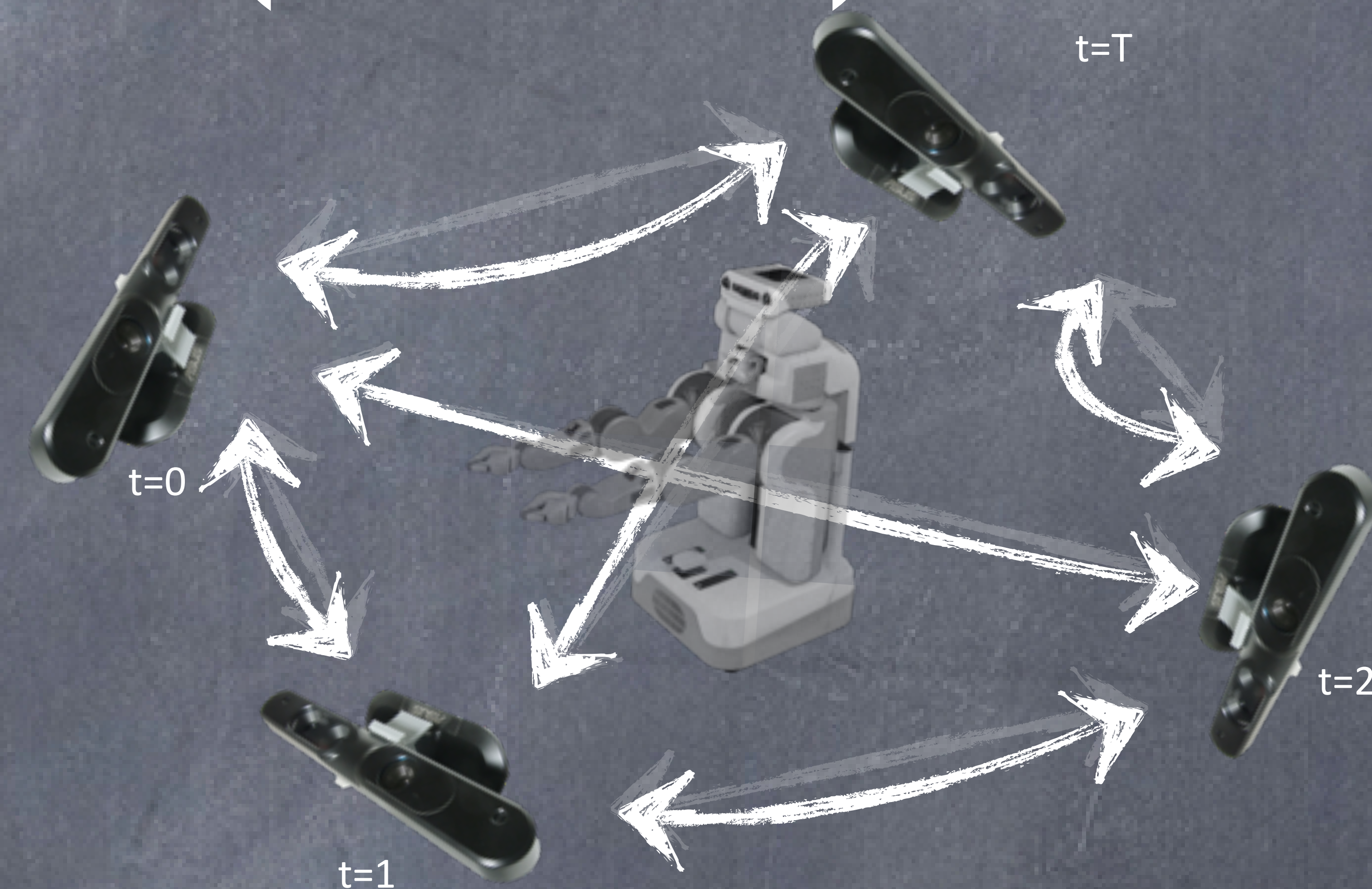
Drift: Elastic Fragments (Offline)

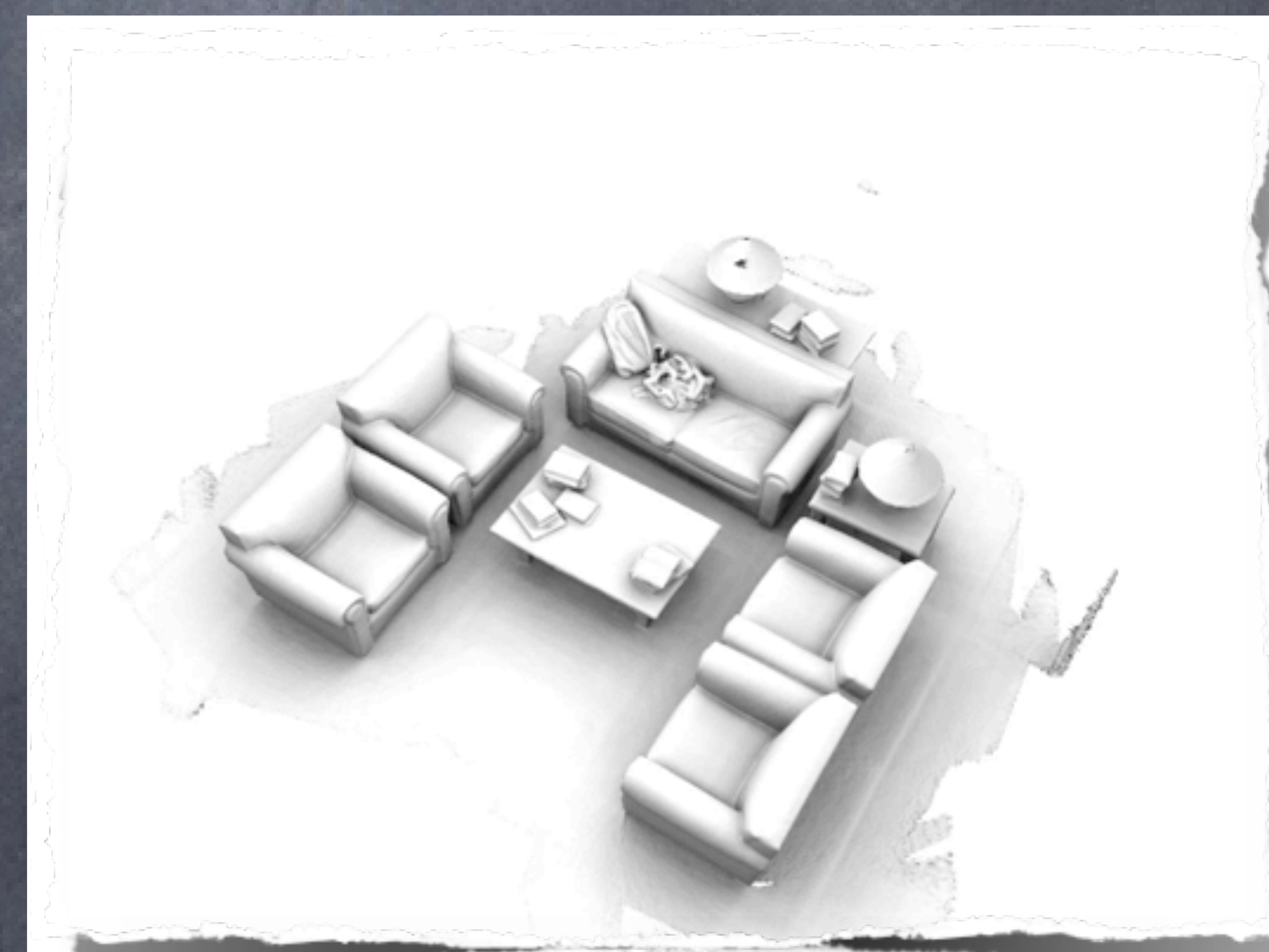
Zhou et al, ICCV '13

- Idea: Take that last idea, but let's bend the submeshes into shape

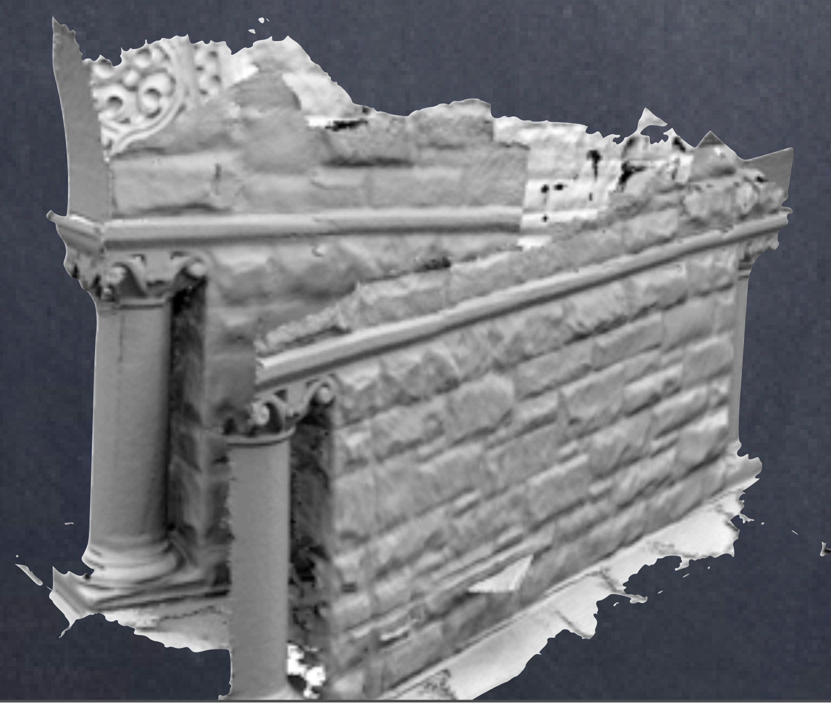
- Motivation: sensor distortion

- "Rigid Registration" -> "As Rigid As Possible" (Igarashi et. al, TOG 2005)





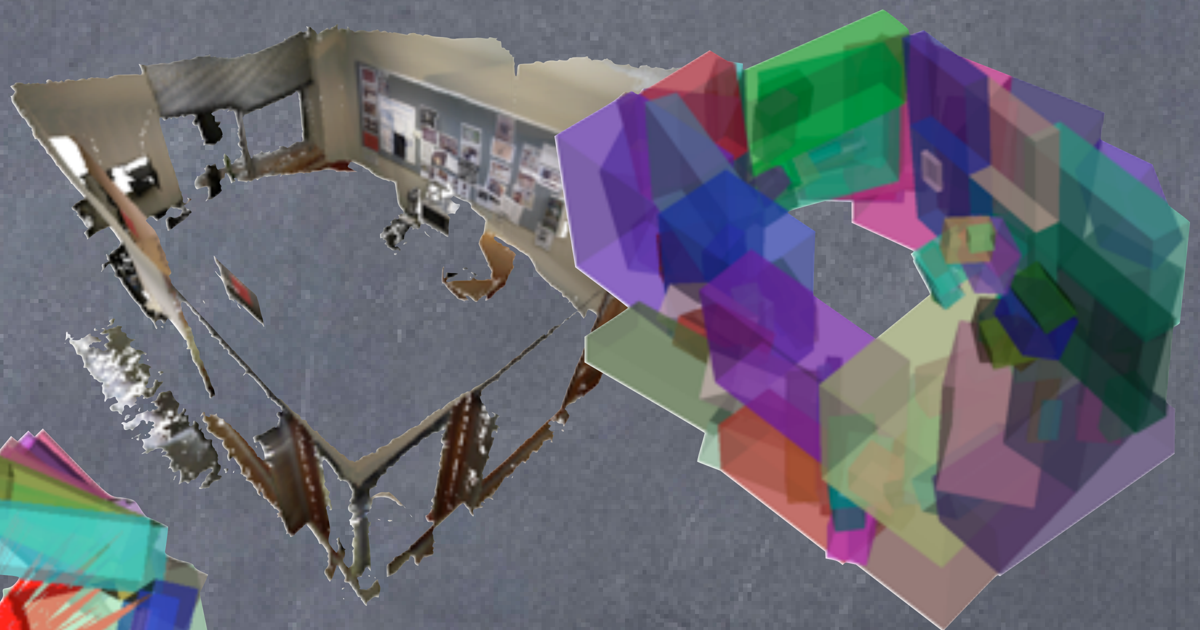
Kinect Fusion

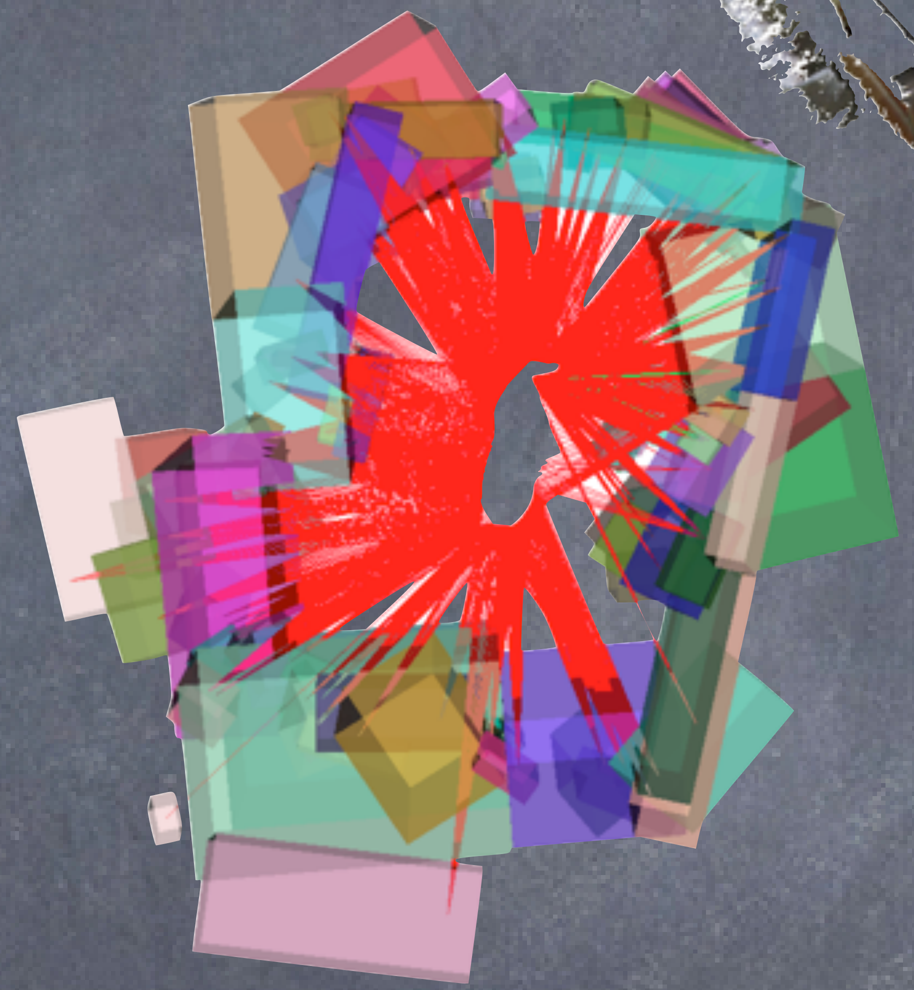


ours

Drift: Patch Volumes

Henry et al, 3DV '13





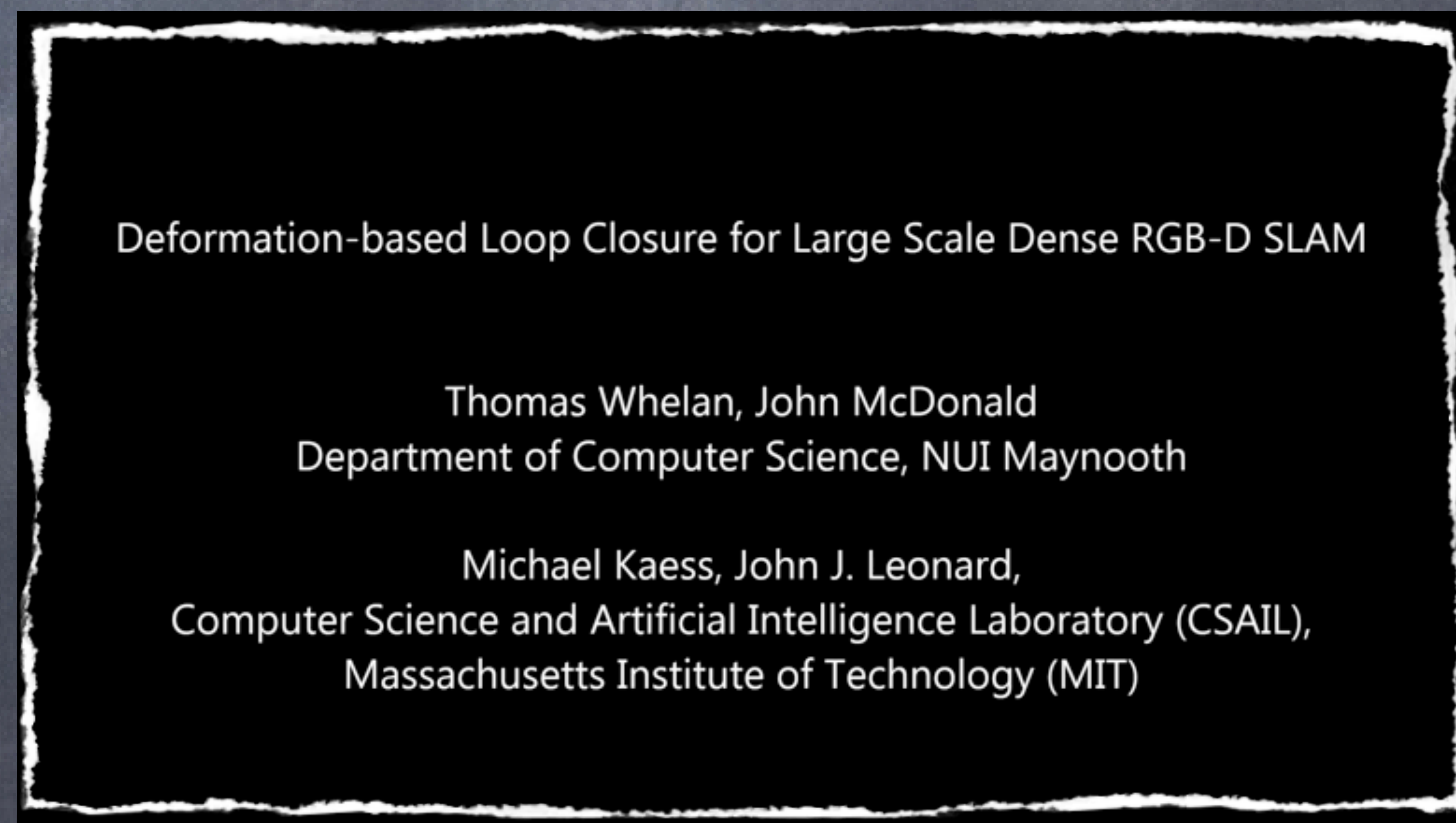
- Idea: Run KinFu on small, adaptively-sized regions (“patch volumes”)
- Run a SLAM solver to align volumes
- Bonuses:
  -more principled choice of w_t

Drift: Kintinuous 2.0

Whelan et al, IROS '13

- Idea: Run KinFu and SLAM in parallel, use SLAM solution to deform the mesh
- Similar to "Elastic Fragments" in spirit, though more about large-scale adjustments
- Also runs in realtime!

Mesh deformation



What's missing?

Real SLAM integration

- Everything runs SLAM and KinectFusion, staples together the output
- None update the TSDF to account for closure -- how do we do it?

Graceful failure handling

-"frame-to-model" means we can't recover from bad models. How do we detect failure?

Smarter registration

-We output a mesh or cloud. Why not use the TSDF? TSDF localization, TSDF object detection, TSDF segmentation?

Thanks!