



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

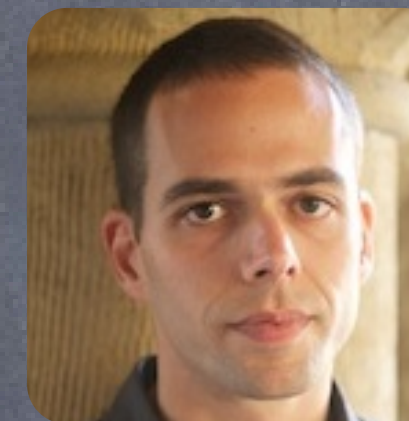
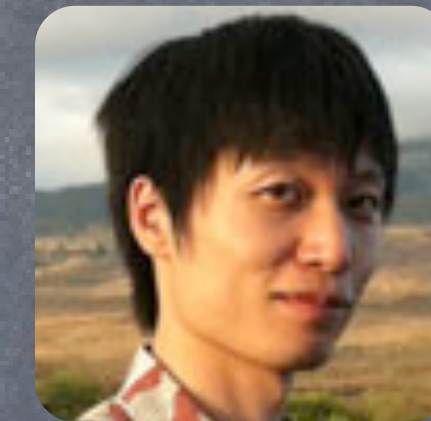
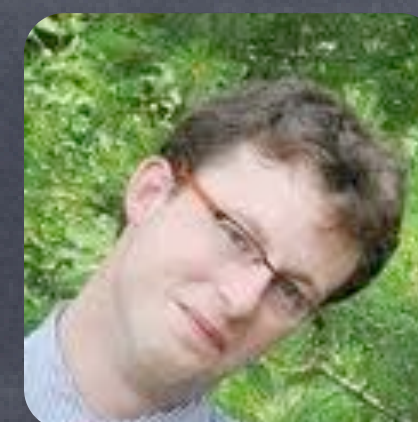
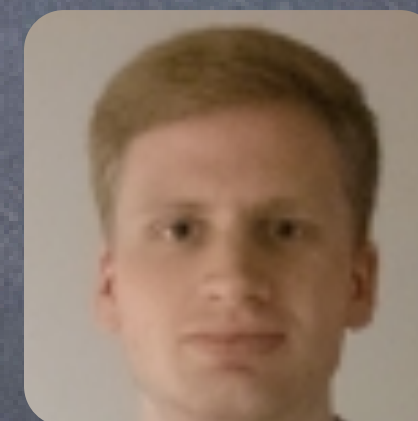
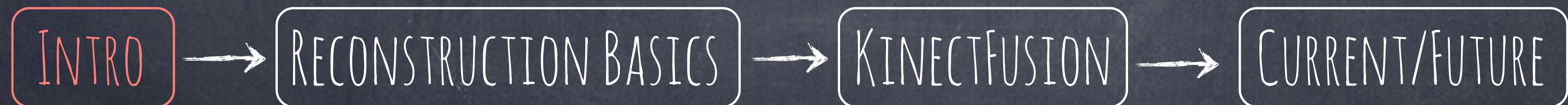


FIGURE-DONORS

Erik Bylow
Juergen Sturm
Thom Whelan
Peter Henry
Radu Rusu



Intro

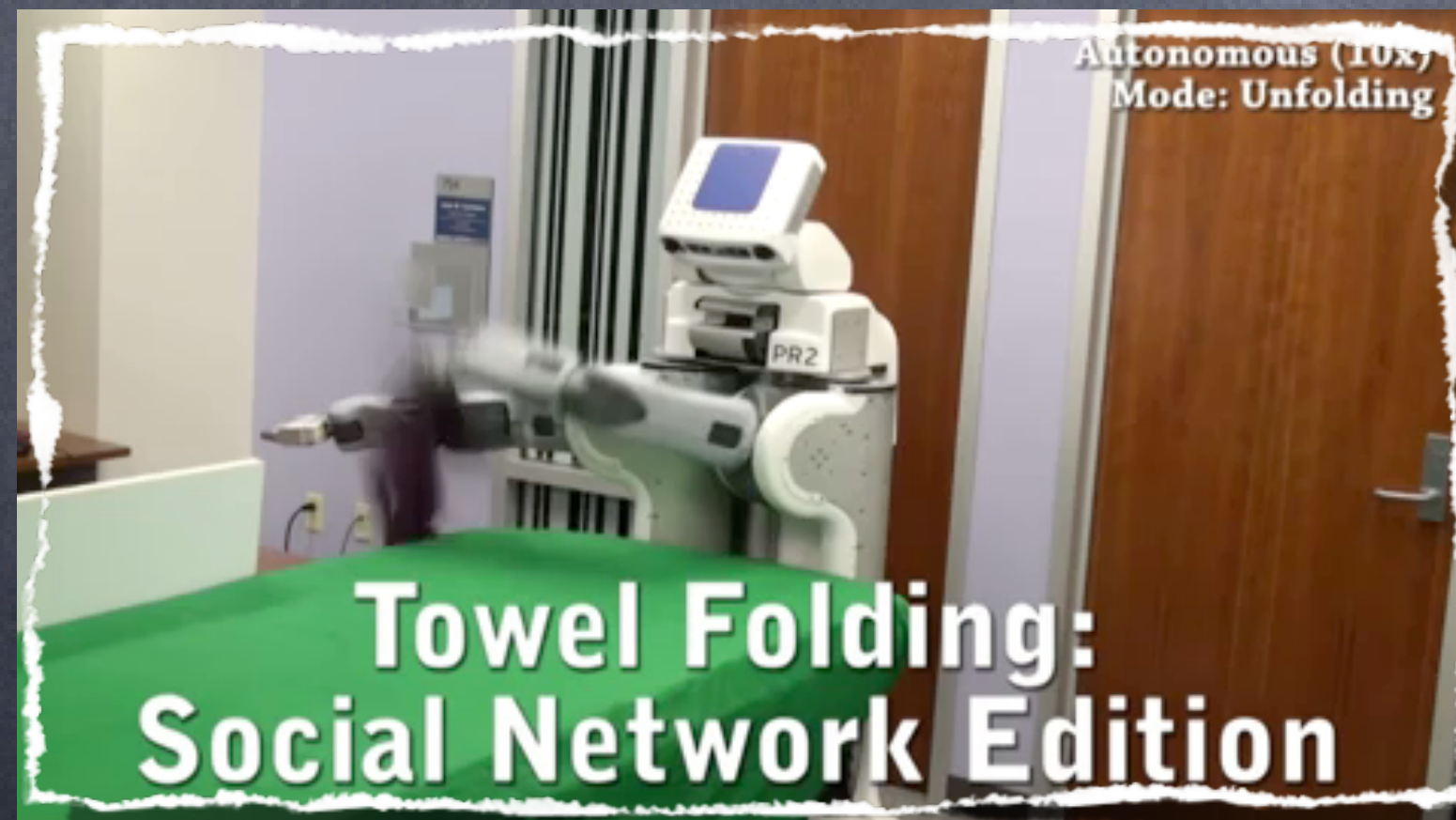
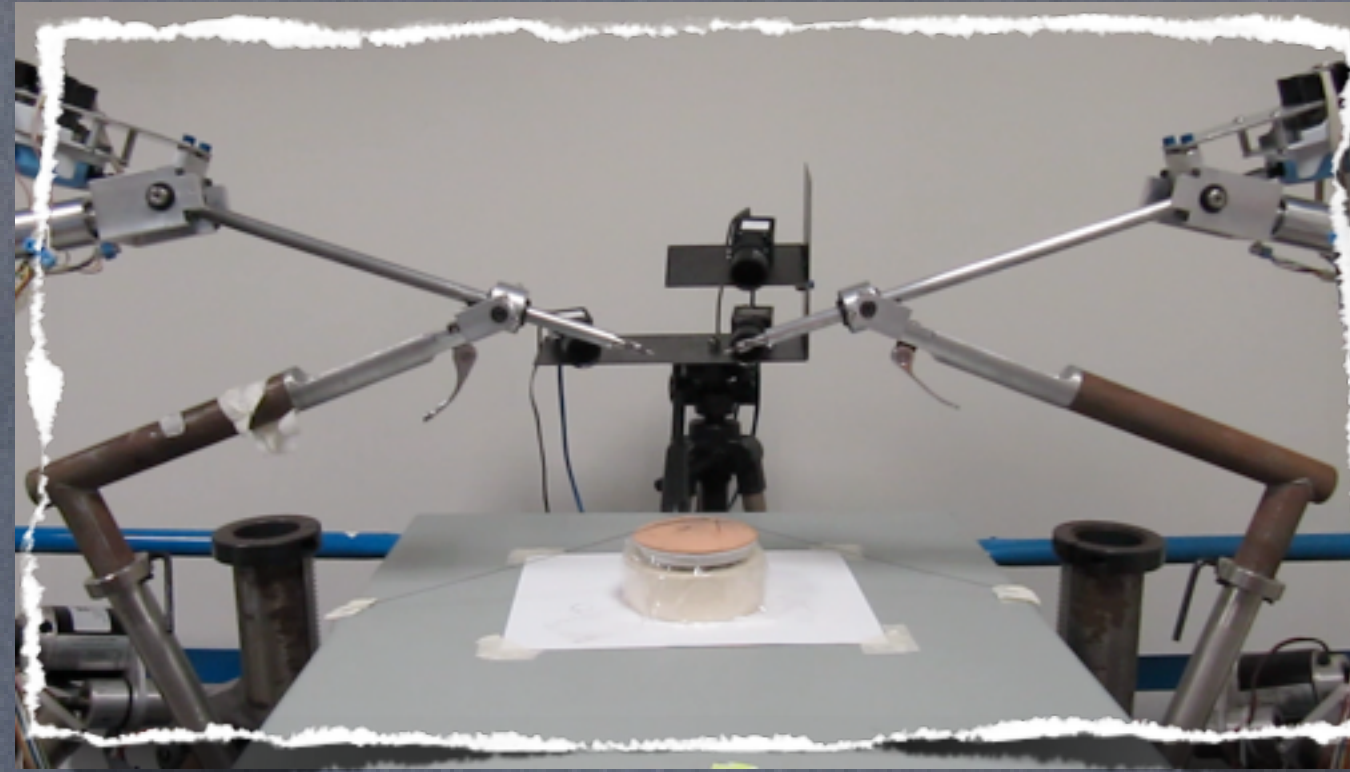


Who am I

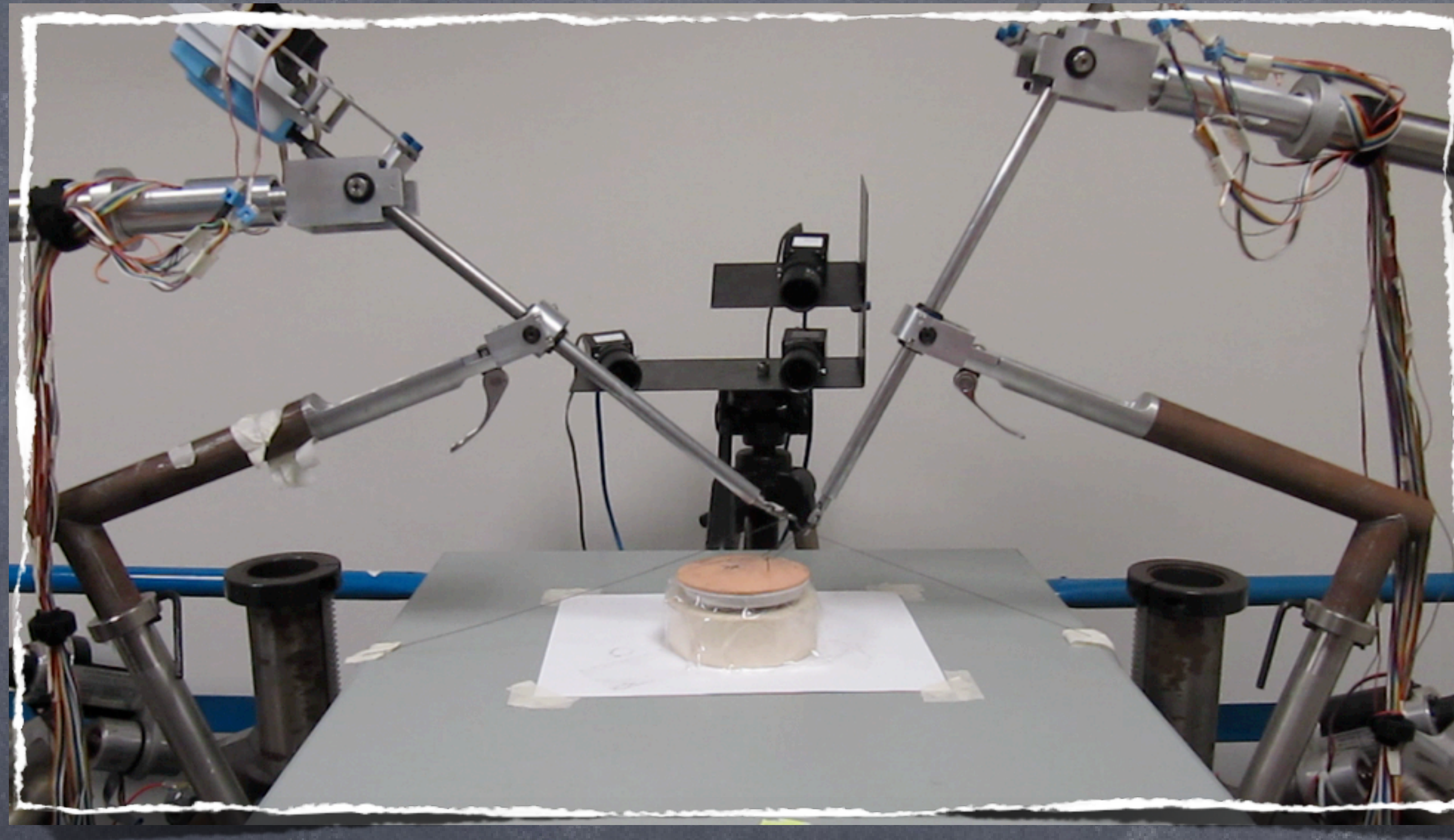
Undergrad: Cal

- (2009)
Surgical Robotics

- (2010-2011)
Personal Robotics



Changing priorities

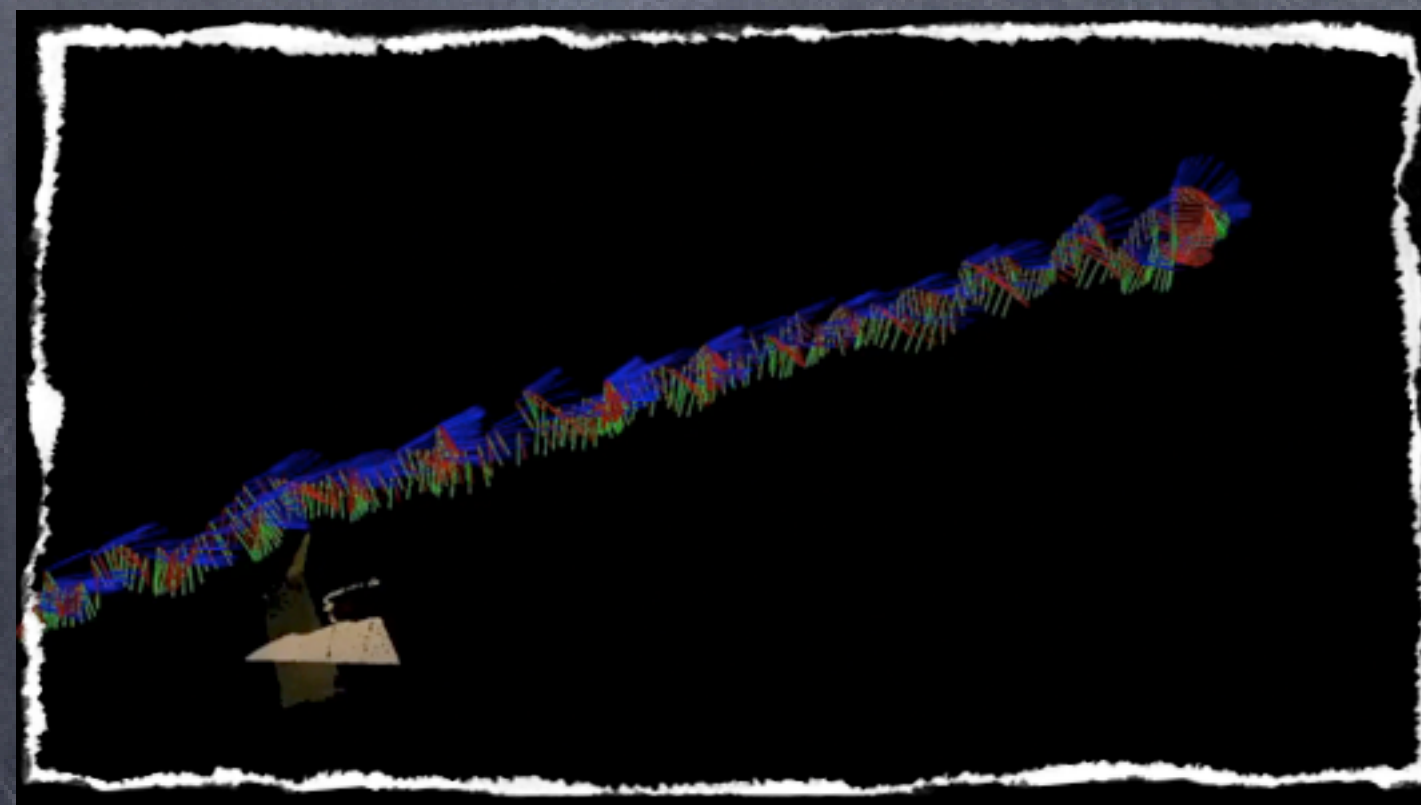
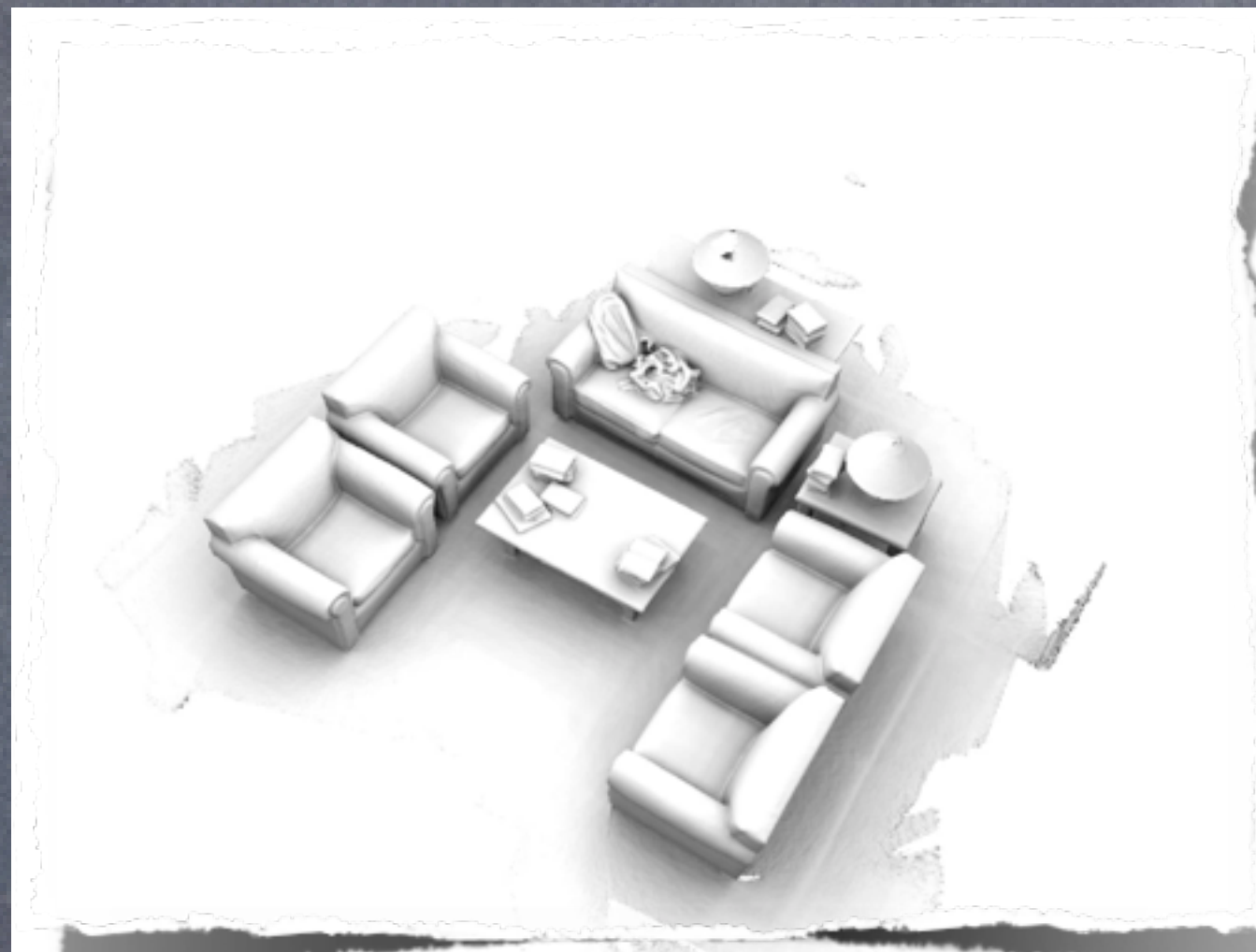


Blind robots are frustrating to work with

Who am I

Grad: Stanford

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



SENSOR 0

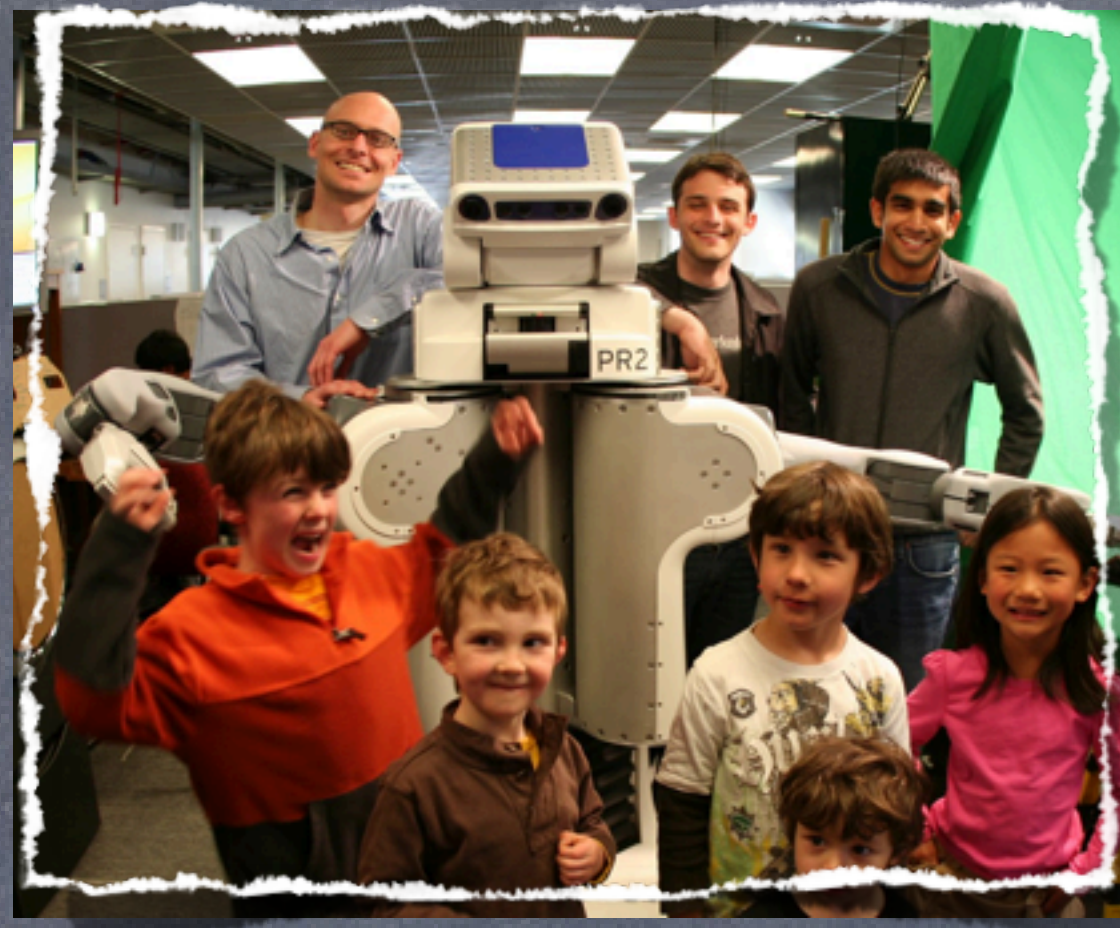


SENSOR 1

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



ANCHORAGE, '10



BERKELEY, '10



SINGAPORE, '10



CBS SMART PLANET, '11

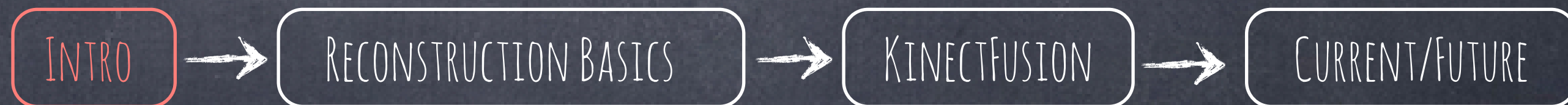


SHANGHAI, '11



TOKYO, '13

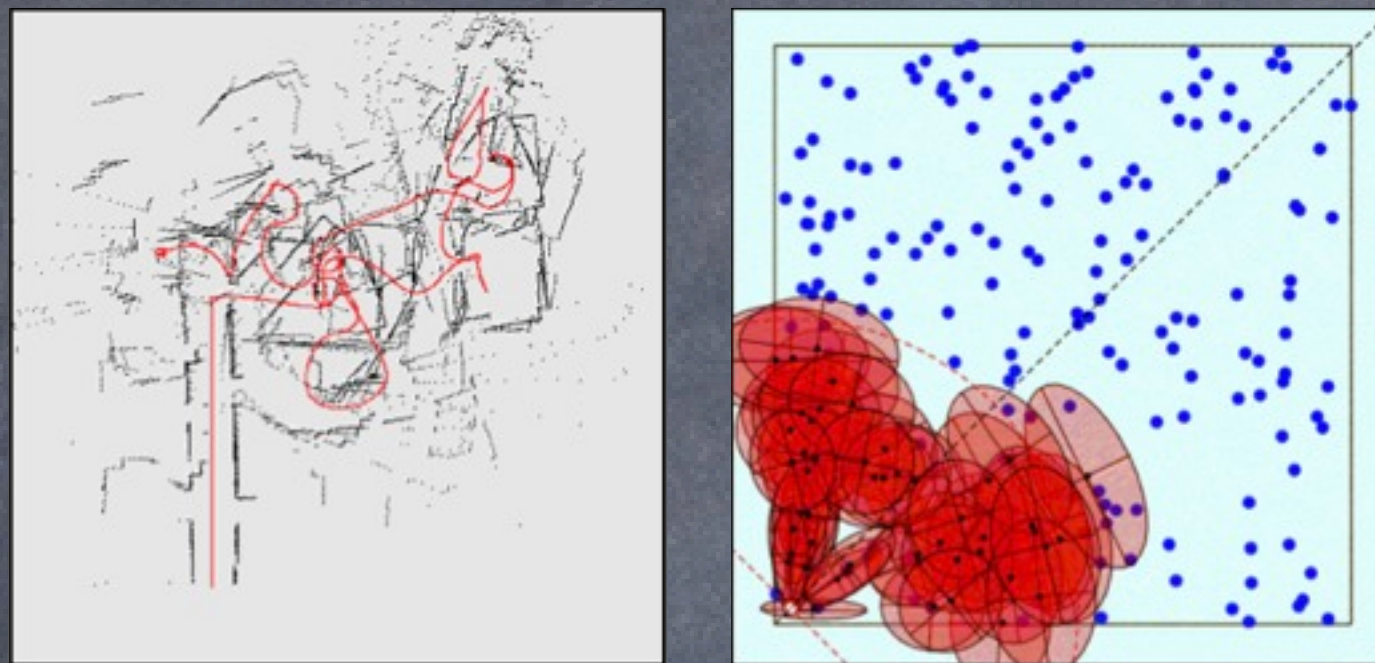
Okay,
back to work



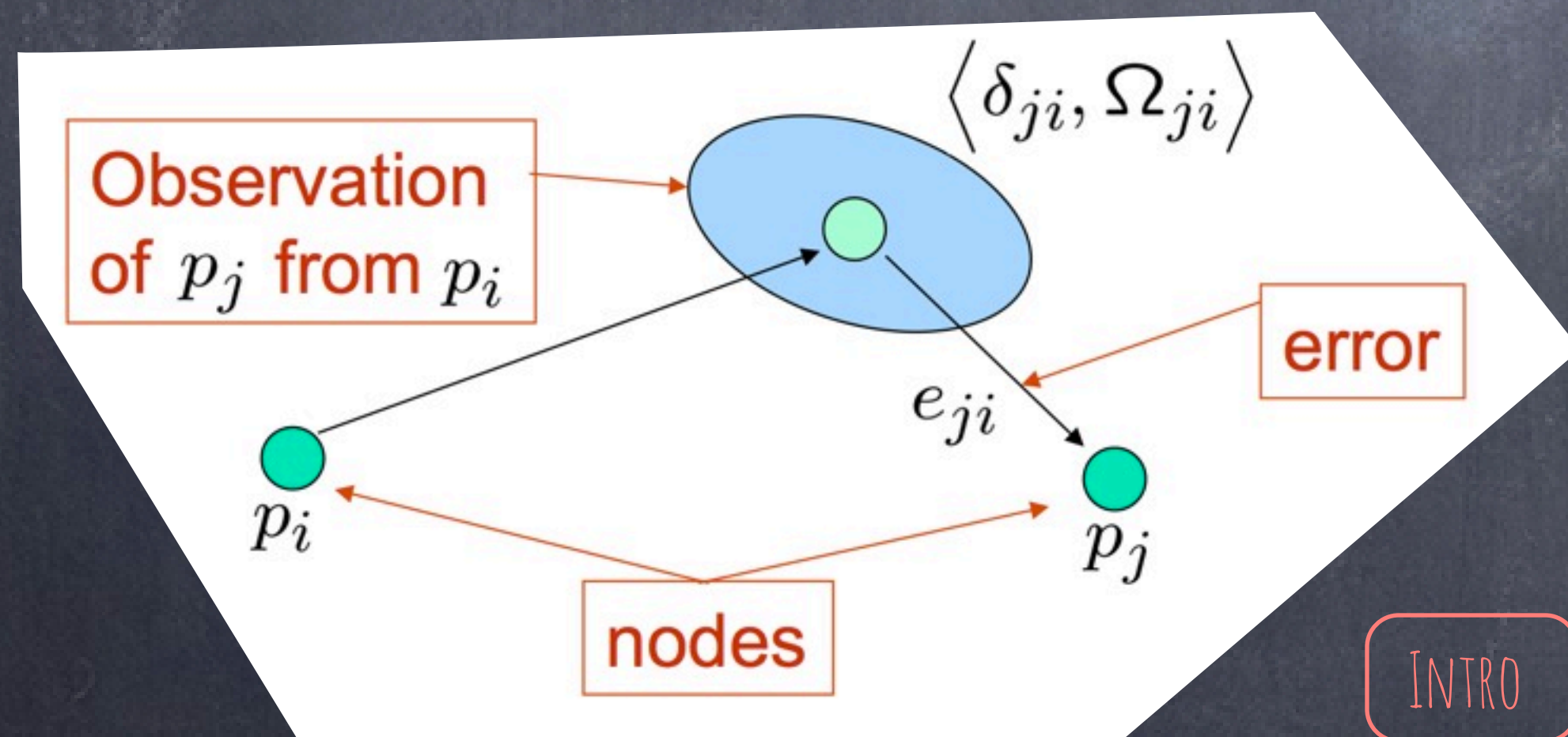
SLAM, as you've learned so far

Track sparse features

“Map” in postprocessing



Poses related via Bayes law



Elegant math integrates all observations

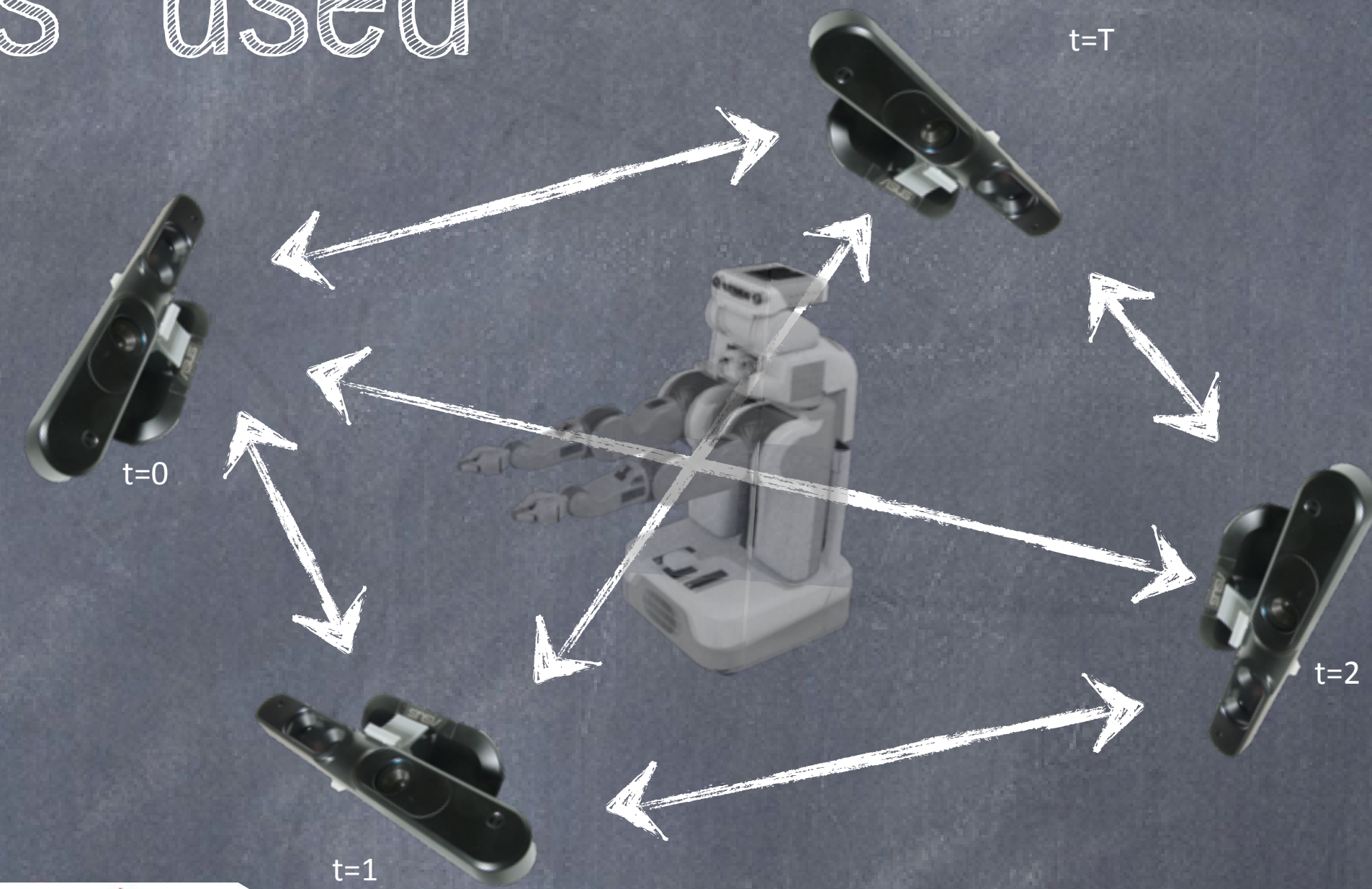
$$w_t^i = \frac{p(z_t | x_{1:t}^i, z_{1:t-1}) p(x_t^i | x_{1:t-1}^i, u_t, z_{1:t-1})}{\pi(x_t^i | x_{t-1}^i, u_t, z_t)}$$



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



• We still need the theory

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

• 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
```

INTRO

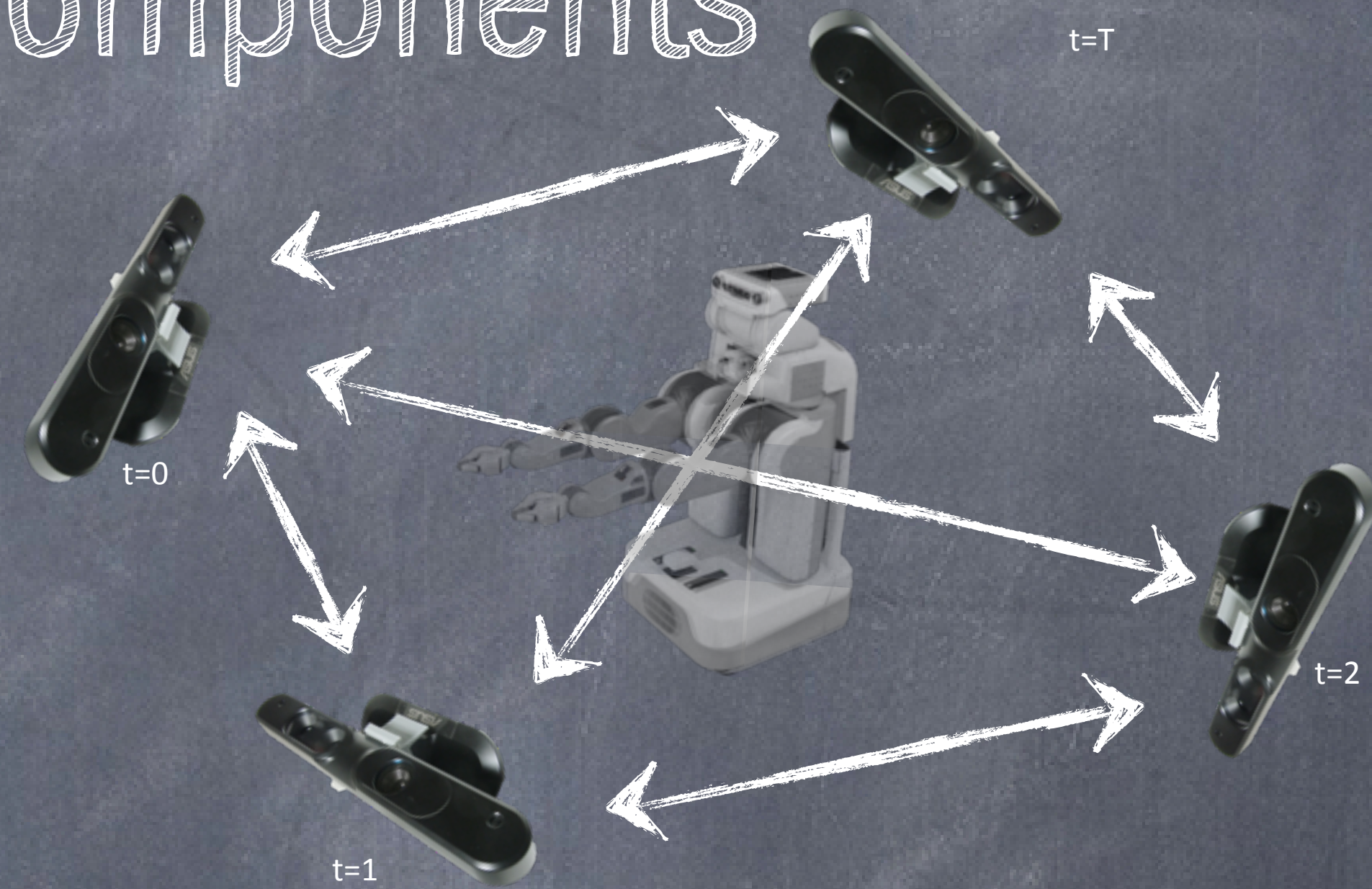
RECONSTRUCTION BASICS

KINECTFUSION

CURRENT/FUTURE

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?



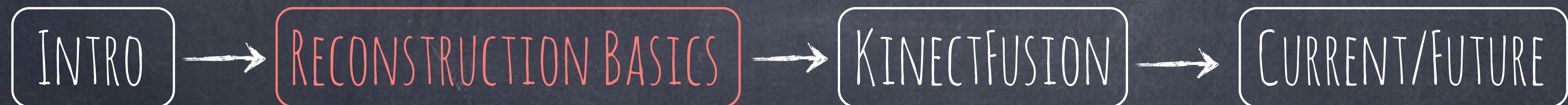
INTRO

RECONSTRUCTION BASICS

KINECTFUSION

CURRENT/FUTURE

Reconstruction Basics



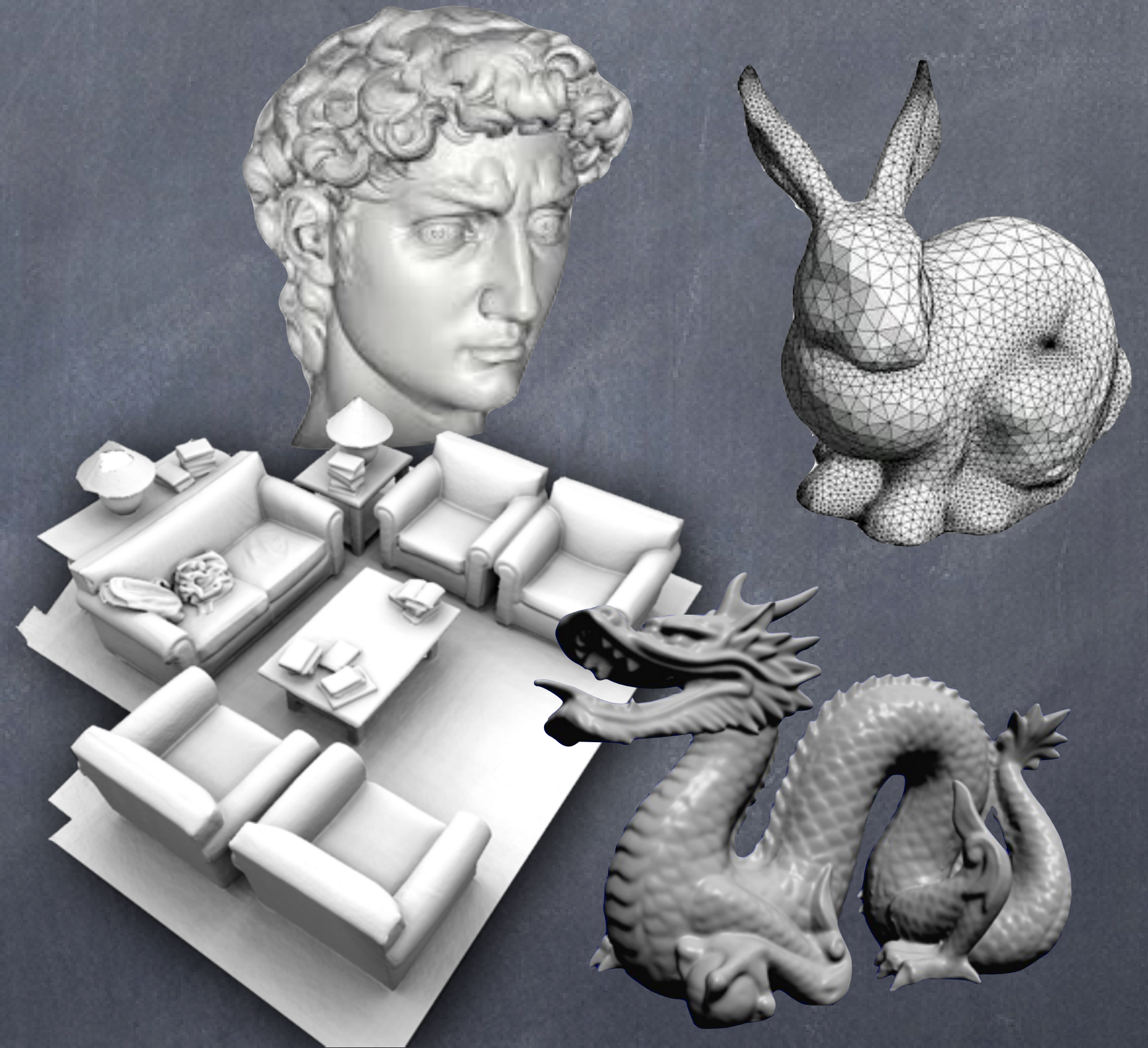
Robotics

SLAM



Graphics

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

INTRO



RECONSTRUCTION BASICS



KINECTFUSION



CURRENT/FUTURE

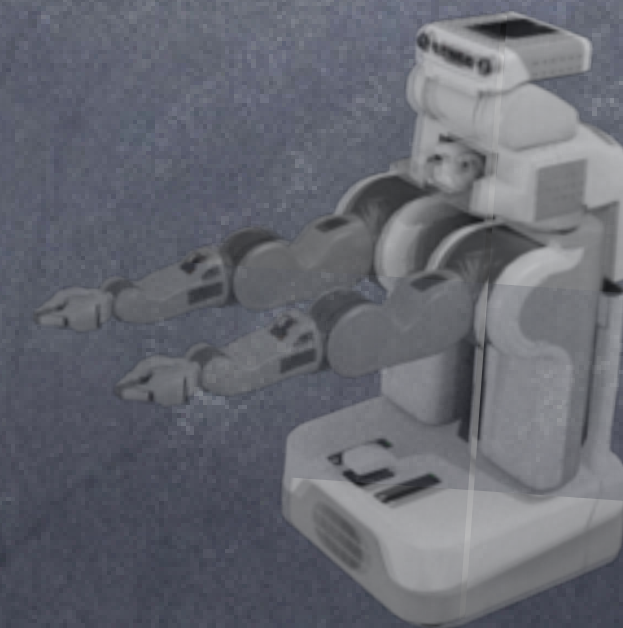
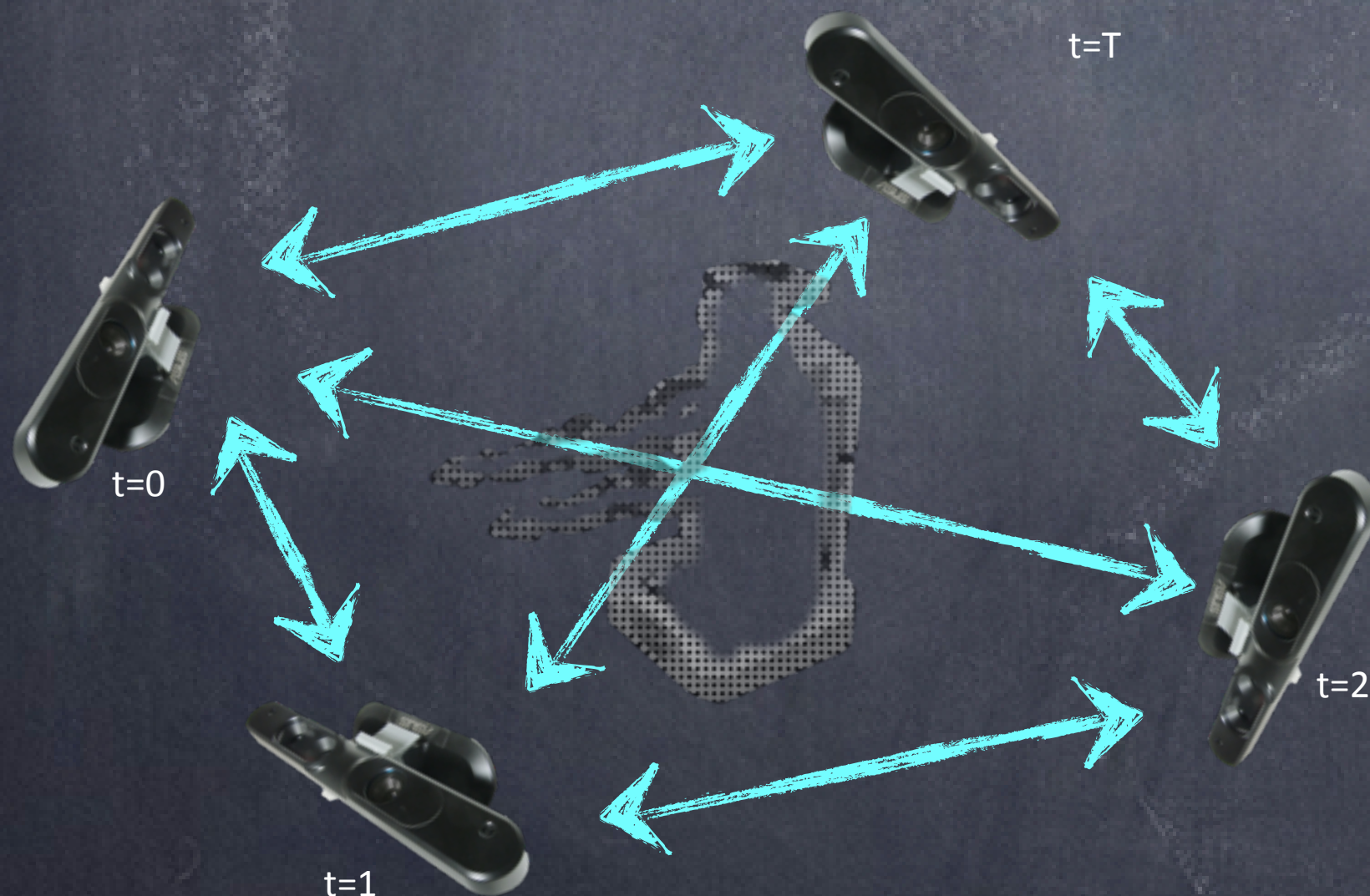
A tale of two modules

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



INTRO

RECONSTRUCTION BASICS

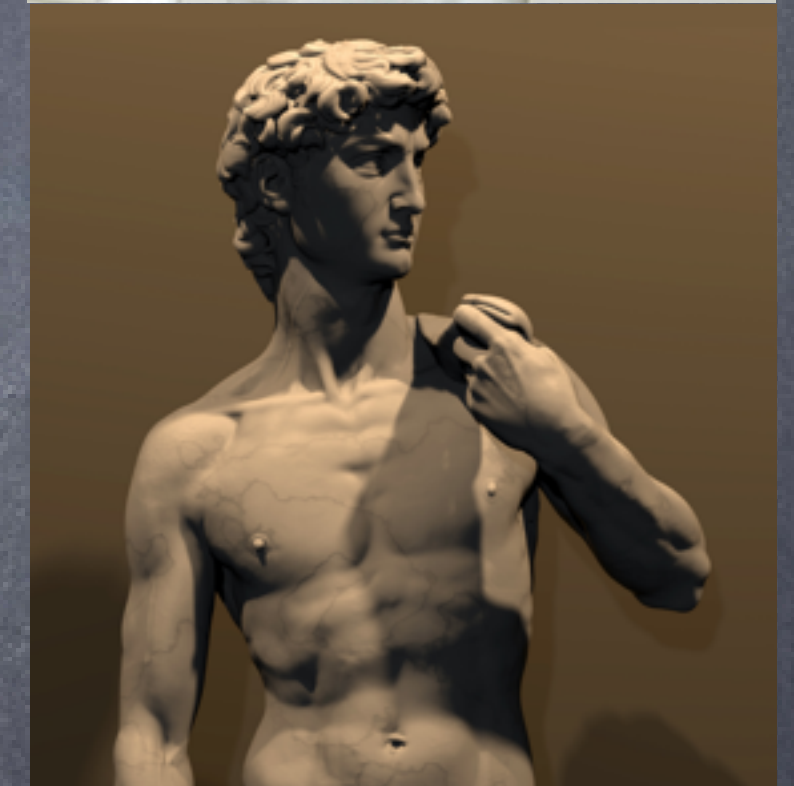
KINECTFUSION

CURRENT/FUTURE

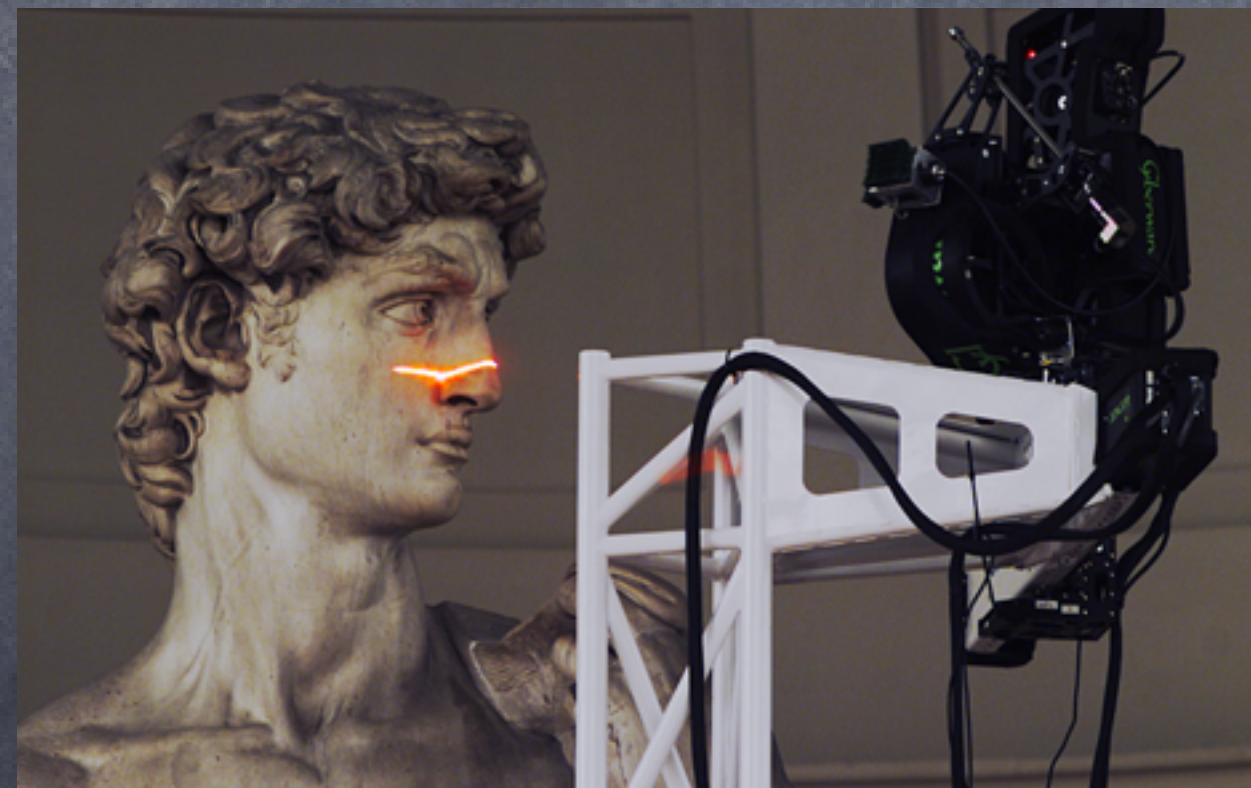
The Michelangelo Project

- 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

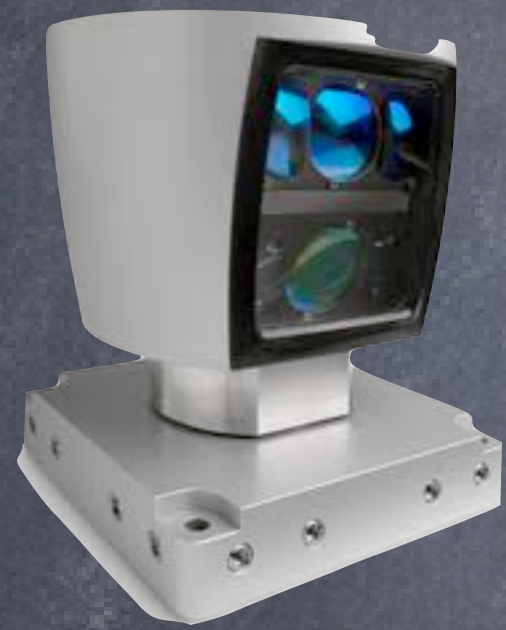


digital, Levoy's website



Why is this hard? Lots of noisy Data

VELODYNE HDL-64E
1.3M POINTS PER SECOND



SLOWLY SCAN EVERY CREVICE
>= 30 MINUTES

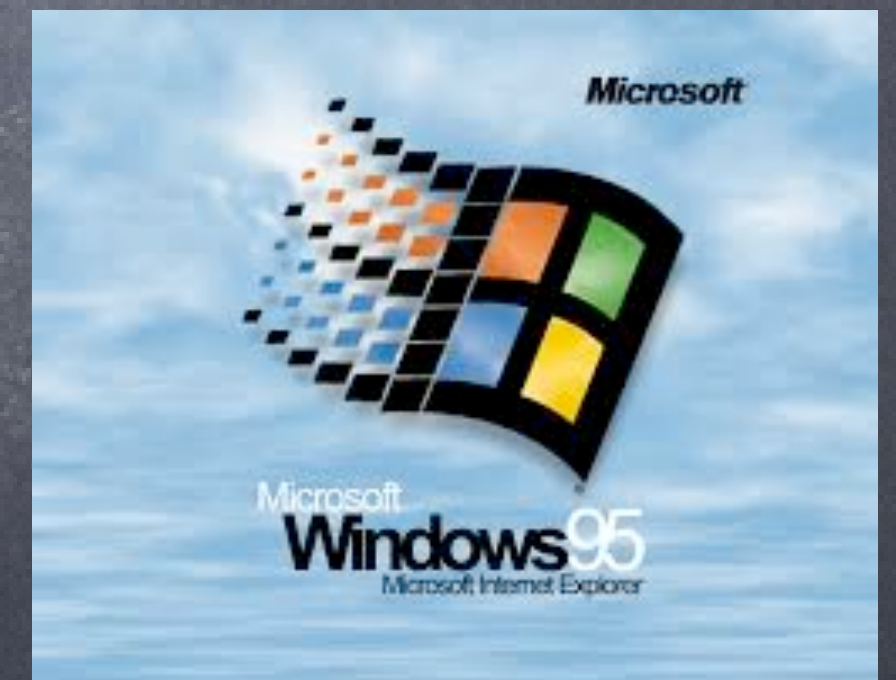
X



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



KINECT
9.2M POINTS PER SECOND



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

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

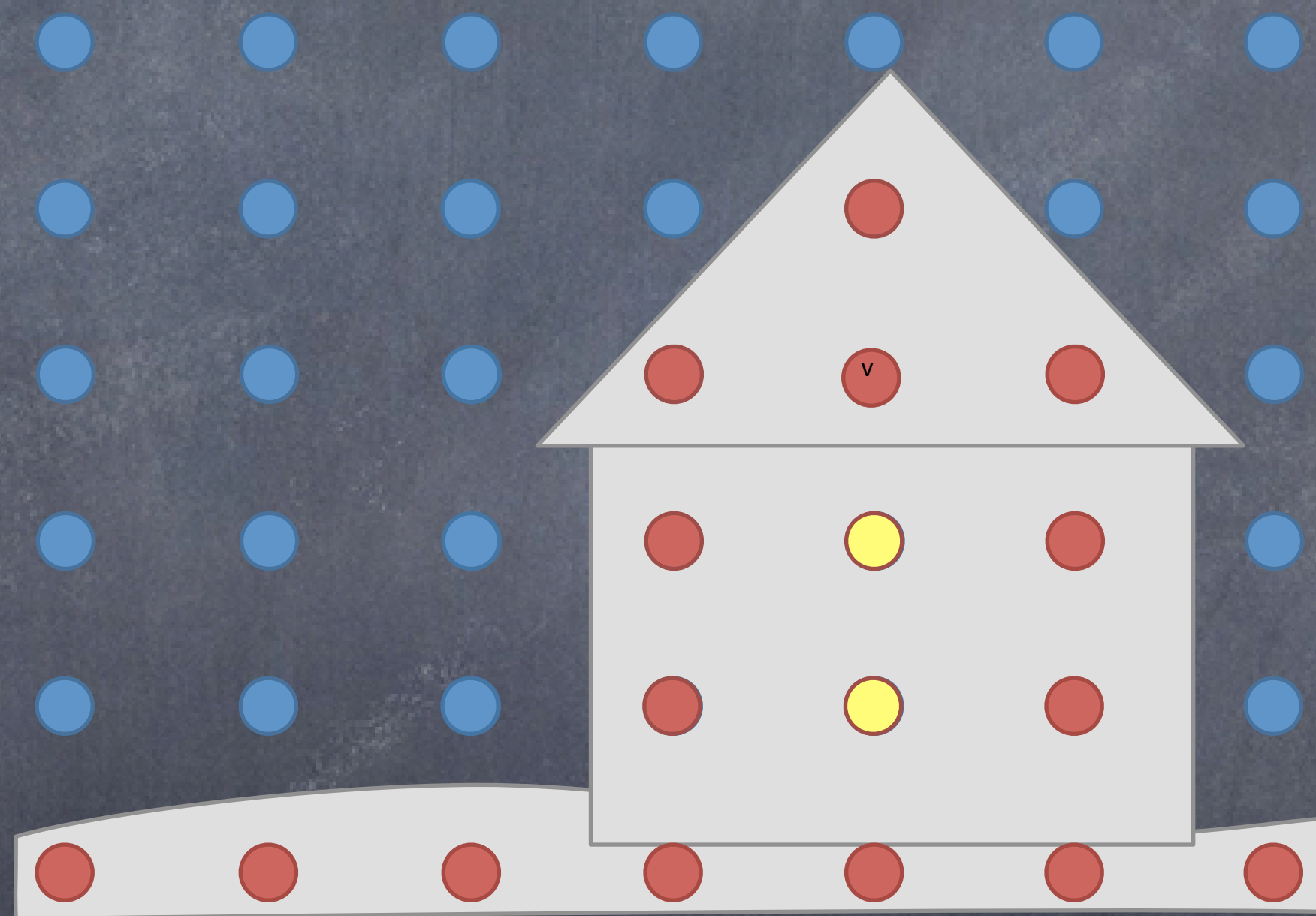
- Must be parametrized to be learned



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

INTRO

RECONSTRUCTION BASICS

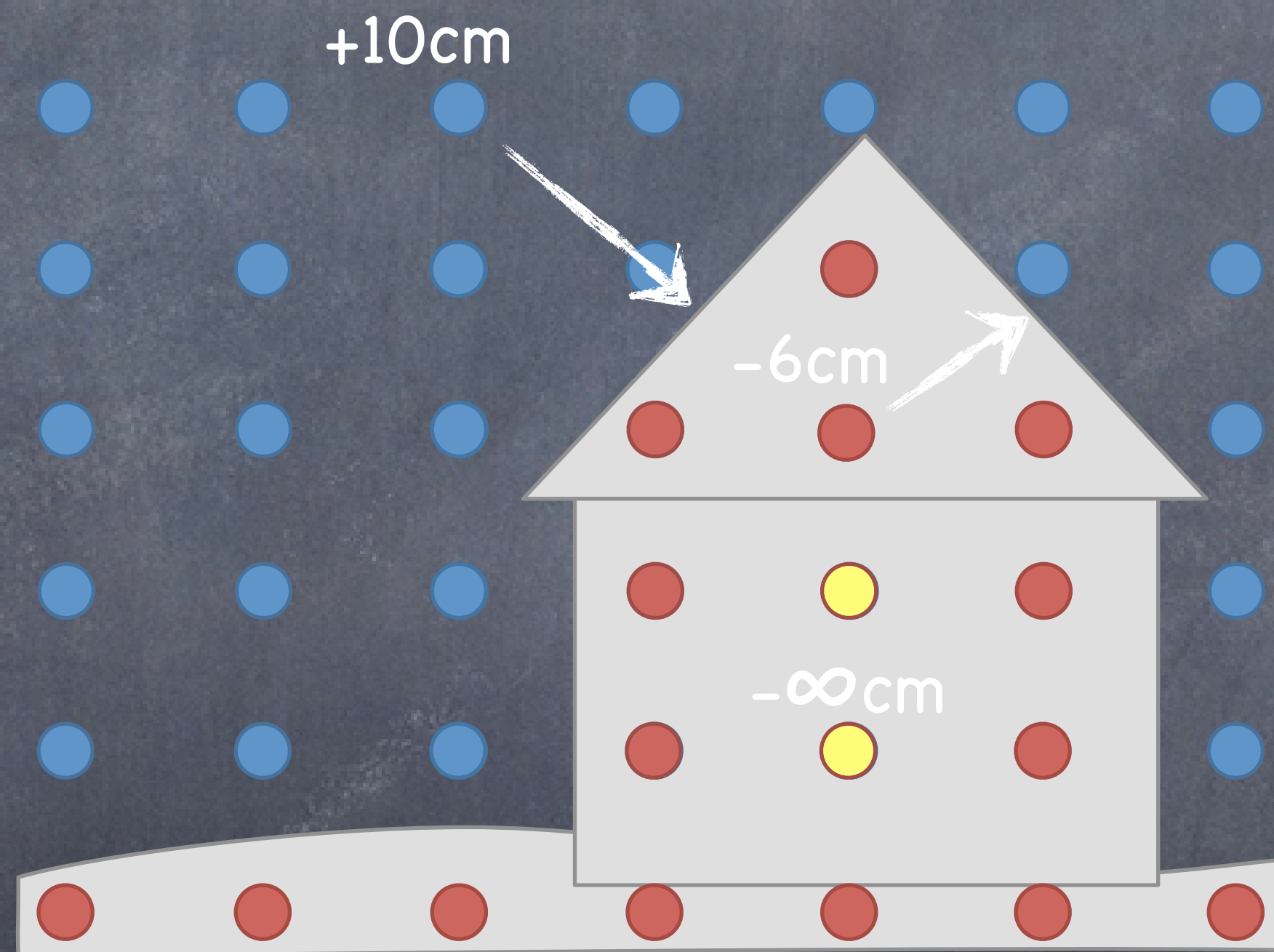
KINECTFUSION

CURRENT/FUTURE

Signed Distance Field

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

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



>0: OUTSIDE

<0: INSIDE

$-\infty$: UNSEEN

INTRO

RECONSTRUCTION BASICS

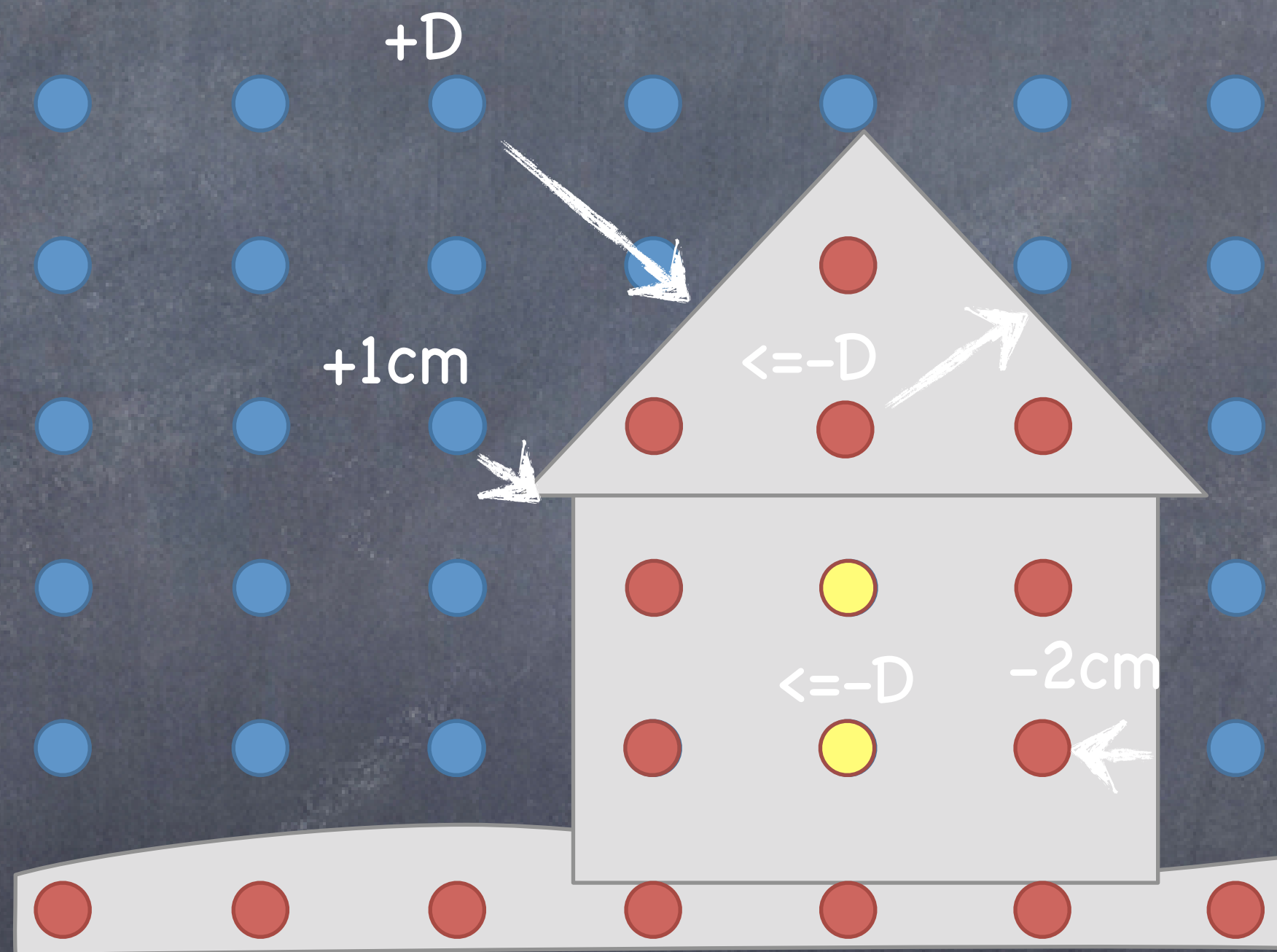
KINECTFUSION

CURRENT/FUTURE

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

INTRO

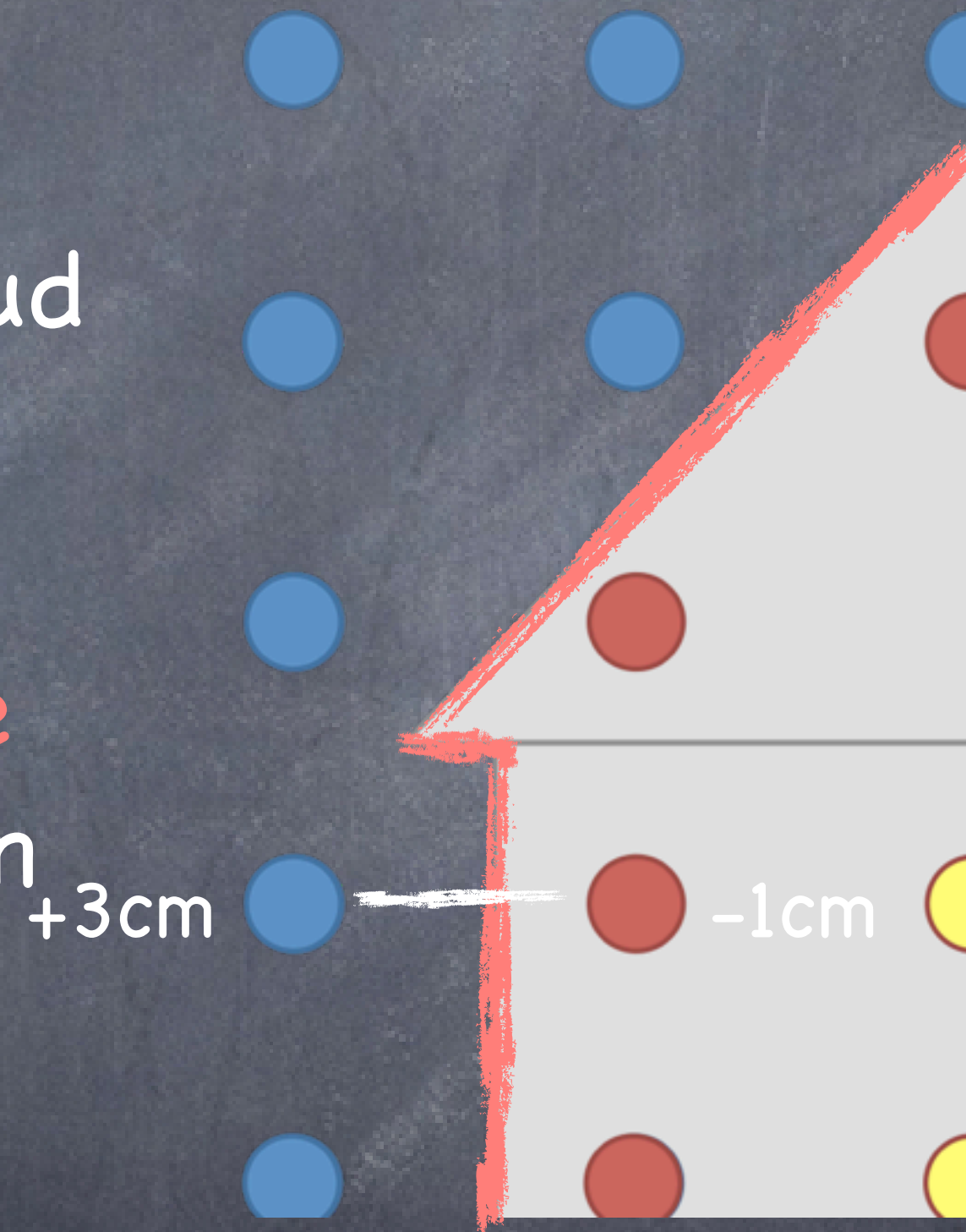
RECONSTRUCTION BASICS

KINECTFUSION

CURRENT/FUTURE

TSDf \rightarrow 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 = $\{\vec{x} \in \mathbb{R}^3 \mid f(\vec{x}) = 0\}$
(where is my distance 0?)
 - Called 0 crossing, or 0 **isosurface**
 - Estimate by trilinear interpolation
- Or add padding
 - iso level d : $\{\vec{x} \in \mathbb{R}^3 \mid f(\vec{x}) = d\}$



INTRO

RECONSTRUCTION BASICS

KINECTFUSION

CURRENT/FUTURE

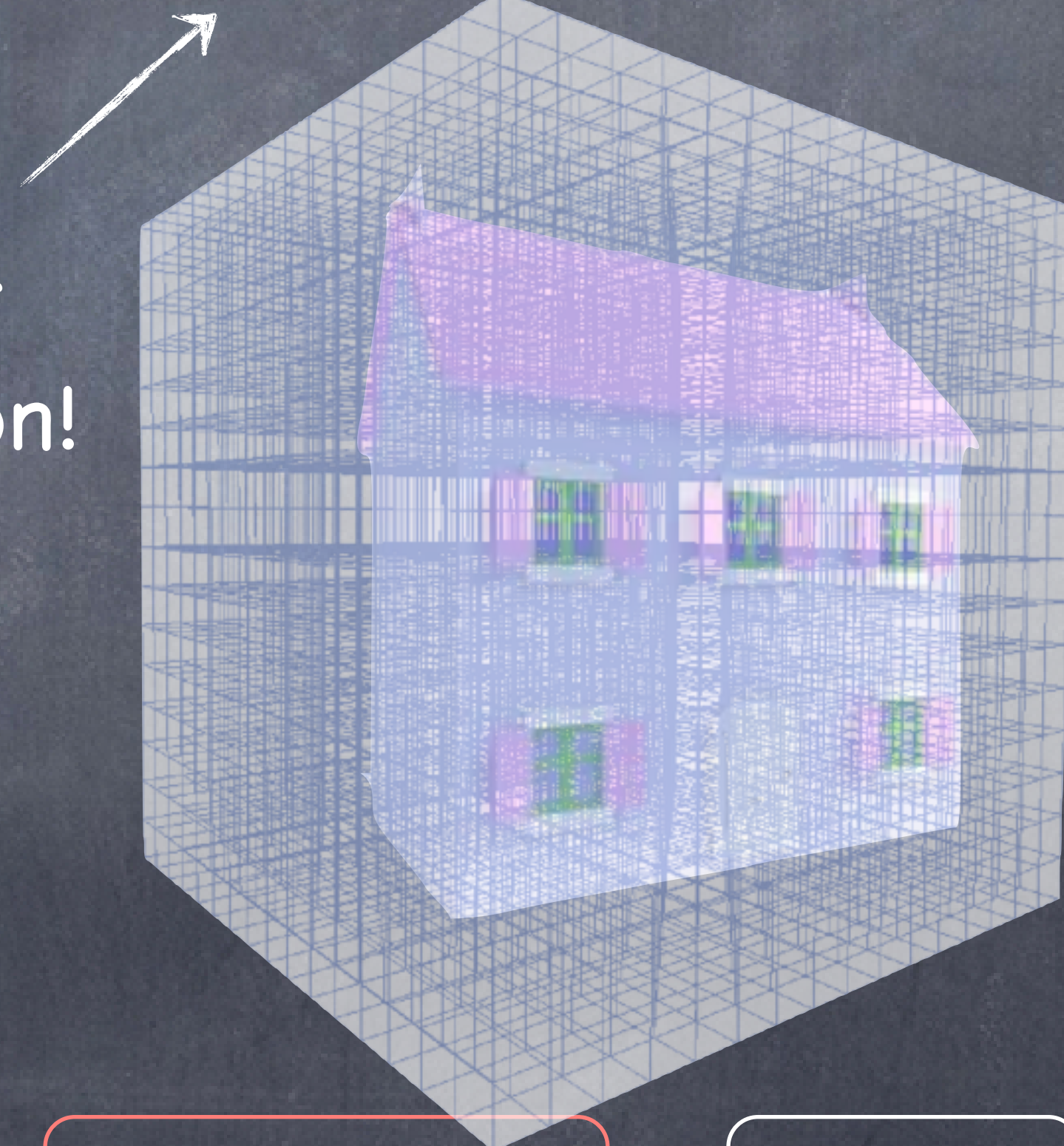
TSDF > Mesh

Mesh: only stores observed surface

TSDF: also stores observed non-surface



Useful for registration!



INTRO

RECONSTRUCTION BASICS

KINECTFUSION

CURRENT/FUTURE

SDF Estimation

- Simple example: stationary sensor

- How far is the wall ("d_w")?

- Assuming Gaussian noise, it will be the average

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



INTRO



RECONSTRUCTION BASICS



KINECTFUSION



CURRENT/FUTURE

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 ("x̄") from the wall?

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

AVERAGE ALL READINGS
WHICH PASS THROUGH \vec{x}

INTRO



RECONSTRUCTION BASICS



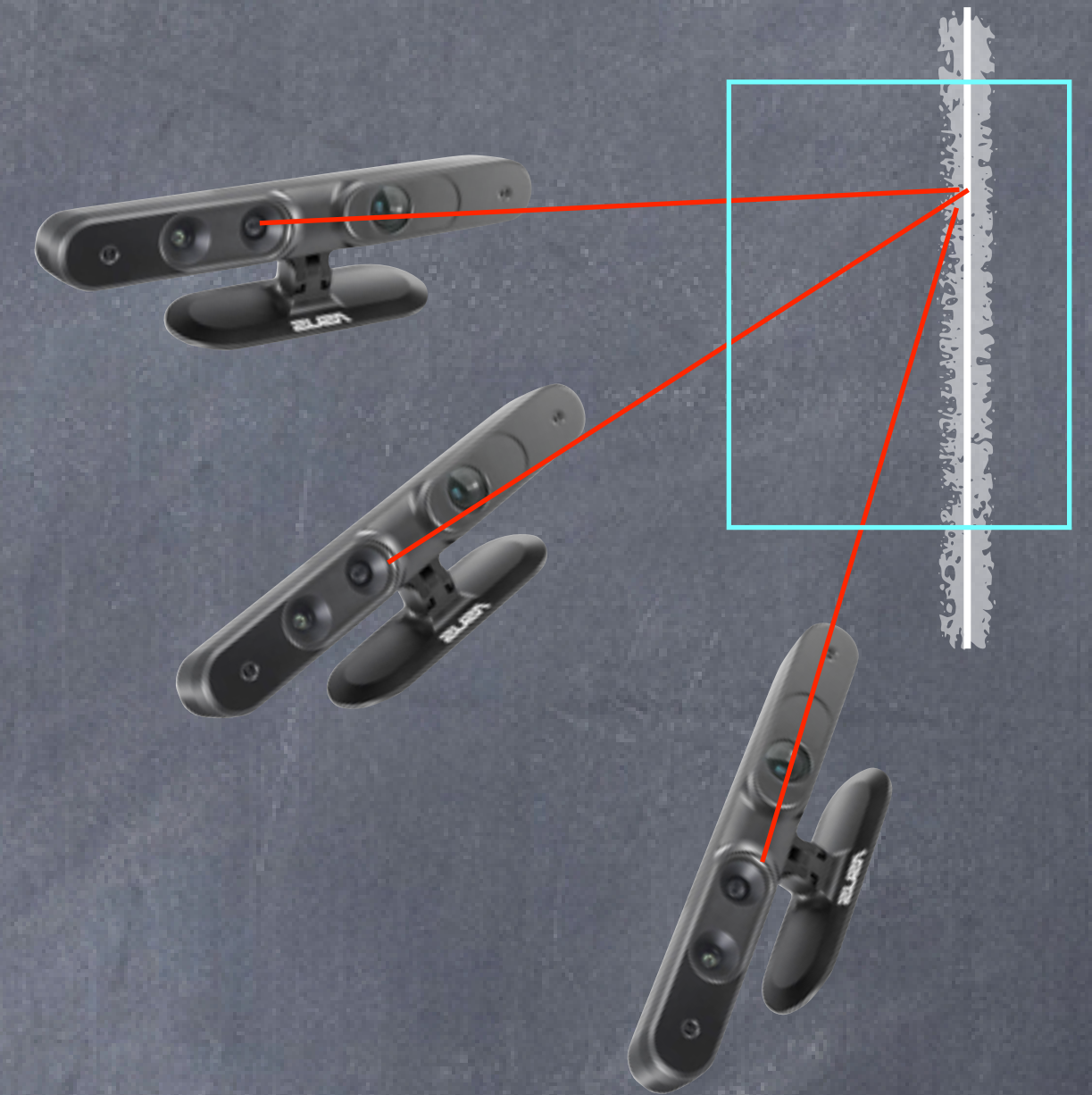
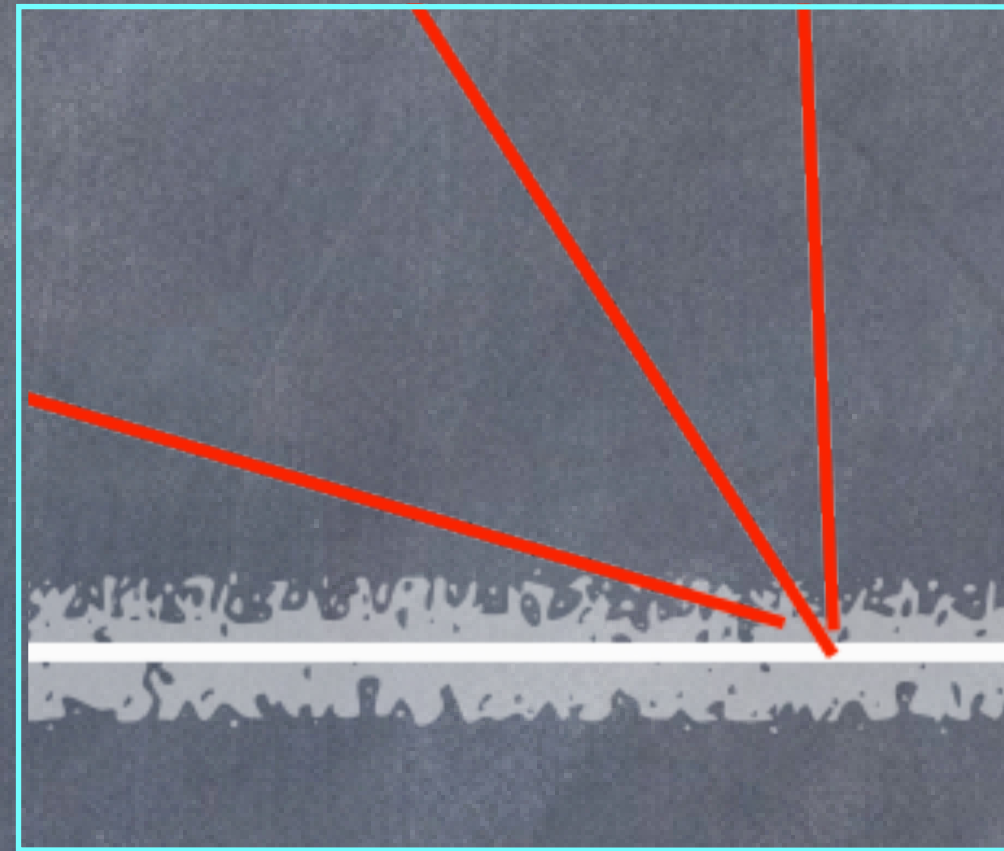
KINECTFUSION



CURRENT/FUTURE

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



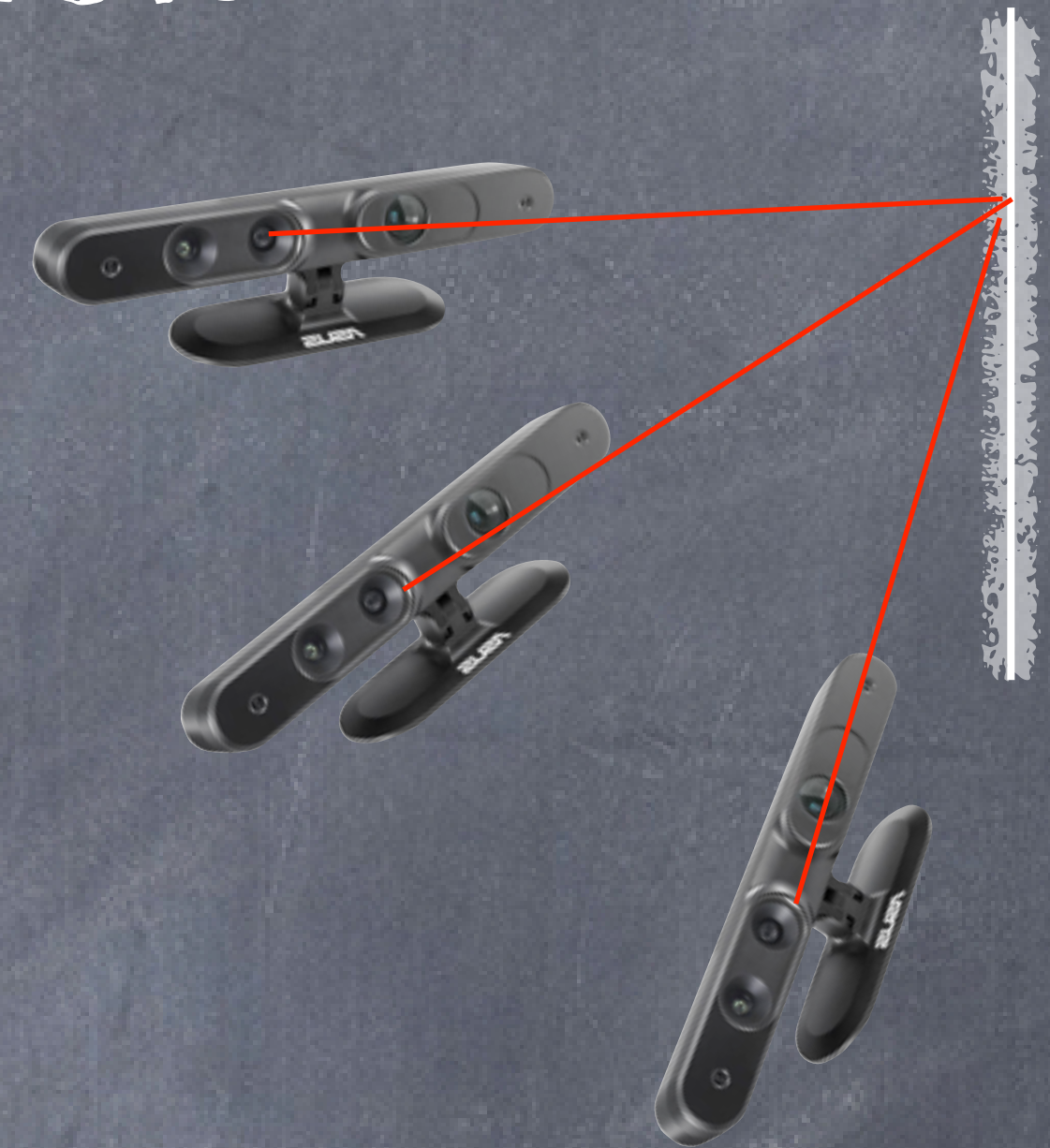
Online SDF Estimation

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

Weighted averaging can be done online

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

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



INTRO



RECONSTRUCTION BASICS



KINECTFUSION



CURRENT/FUTURE

Online TSDF Estimation

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

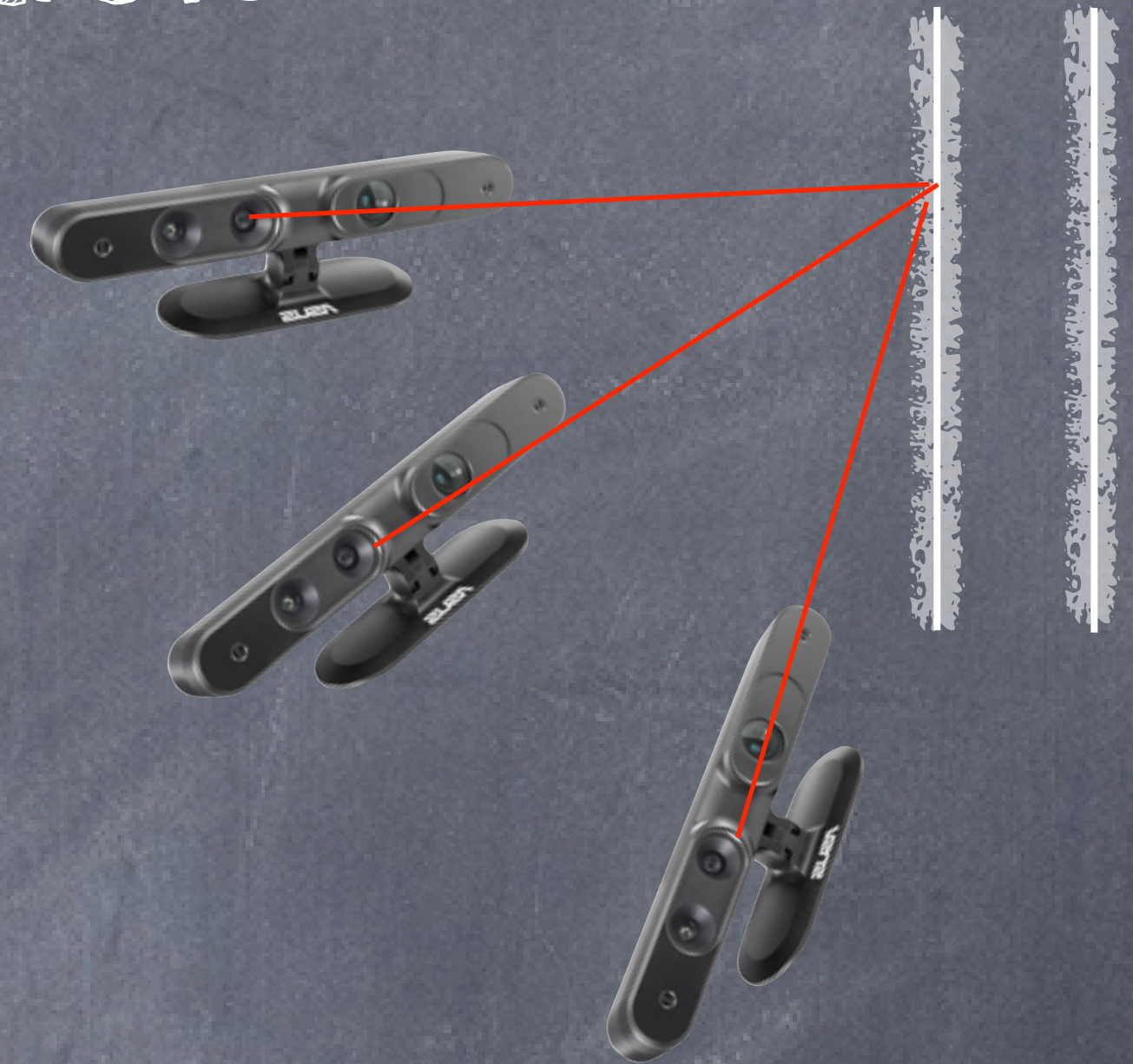
- Weighted averaging can be done online

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

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

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



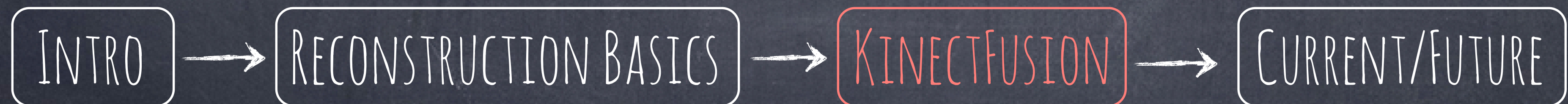
INTRO

RECONSTRUCTION BASICS

KINECTFUSION

CURRENT/FUTURE

Kinect Fusion



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

Abstract

Object recognition system has been developed that uses a class of local image features. The features are invariant to image scaling, translation, and rotation, and partially invariant to illumination changes and affine or 3D projection. These features share similar properties with neurons in inferior temporal cortex that are used for object recognition in primate vision. Features are efficiently detected through a fast filtering approach that identifies stable points in 3D space. Image keys are created that allow for local geometric deformations by representing blurred image gradients in multiple orientation planes and at multiple scales. Image keys are used as input to a nearest-neighbor indexing method that identifies candidate object matches. Final verification is done using a more computationally intensive method.

translation, scaling, and rotation, and partially invariant illumination changes and affine or 3D projection. Previous approaches to local feature generation lacked invariance to scale and were more sensitive to projective distortion illumination change. The SIFT features share a number of properties in common with the responses of neurons in inferior temporal (IT) cortex in primate vision. This paper describes improved approaches to indexing and model verification.

The scale-invariant features are efficiently identified using a staged filtering approach. The first stage identifies key locations in scale space by looking for locations that are maxima or minima of a difference-of-Gaussian function. Each point is used to generate a feature vector that describes the local image region. The feature vector is then compared to its scale-invariant counterparts.

SIFT '99
UKF '97

Sebastian Thrun
Michael Montemerlo

Stanford AI Lab
Stanford University
(thrun, mende)@stanford.edu

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

Abstract

This article presents GraphSLAM, a unifying algorithm for the offline SLAM problem. GraphSLAM is closely related to a recent sequence of research papers on applying optimization techniques to SLAM problems. It transforms the SLAM posterior into a graphical network, representing the log-likelihood of the data. It then reduces this graph using variable elimination techniques, arriving at a lower-dimensional problem that is then solved using conventional optimization techniques. As a result, GraphSLAM can generate maps with 10^5 or more features. The paper discusses a greedy algorithm

prisingly, some of the primary work in this area has emerged from a number of different scientific fields, such as photogrammetry, computer vision (Tomasi and Kanade 1992; Pollefeys, Koch, and Gool 1998; Soatto and Brockett 1998), computer graphics (Levoy 1999; Rusinkiewicz and Levoy 2001), and robotics (Dissanayake et al. 2001).

In the SLAM community (SLAM is short for simultaneous localization and mapping), filter techniques such as the well-studied extended Kalman filter (EKF) have become a method of choice for model acquisition. The EKF was introduced mathematically by Chebotarev et al. (1986), and is widely

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:



Multimedia Archives

FAB-MAP '08

A New Extension of the Kalman Filter to Nonlinear Systems

Simon J. Julier Jeffrey K. Uhlmann
siju@robots.ox.ac.uk 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

ABSTRACT

The Kalman filter (KF) is one of the most widely used methods for tracking and estimation due to its simplicity, optimality, tractability and robustness. However, the application of the KF to nonlinear systems can be difficult. The most common approach is to use the Extended Kalman Filter (EKF) which simply linearises all nonlinear models so that the traditional linear Kalman filter can be applied. Although the EKF (in its many forms) is a widely used filtering strategy, over thirty years of experience with it has led to a general consensus within the tracking and control community that it is difficult to implement, difficult to tune, and only reliable for systems which are almost linear on the time scale of the update intervals.

In this paper a new linear estimator is developed and demonstrated. Using the principle that a set of discrete-sampled points can be used to parameterise mean and covariance, the estimator yields performance equivalent to the KF for linear systems yet generalises elegantly to nonlinear systems without the need for linearisation. The new EKF, which we call the Unscented Kalman Filter (UKF), is shown to be superior to that of the EKF. We also discuss the application of the UKF to nonlinear systems with state-dependent process noise.

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
mende@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

Abstract

The ability to simultaneously localize a robot and accurately map its surroundings is considered by many to be a key prerequisite of truly autonomous robots. However, few approaches to this problem scale up to handle the very large number of landmarks present in real environments. Kalman filter-based algorithms, for example, require time quadratic in the number of landmarks to incorporate each sensor observation. This paper presents FastSLAM, an algorithm that recursively estimates the full posterior distribution over robot pose and landmark locations, yet scales logarithmically with the number of landmarks in the map. This algorithm is based on an exact factorization of the posterior into a product of conditional landmark distributions and a distribution over robot paths. The algorithm has been run successfully on as many as 50,000 landmarks, environments far beyond the reach of previous approaches. Experimental results demonstrate the advantages and limitations of the FastSLAM algorithm on both simulated and real-world data.

Introduction

A key limitation of EKF-based approaches is their computational complexity. Sensor updates require time quadratic in the number of landmarks K to compute. This complexity stems from the fact that the covariance matrix maintained by the Kalman filters has $O(K^2)$ elements, all of which must be updated even if just a single landmark is observed. The quadratic complexity limits the number of landmarks that can be handled by this approach to only a few hundred—whereas natural environment models frequently contain millions of features. This shortcoming has long been recognized by the research community [6, 8, 14].

In this paper we approach the SLAM problem from a Bayesian point of view. Figure 1 illustrates a generative probabilistic model (dynamic Bayes network) that underlies the rich corpus of SLAM literature. In particular, the robot poses, denoted $\theta_1, \theta_2, \dots, \theta_t$, evolve over time as a function of the robot controls, denoted u_1, \dots, u_t . Each of the landmark measurements, denoted z_1, \dots, z_t , is a function of the position θ_t of the landmark measured and of the robot pose at the time the measurement was taken. From this diagram it is evident that the SLAM problem exhibits important conditional independencies. In particular, knowledge of the robot's path $\theta_1, \theta_2, \dots, \theta_t$ is sufficient to compute the likelihood of any measurement z_t .

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

Abstract—We introduce incremental smoothing and mapping (ISAM), a novel approach to the problem of simultaneous localization and mapping (SLAM) that addresses the data association problem and allows real-time application in large-scale environments. We employ smoothing to obtain the complete trajectory and map without the need for any approximations, exploiting the natural sparsity of the smoothing information matrix. A QR-factorization of this information matrix is at the heart of our approach. It provides efficient access to the exact covariances as well as to conservative estimates that are used for online data association. It also allows recovery of the exact trajectory and map at any given time by back-substitution. Instead of refactoring in each step, we update the QR-factorization whenever a new measurement arrives. We analyze the effect of loops, and show how our approach extends to the non-linear case. Finally, we provide experimental validation of the overall non-linear algorithm based on the standard Victoria Park data set with unknown correspondences.

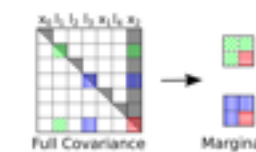


Fig. 1. Only a small number of entries of the dense covariance matrix are of interest for data association. In this example, the marginals between the last pose θ_t and the landmarks l_1 and l_2 are retrieved. The entries that need to be calculated in general are marked in gray. Only the triangular blocks on the diagonal and the last block columns are needed, due to symmetry. Based on our factored information matrix representation, the last column can be obtained by simple back-substitution. The blocks on the diagonal can either be calculated exactly by only calculating the entries corresponding to non-zero in the sparse factor R , or approximated by conservative estimates for online data association.

Filtering algorithms and the QR-factorization are the most widely used approach to real-time

ISAM '07

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

The practical applications of 3D model acquisition are manifold. In this paper, we present our RGB-D SLAM system, i.e., an approach to generate colored 3D models of objects and indoor scenes using the hand-held Microsoft Kinect sensor. Our approach consists of four processing steps as illustrated in Figure 1. First, we extract SURF features from the incoming color images. Then we match these features against features from the previous images. By evaluating the depth images at the locations of these feature points, we obtain a set of point-wise 3D correspondences between any two frames. Based on these correspondences, we estimate the relative transformation between the frames using RANSAC. The third step is to improve this initial estimate using a variant of the ICP algorithm [1]. As the pair-wise pose estimates between frames are not necessarily globally consistent, we optimize the resulting pose graph in the fourth step using a pose graph solver [4]. The output of our algorithm is a globally consistent 3D model of the perceived environment, represented as a colored point cloud.

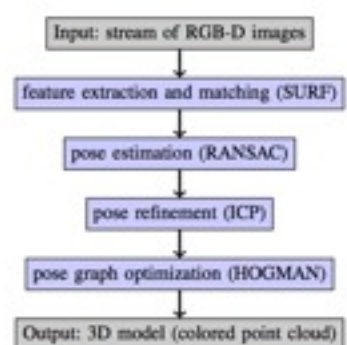


Fig. 1: The four processing steps of our approach. Our approach generates colored 3D environment models from images acquired with a hand-held Kinect sensor.

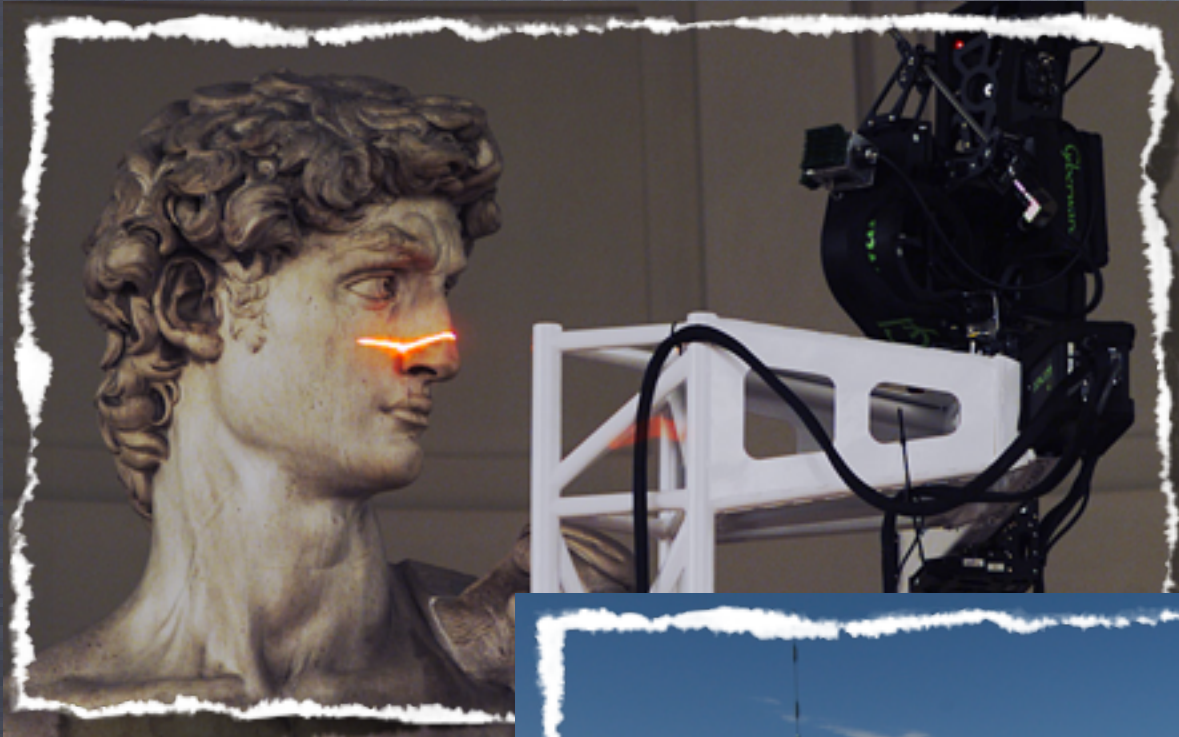
INTRO

RECONSTRUCTION BASICS

KINECTFUSION

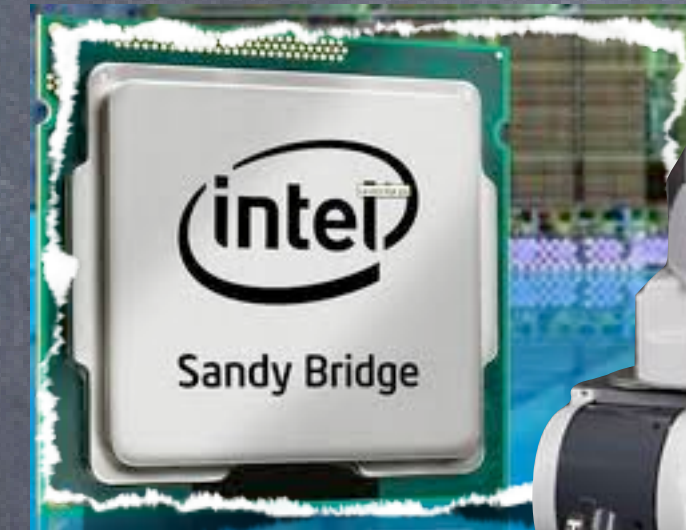
CURRENT/FUTURE

1996 to 2011: Technology

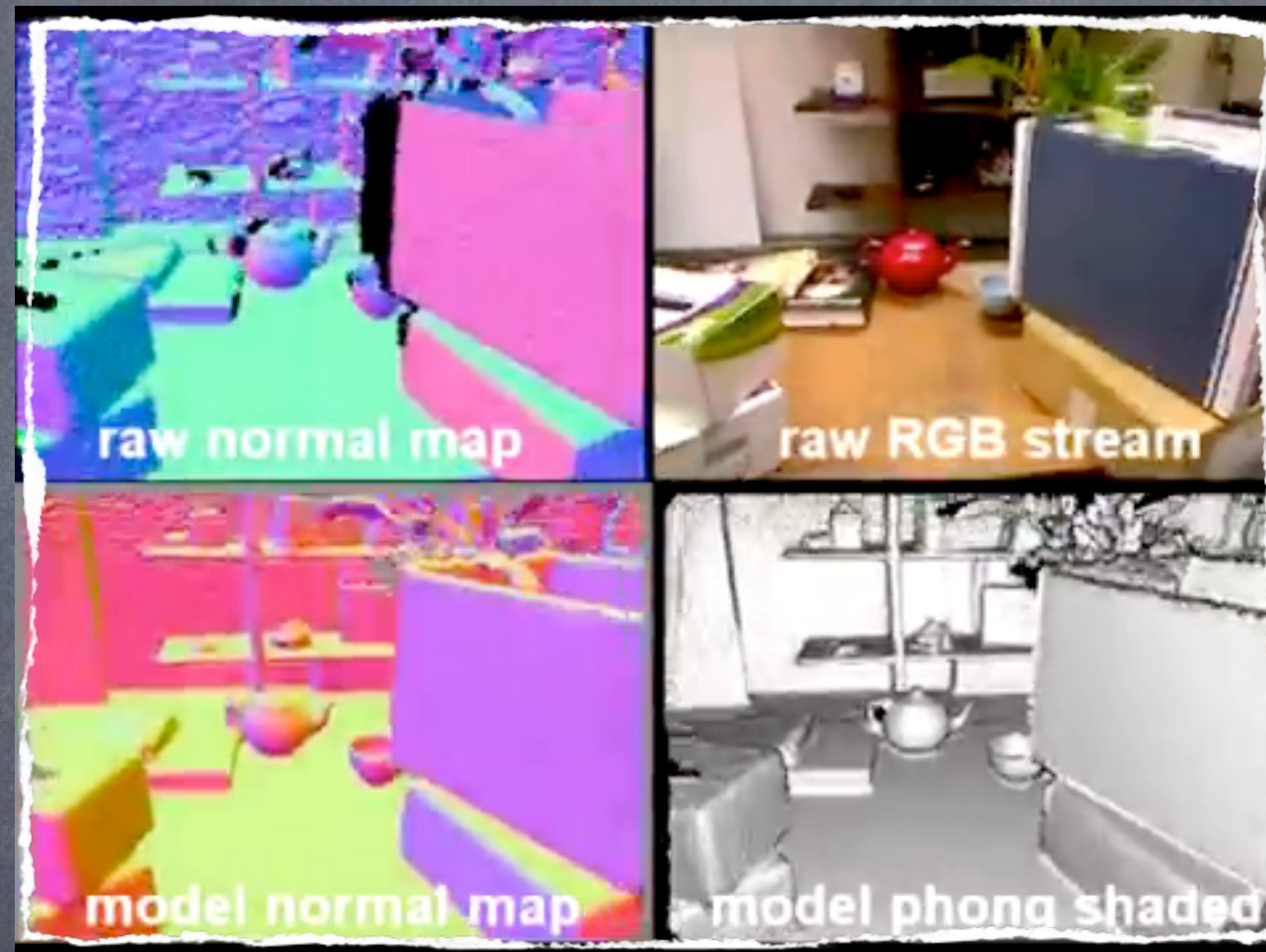


Inexpensive, noisy
data

Powerful
Processing



Kinect Fusion



Newcomb et al, ISMAR '11

Izadi et al, SIGGRAPH '11

INTRO



RECONSTRUCTION BASICS



KINECTFUSION



CURRENT/FUTURE

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:

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

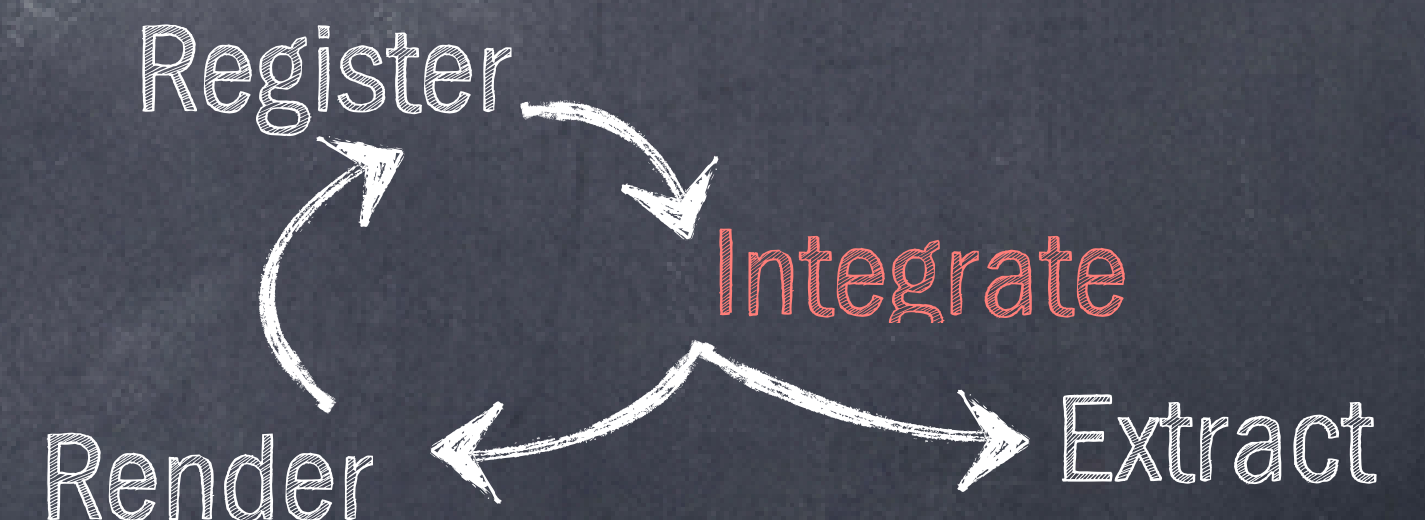
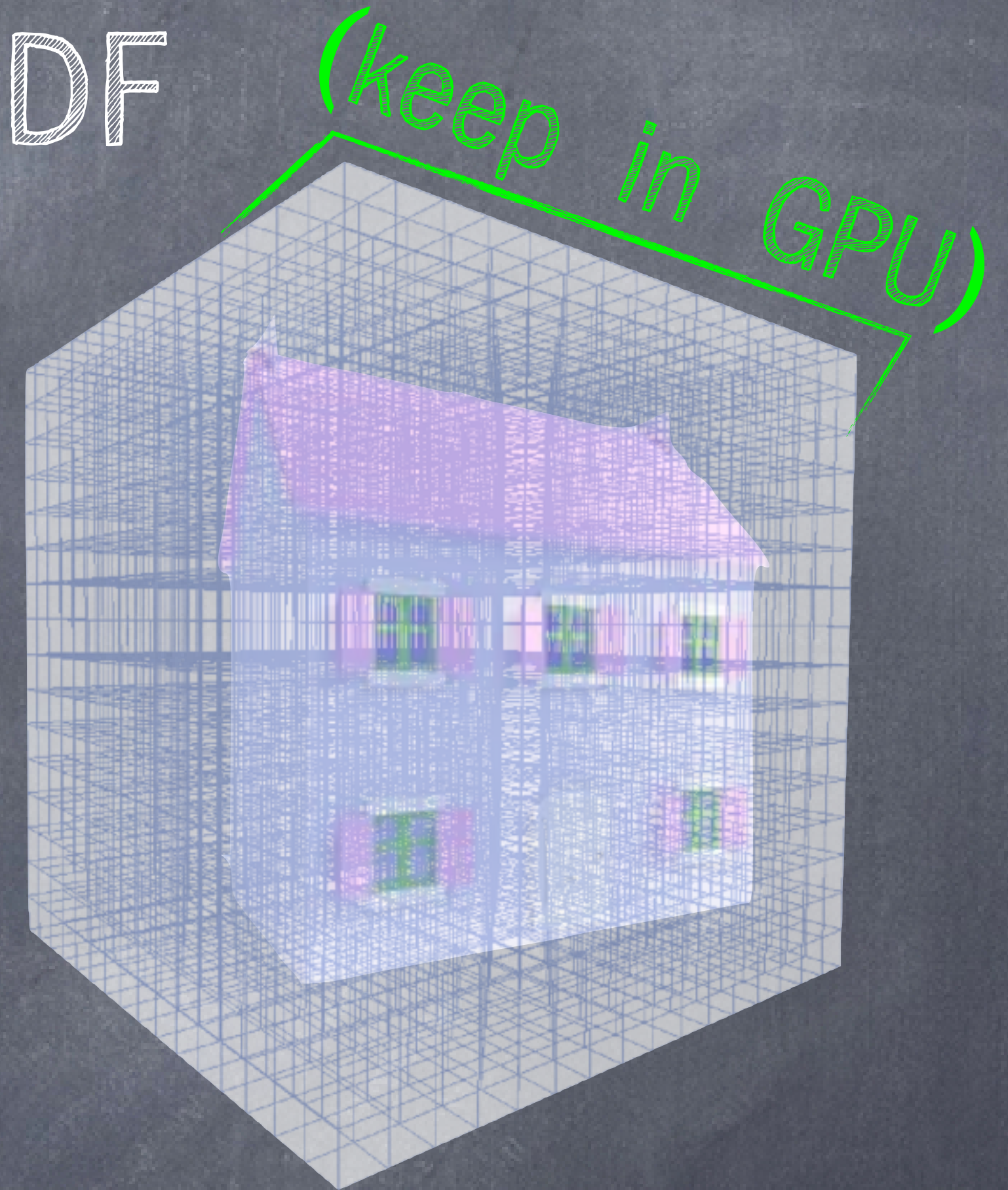
$$w_{\vec{x}} \leftarrow w_t + w_{\vec{x}}$$

- "Simple works well enough"

$$w_t = 1$$

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

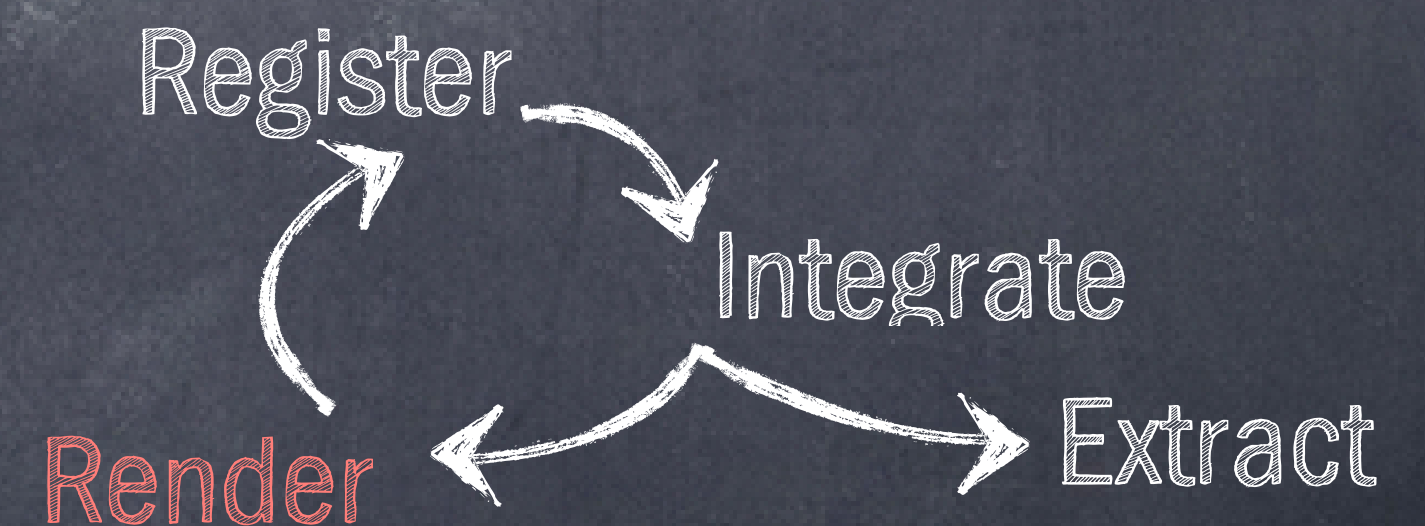
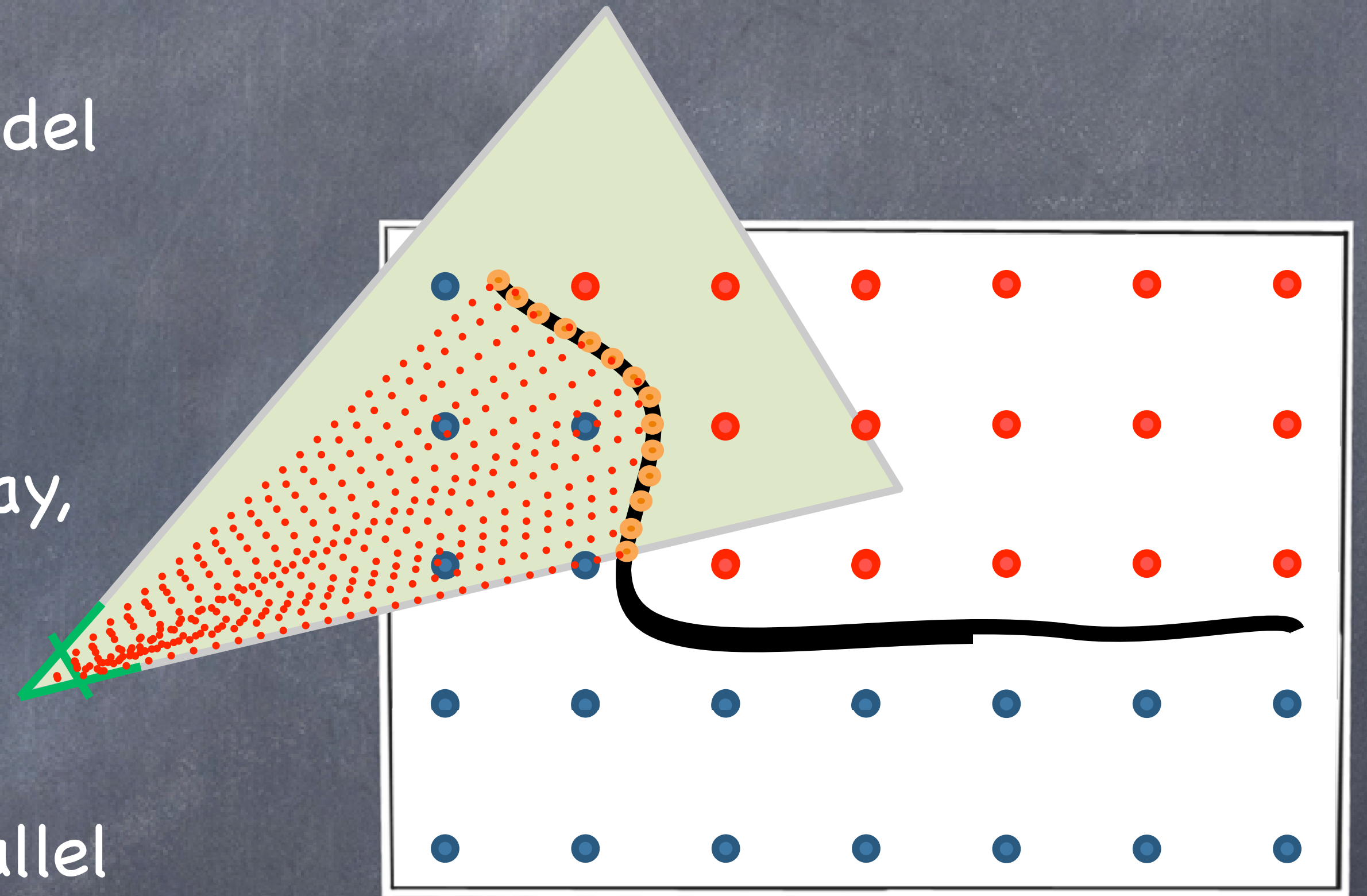
- March over all voxels in parallel;
project center to depth map pixels



Rendering: Raycasting

- See what the current model should 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

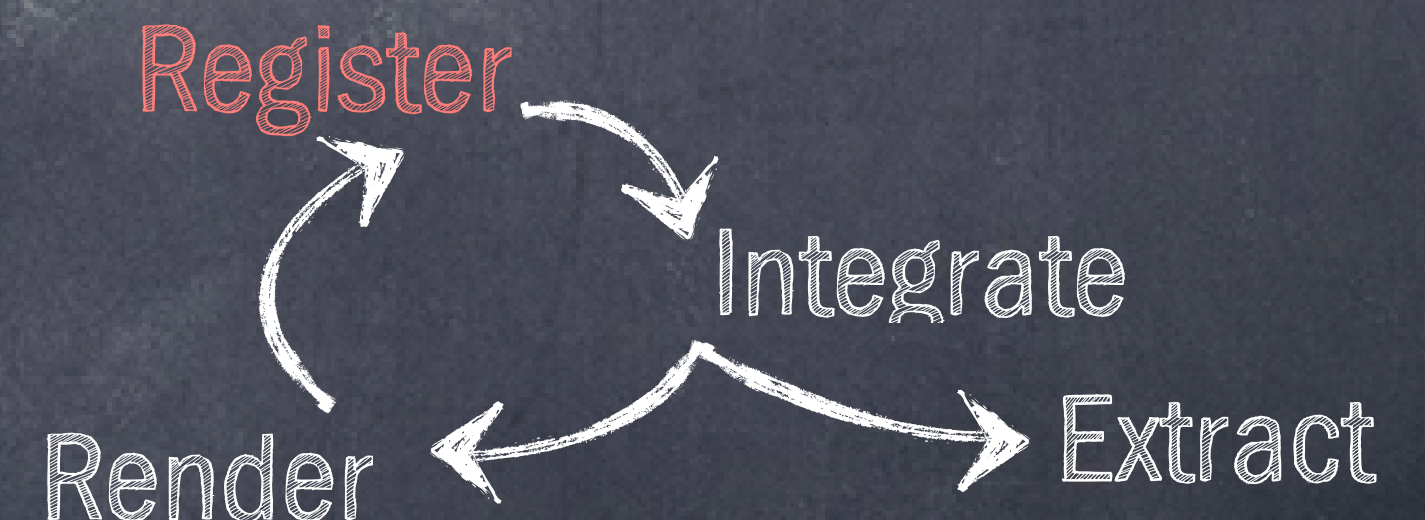
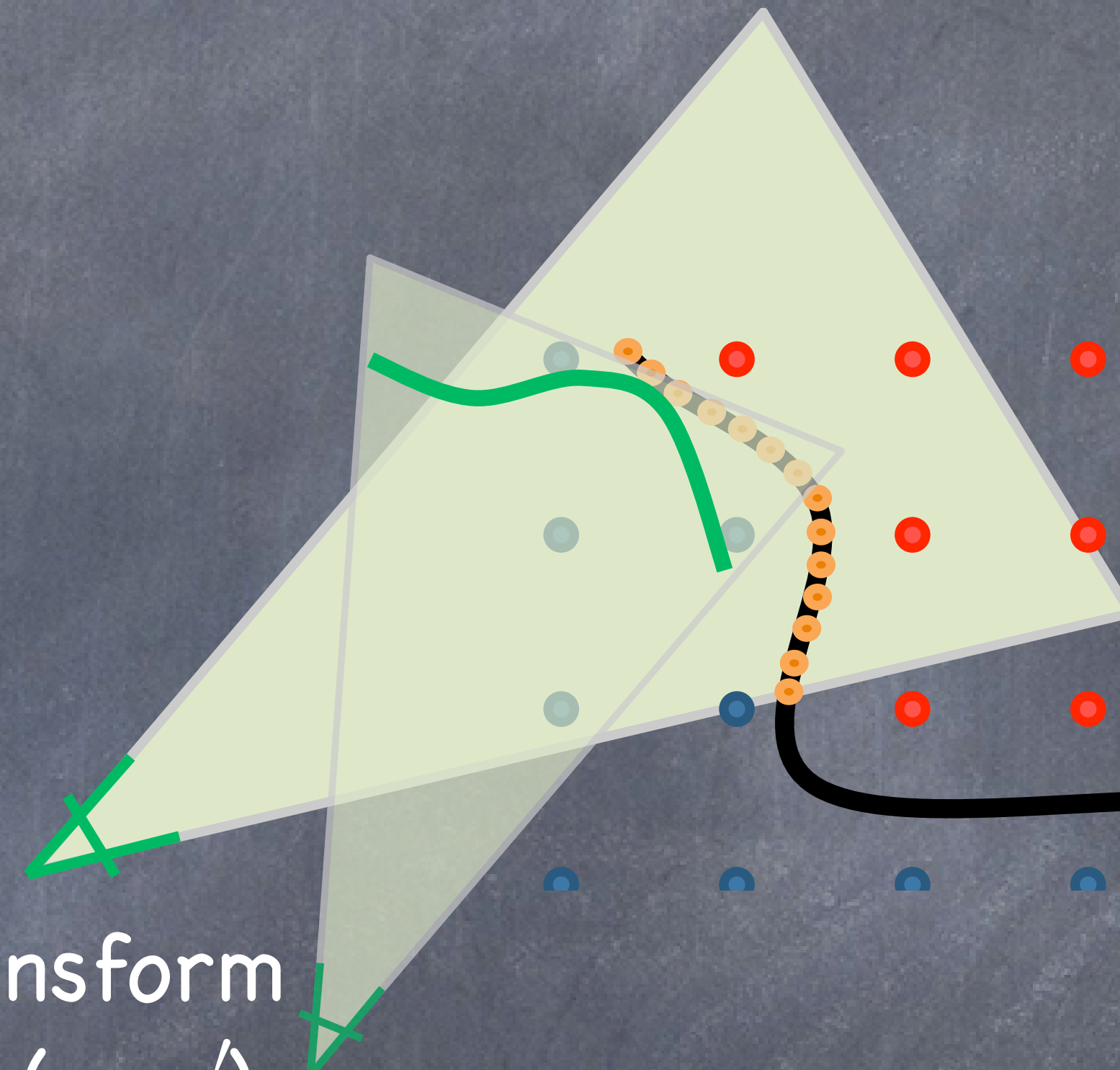
- Approximates:

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

for p' drawn from continuous surface

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

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



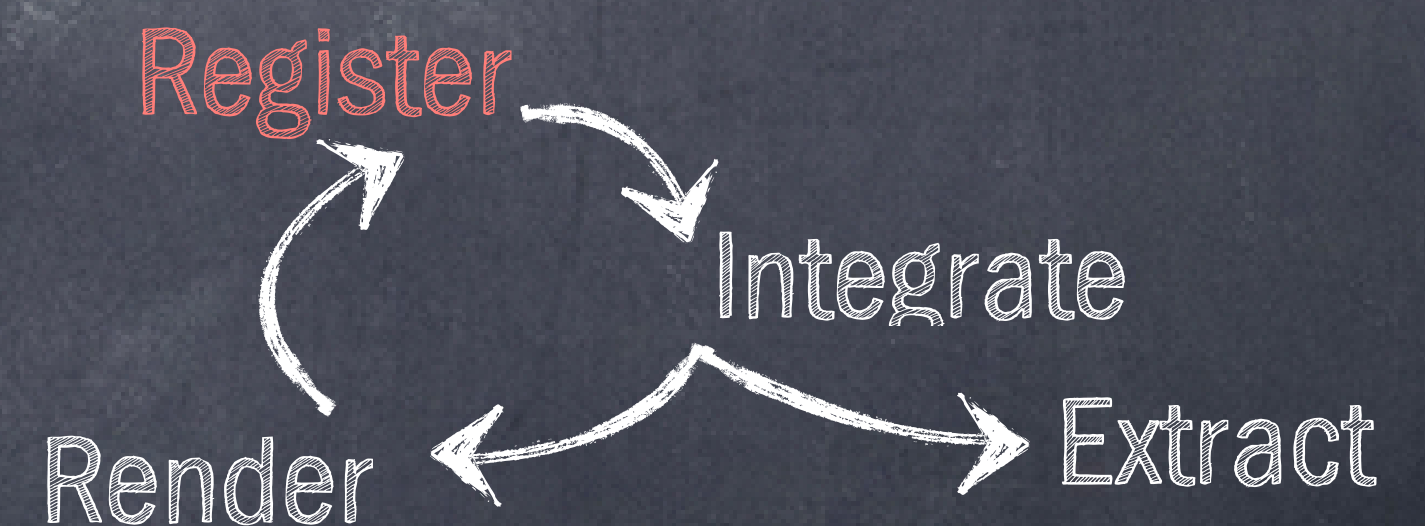
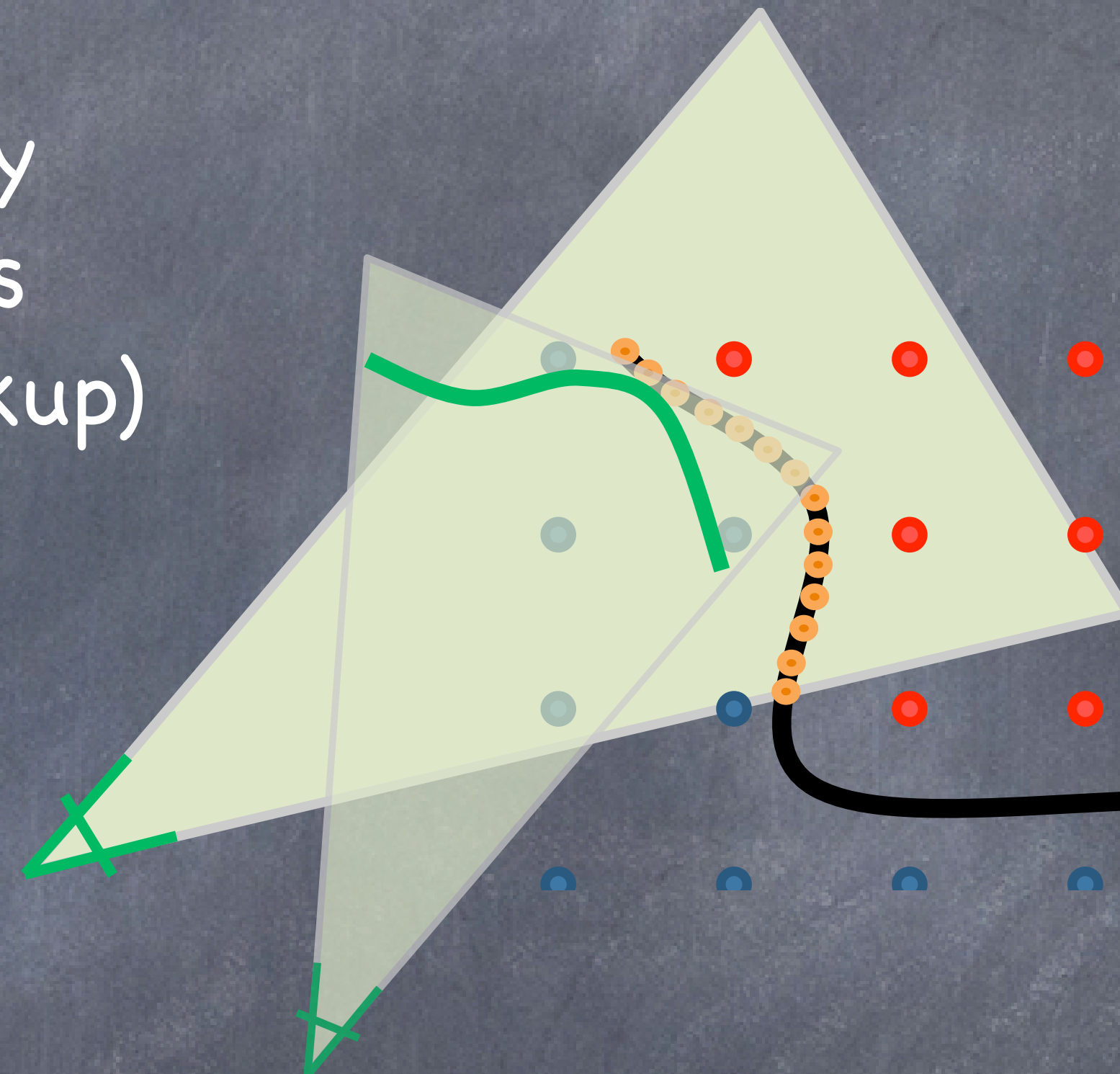
Registering: Point-to-Plane ICP

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

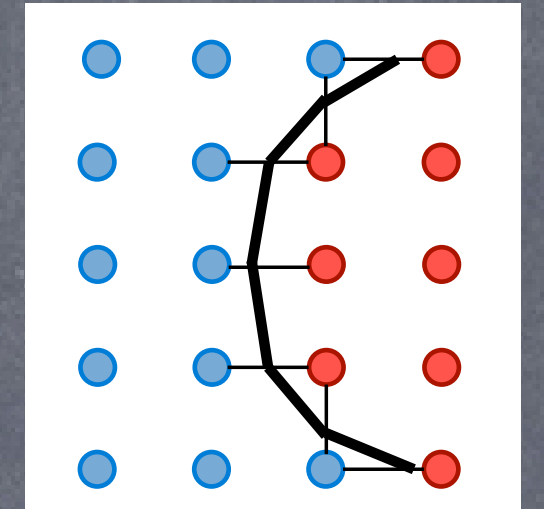
(GPU-friendly)

- Transform found by linearizing rotation matrix and solving linear system

(GPU-friendly)



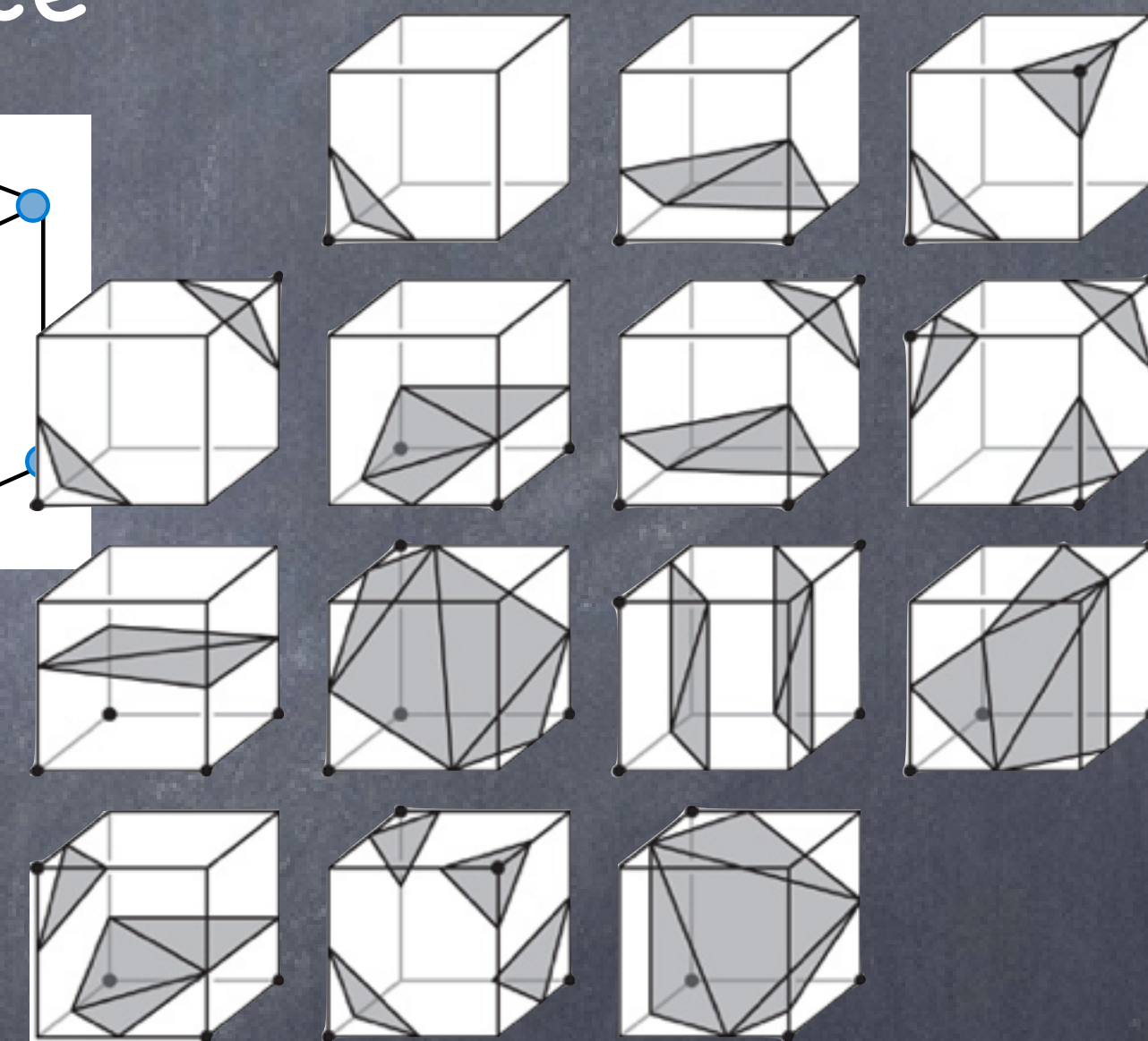
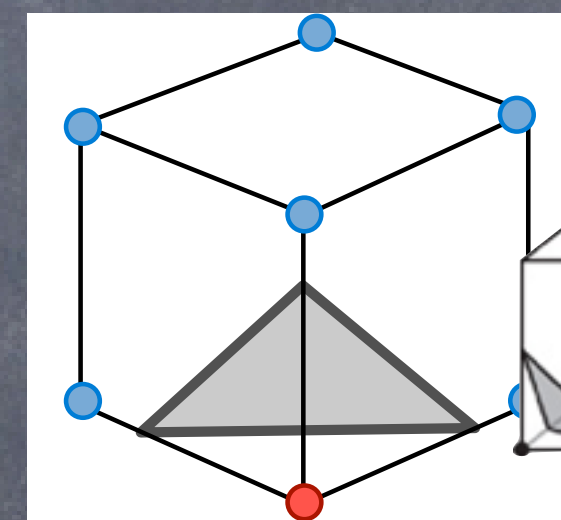
Extraction: Marching Cubes



- Classic method for finding 0 crossing

- Voxel centers have **positive** or **negative** distance

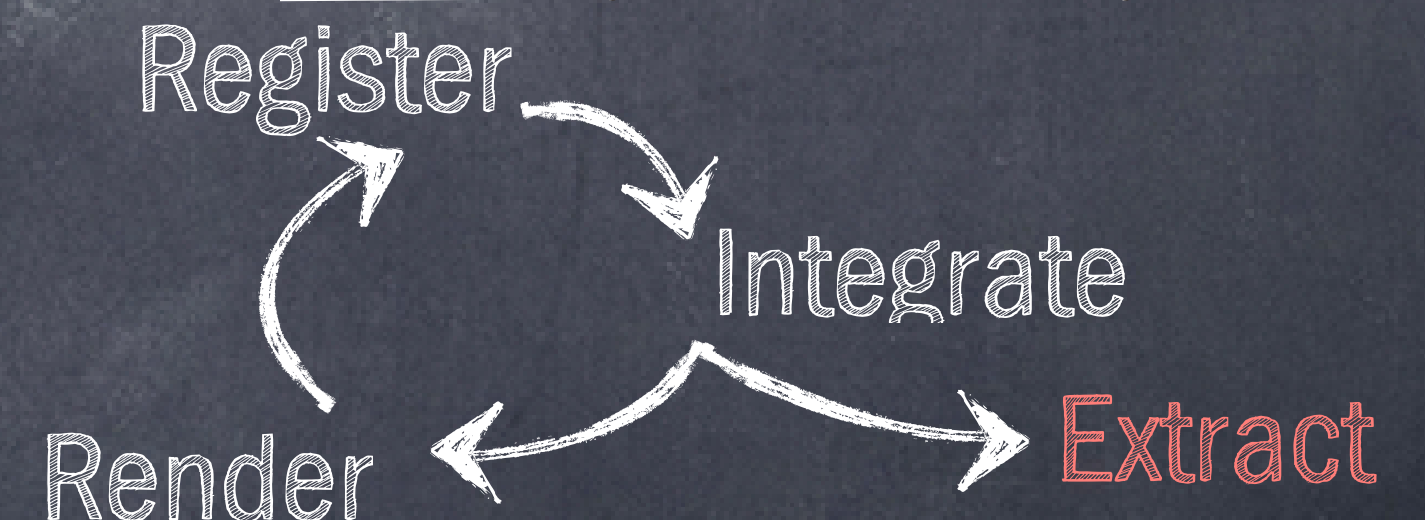
- 8 neighboring voxels makes a "cube"



- 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)

All on GPU

=

Realtime

Register

Simple, known technique

Integrate

(Online TSDF integration)

Simple, known technique

(Run marching cubes)

Extract

Render

(Raycast from the TSDF)

Simple, known technique

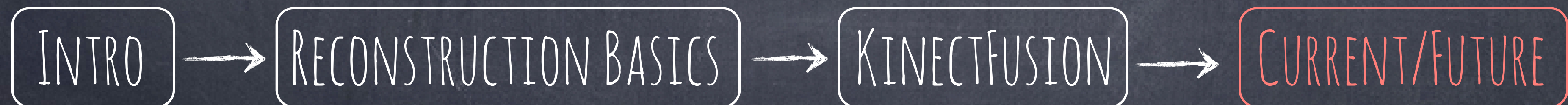
INTRO

RECONSTRUCTION BASICS

KINECTFUSION

CURRENT/FUTURE

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

SOLVABLE!

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

PRECISION

IMPROVABLE!

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

DRIFT

VERY UNCLEAR!

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

INTRO



RECONSTRUCTION BASICS



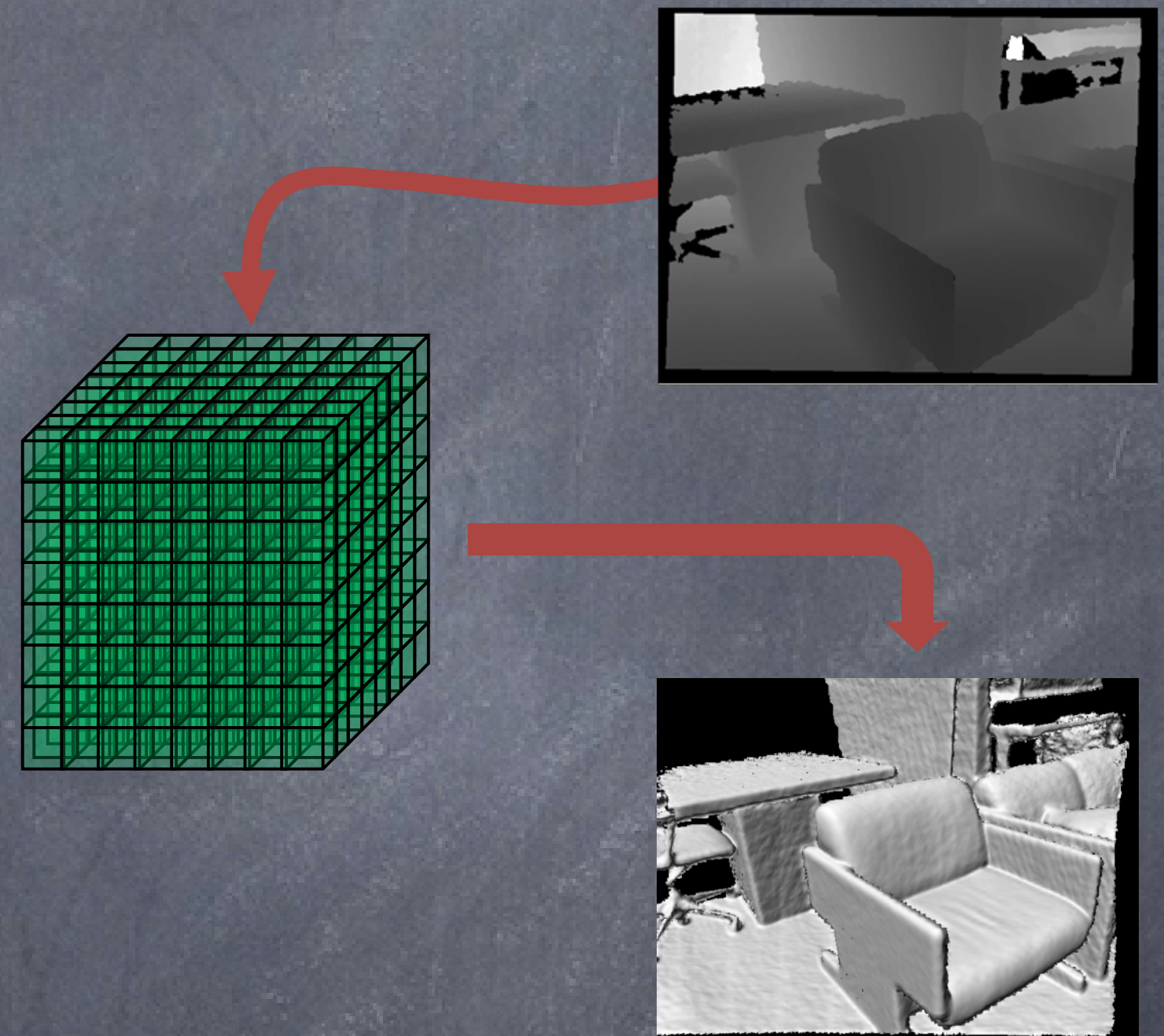
KINECTFUSION



CURRENT/FUTURE

Scalability: Intuition

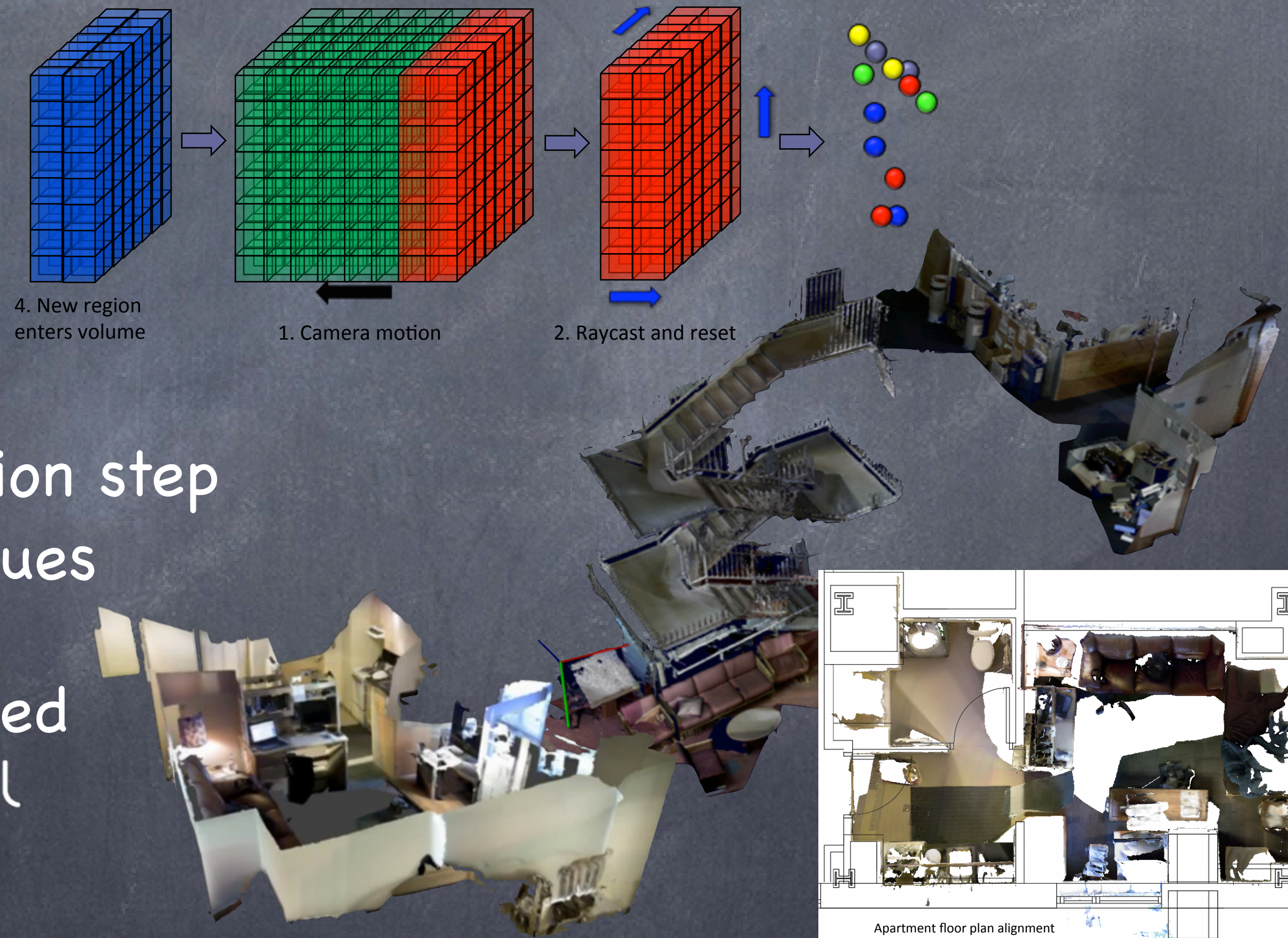
- Kinect Fusion averages over all space, all the time
- GPU Memory \sim 1GB
- $(1\text{GB}) / (\text{sizeof(float } d) + \text{sizeof(float } w))$
= $512 \times 512 \times 512$ voxels
= $3\text{m} \times 3\text{m} \times 3\text{m}$ (for 6mm voxels)
- Space goes with length^3 . Moore's law won't solve 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

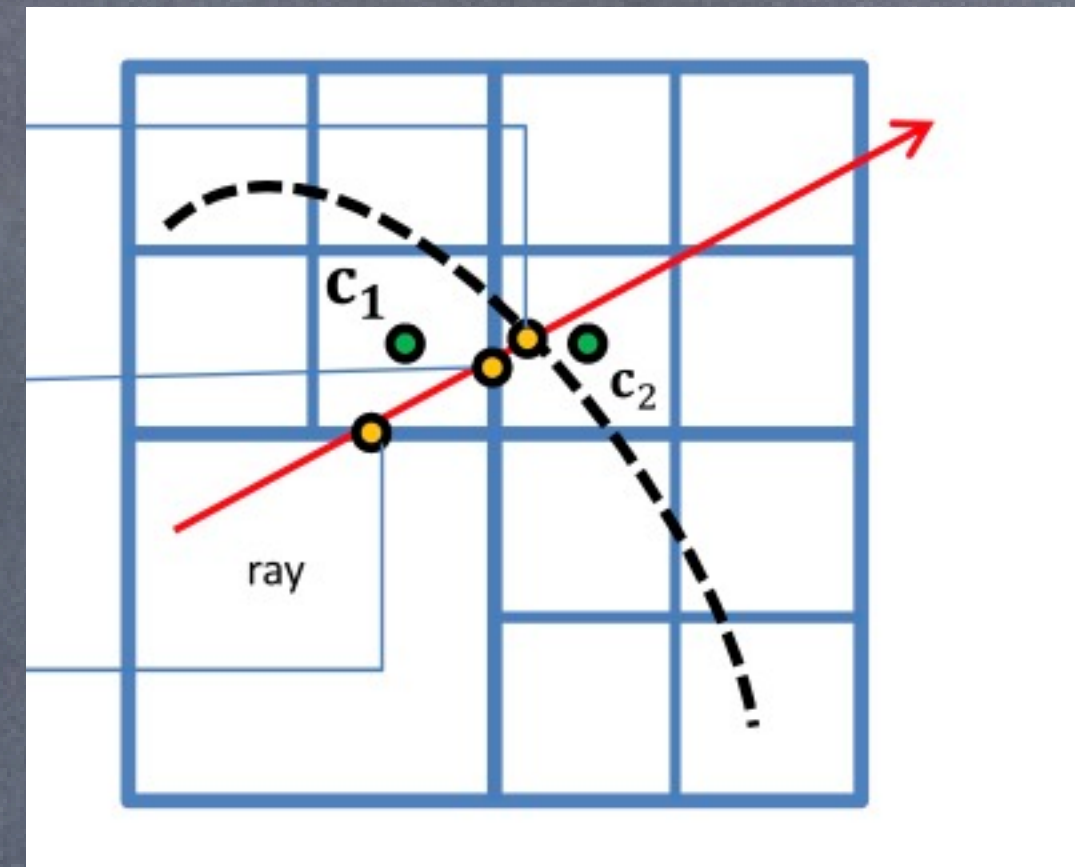


- Added color to registration step
 - Helps with precision issues
- Similar concepts developed independently in  pcl (kinfu_large_scale_app)

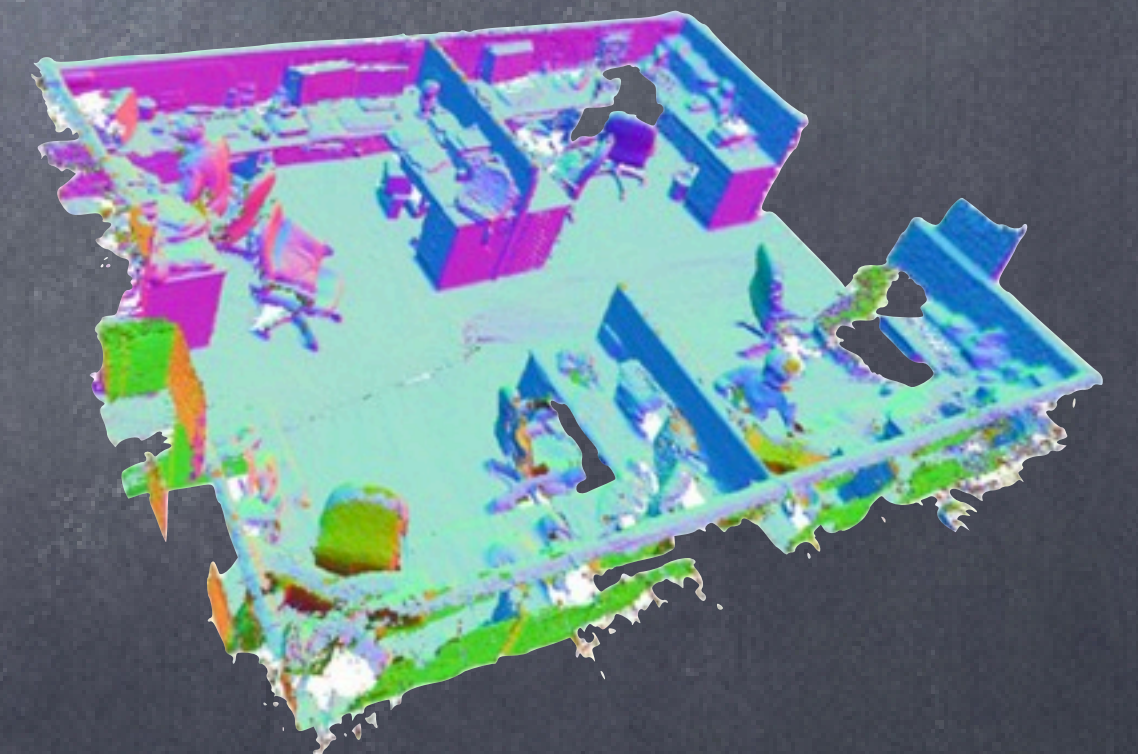
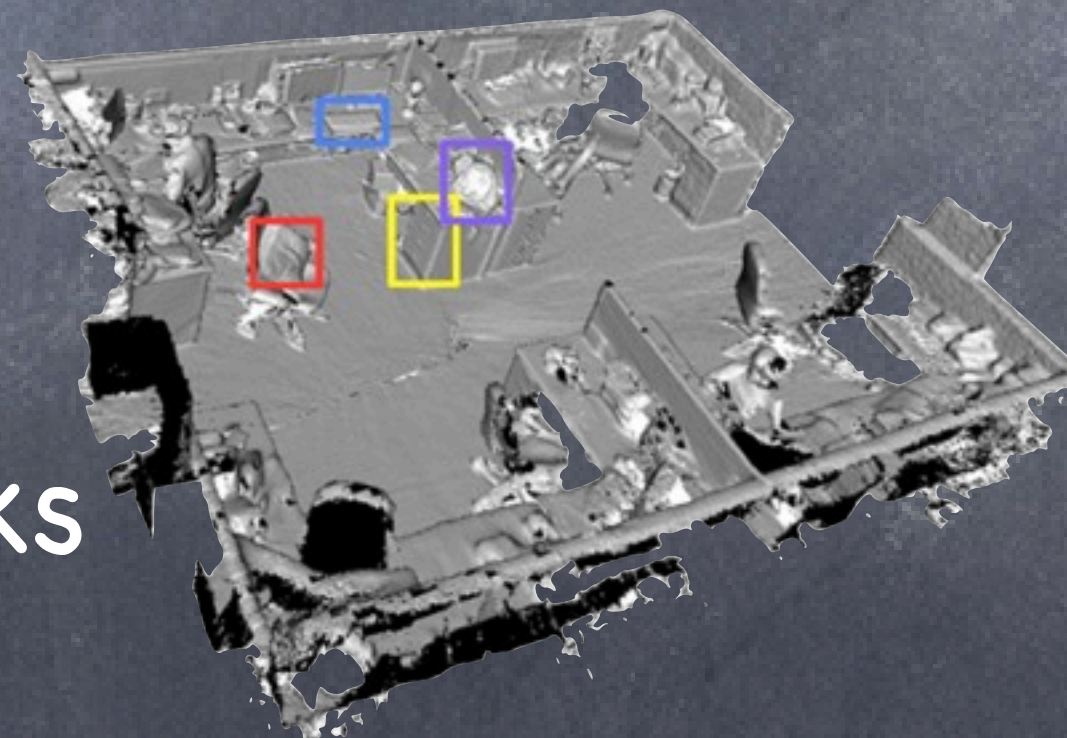
Scalability: Octrees

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



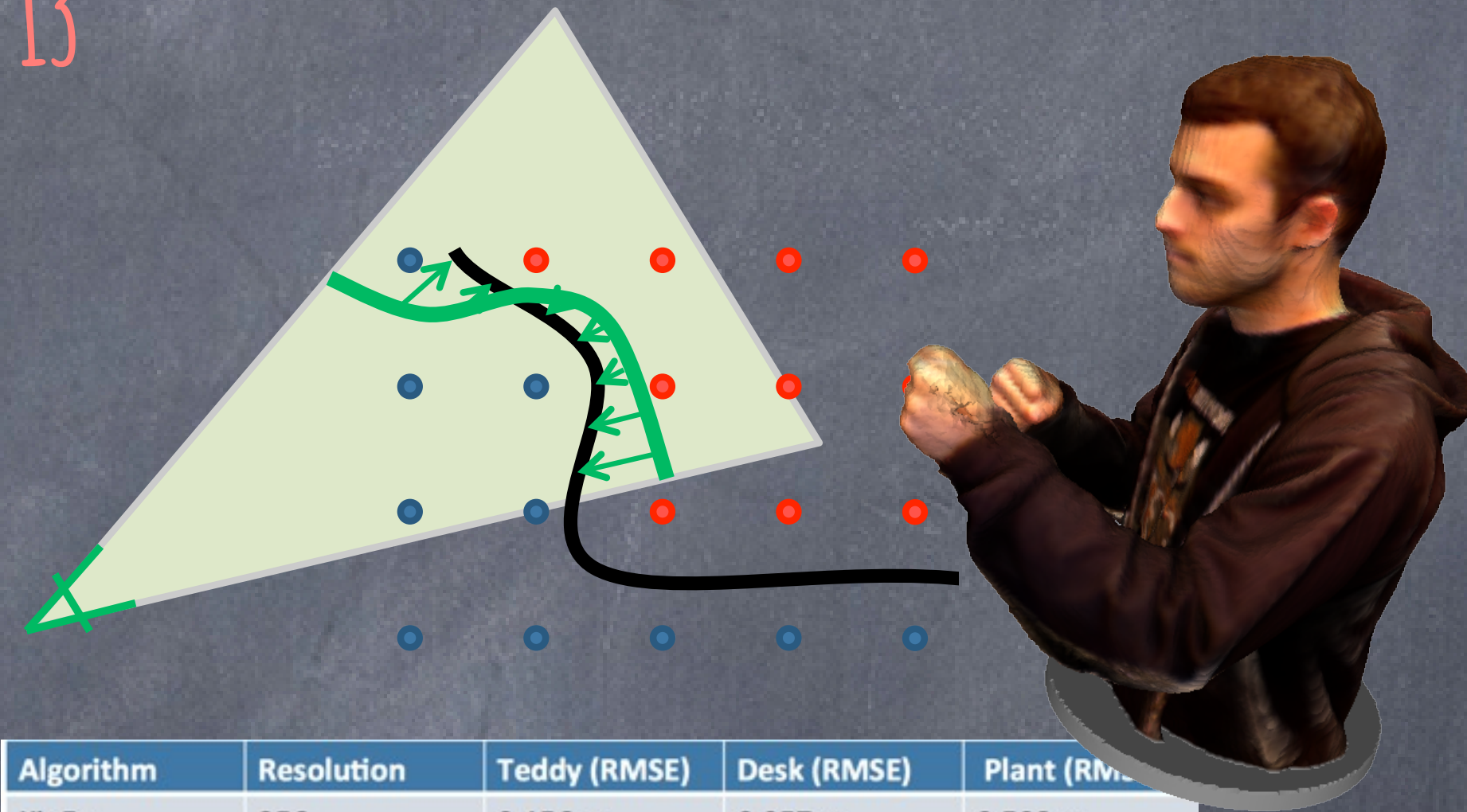
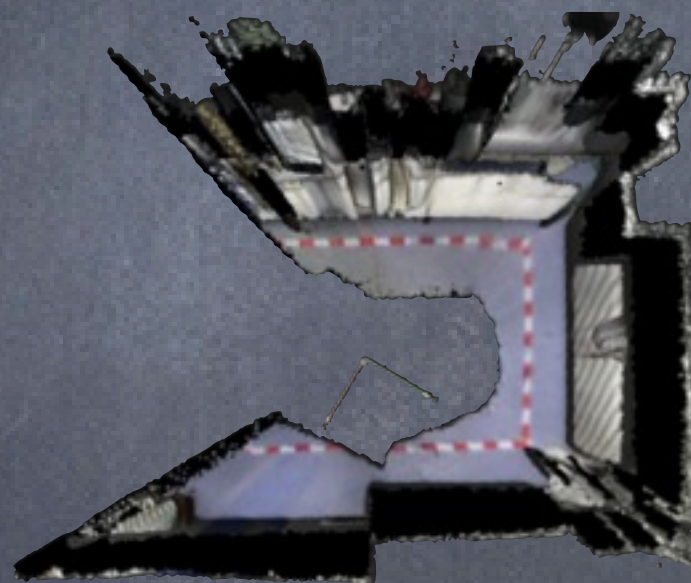
Precision: Volumetric Registration

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!

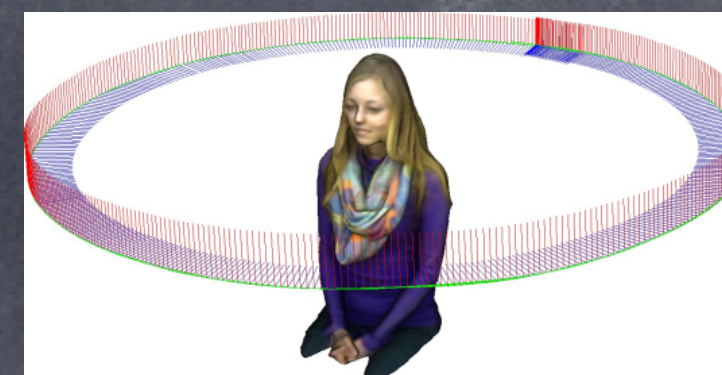
$$T = \arg \min_T \sum_p f(Tp)^2$$



Algorithm	Resolution	Teddy (RMSE)	Desk (RMSE)	Plant (RMSE)
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

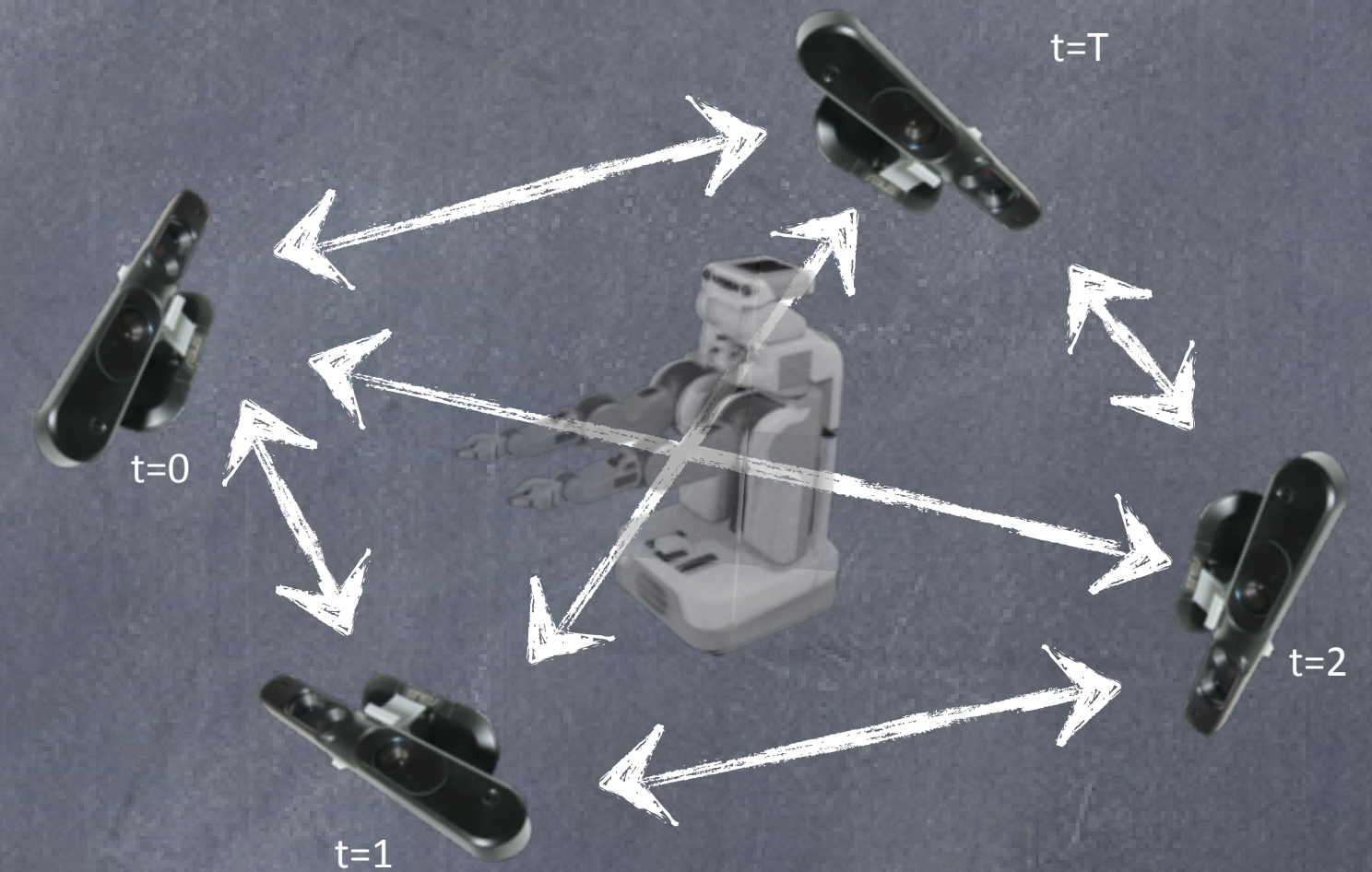


GOOD ENOUGH FOR WATERTIGHT MODELS

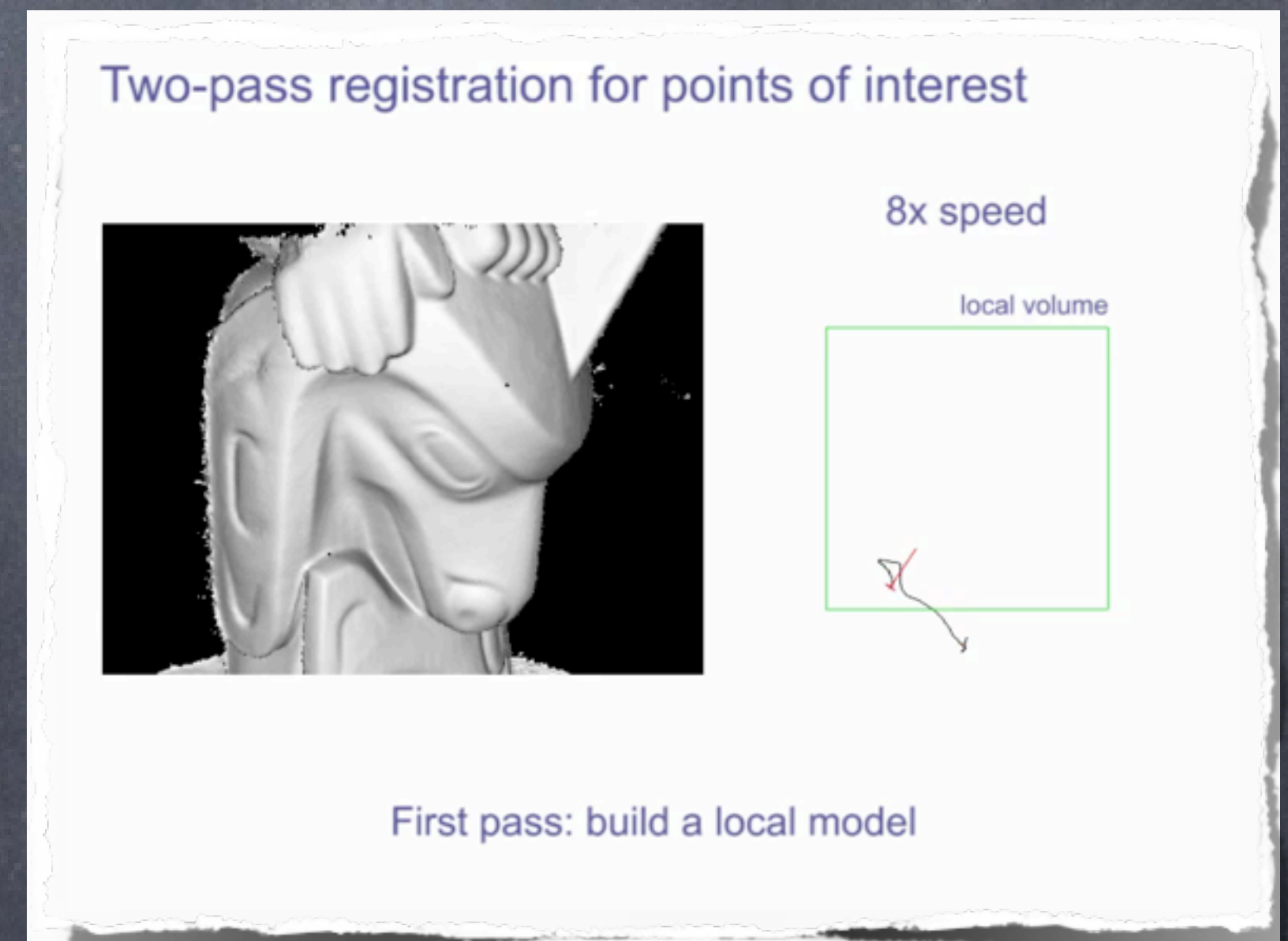
Drift: Adding SLAM (Offline)

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



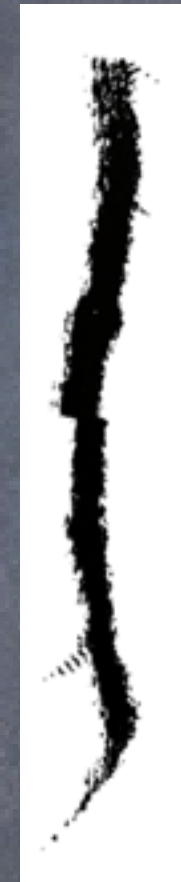
Drift: Elastic Fragments (Offline)

ZHOU ET AL, ICCV '13

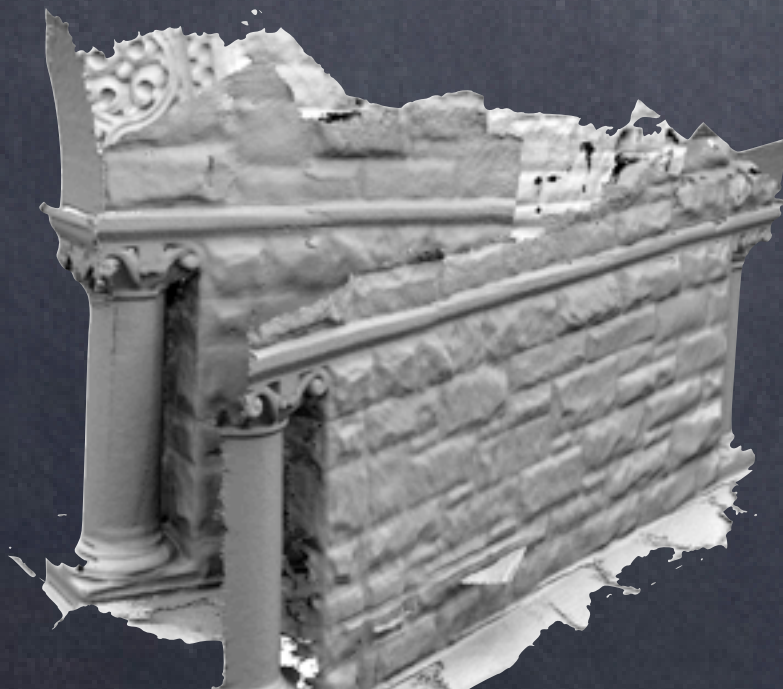
• Idea: Take that last idea, but let's bend the submeshes into shape

• Motivation: sensor distortion

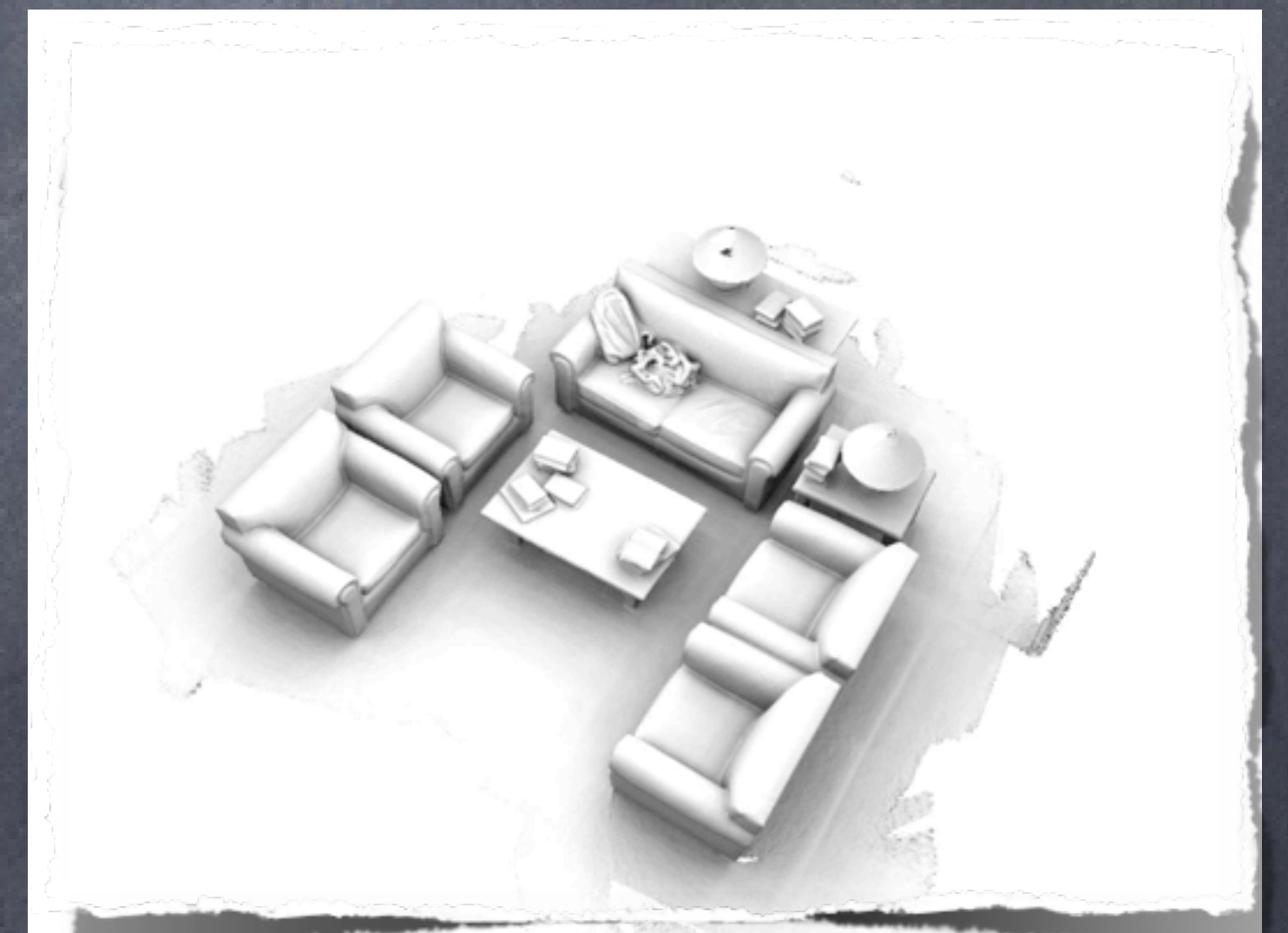
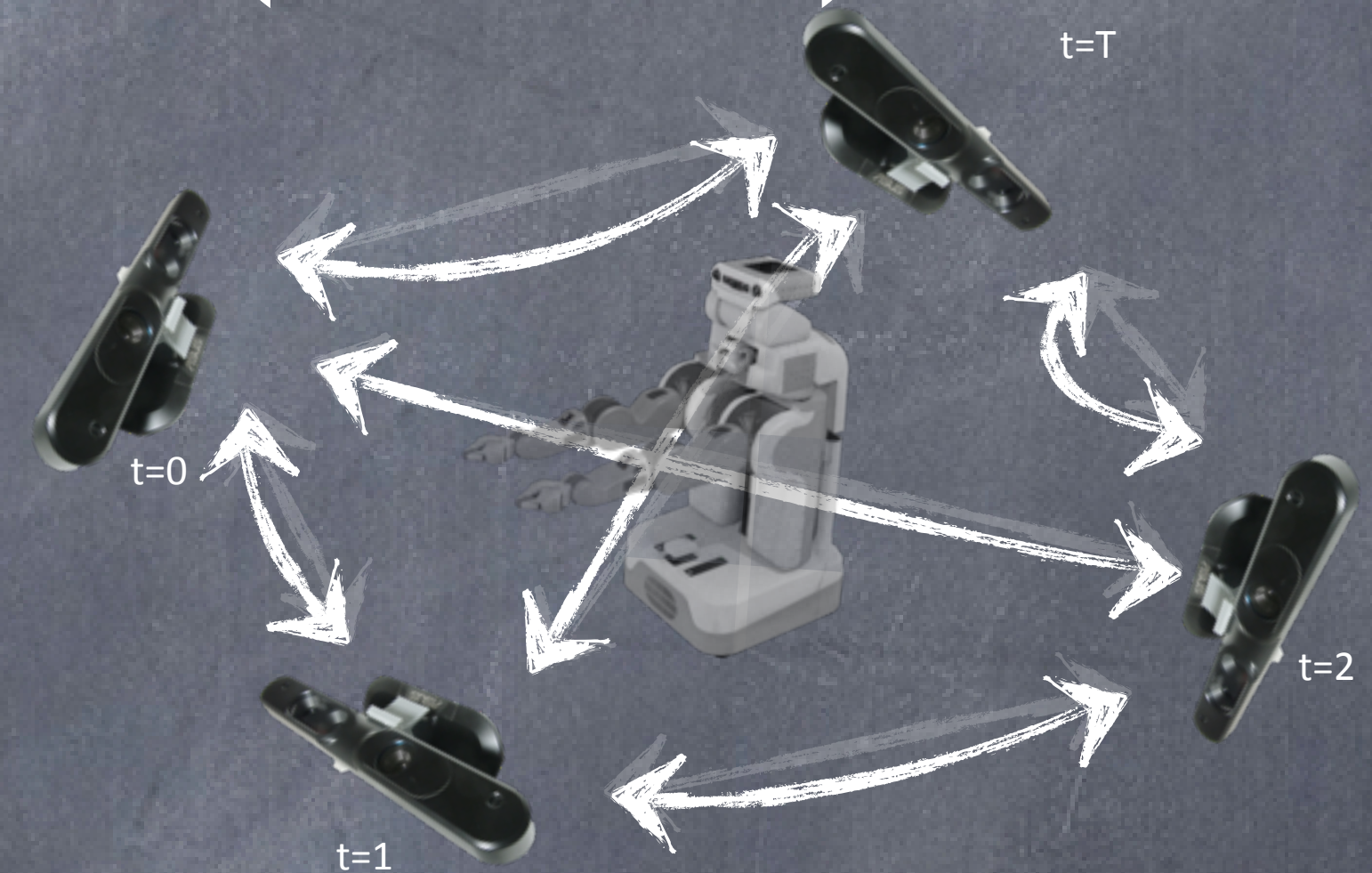
• "Rigid Registration"
→ "As Rigid As Possible"
(Igarashi et. al, TOG 2005)



Kinect Fusion



ours

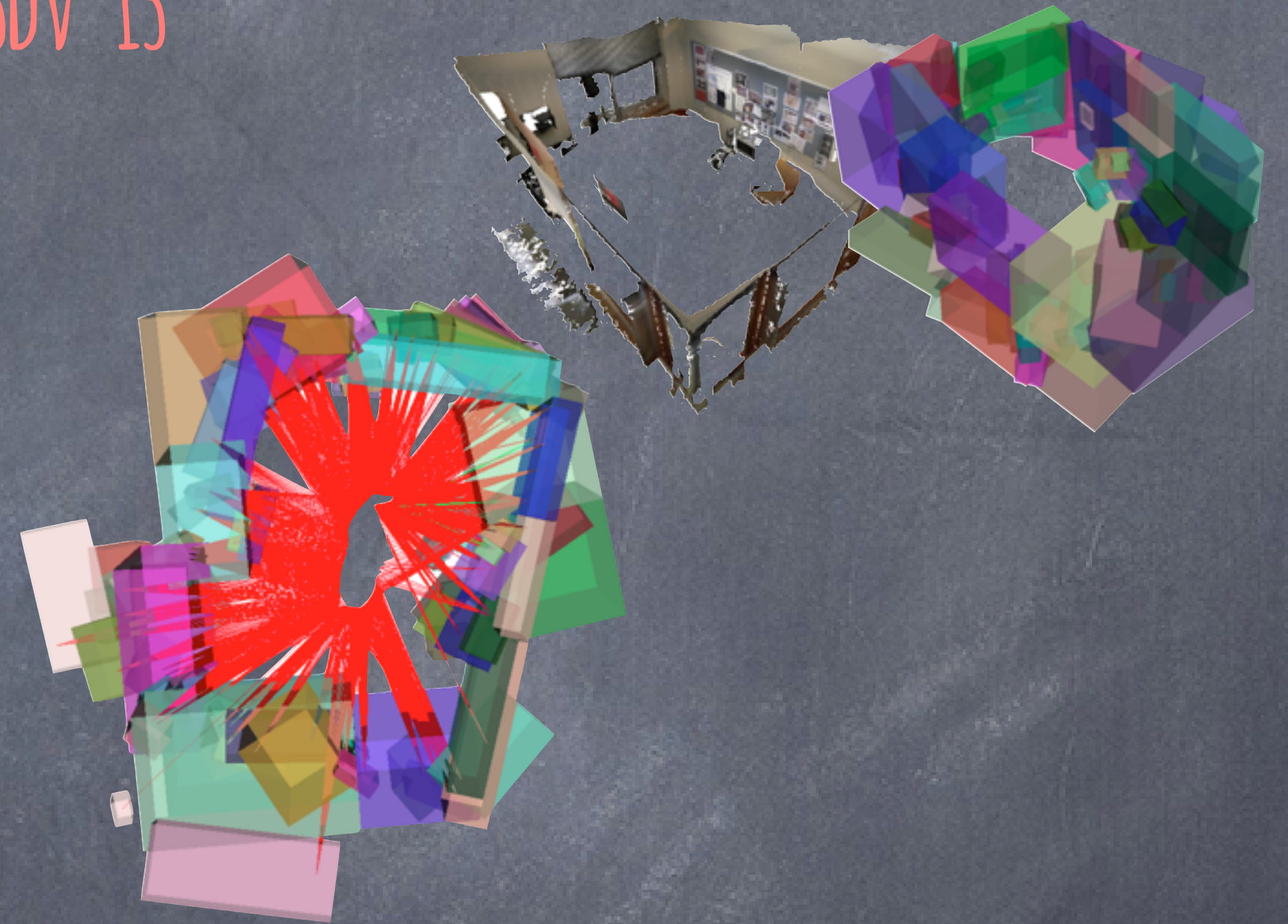


Drift: Patch Volumes

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

Deformation-based Loop Closure for Large Scale Dense RGB-D SLAM

Thomas Whelan, John McDonald
Department of Computer Science, NUI Maynooth

Michael Kaess, John J. Leonard,
Computer Science and Artificial Intelligence Laboratory (CSAIL),
Massachusetts Institute of Technology (MIT)



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!