Dense Indoor 3D Reconstruction

KinectFusion and DTAM
(and their variations)

Wei Dong
2021/04/07

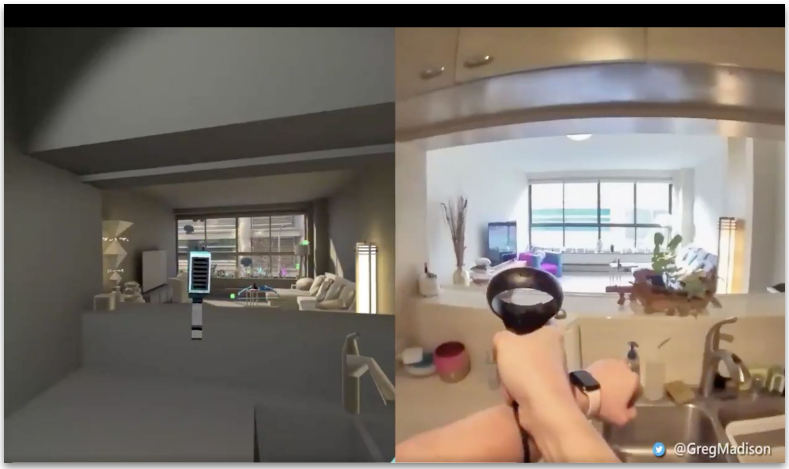
Outline

- **Intro**
- Mapping: point-based fusion
- Mapping: implicit function and SDF
- Tracking: RGB-D odometry
- Depth estimation: from TV to deep learning

Motivation

- Realistic, immersive Mixed Reality (MR)
- Easier content generation for Virtual Reality (VR)
- We need dense mapping!

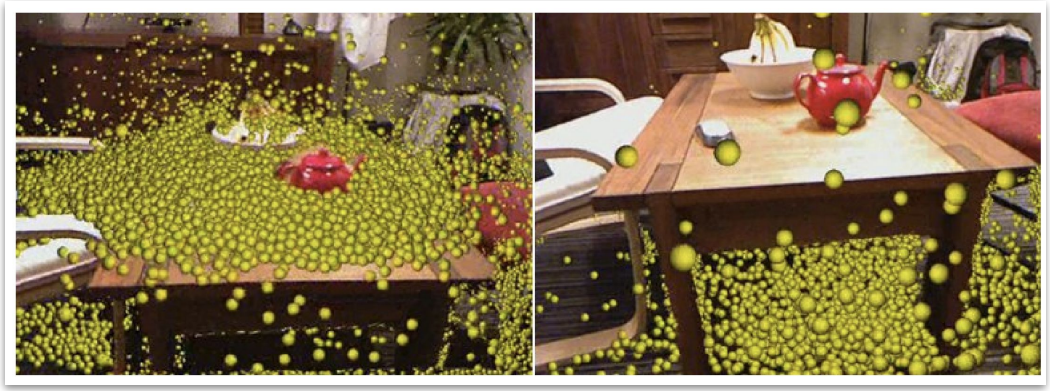
Mixed Reality: *Microsoft Hololens*



Virtual Reality: *Oculus* + *Unity 3D*
Credit: twitter @GregMadison

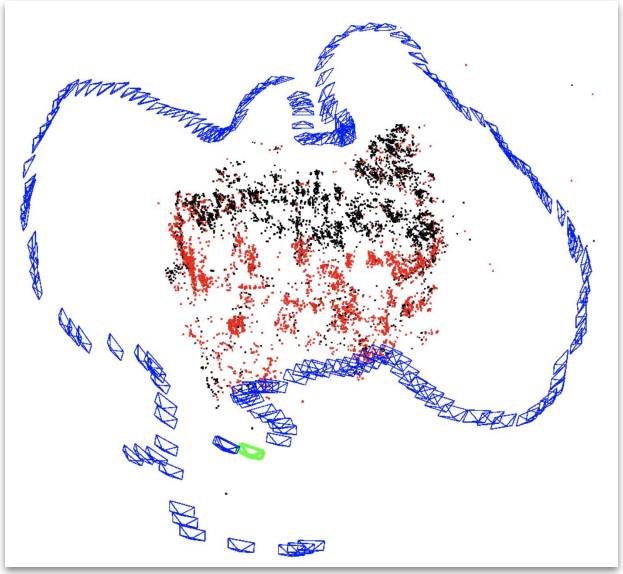
Dense mapping: accurate 3D geometry

More correct occlusions, less artifacts
Dense Planar SLAM, ISMAR 2014

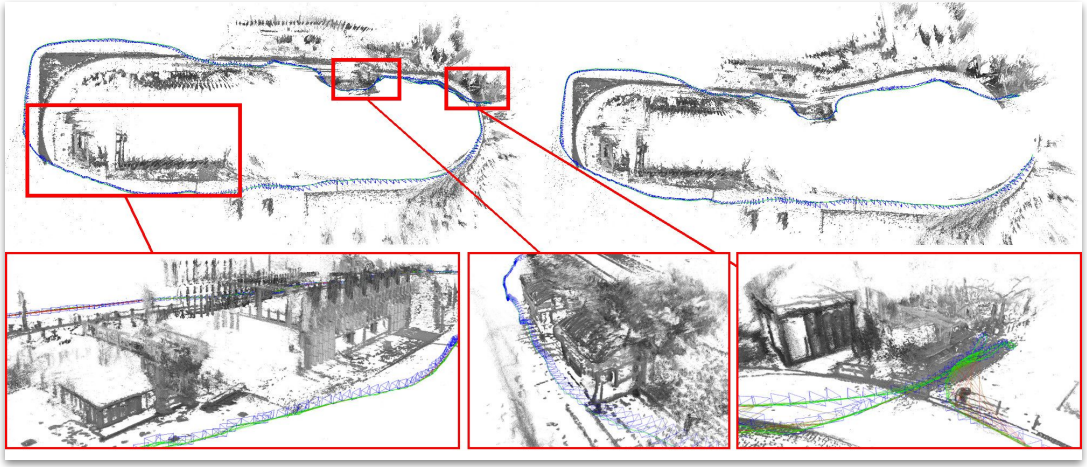


Reasonable physically plausible interactions
KinectFusion, ISMAR 2011

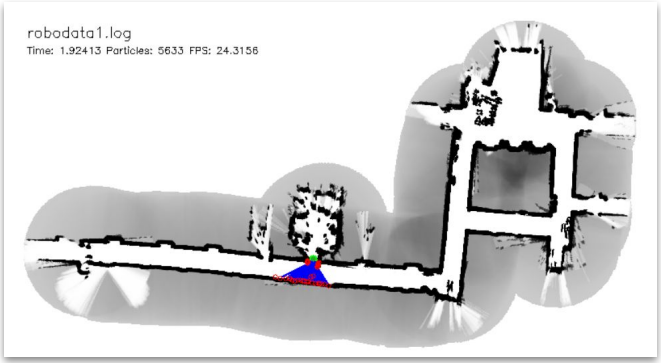
Maps we know



Sparse feature points



Semi-dense points

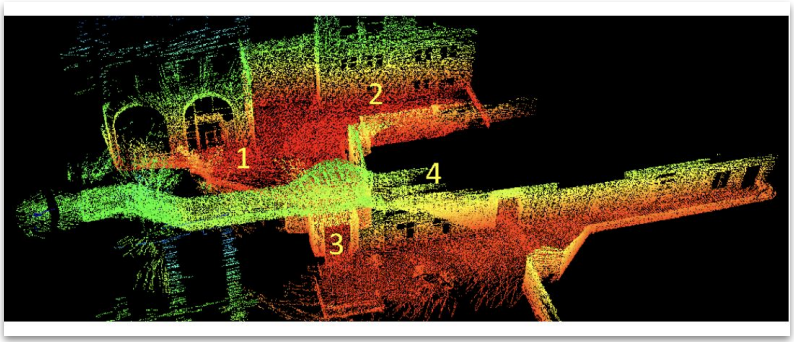


Dense 2D occupancy grids

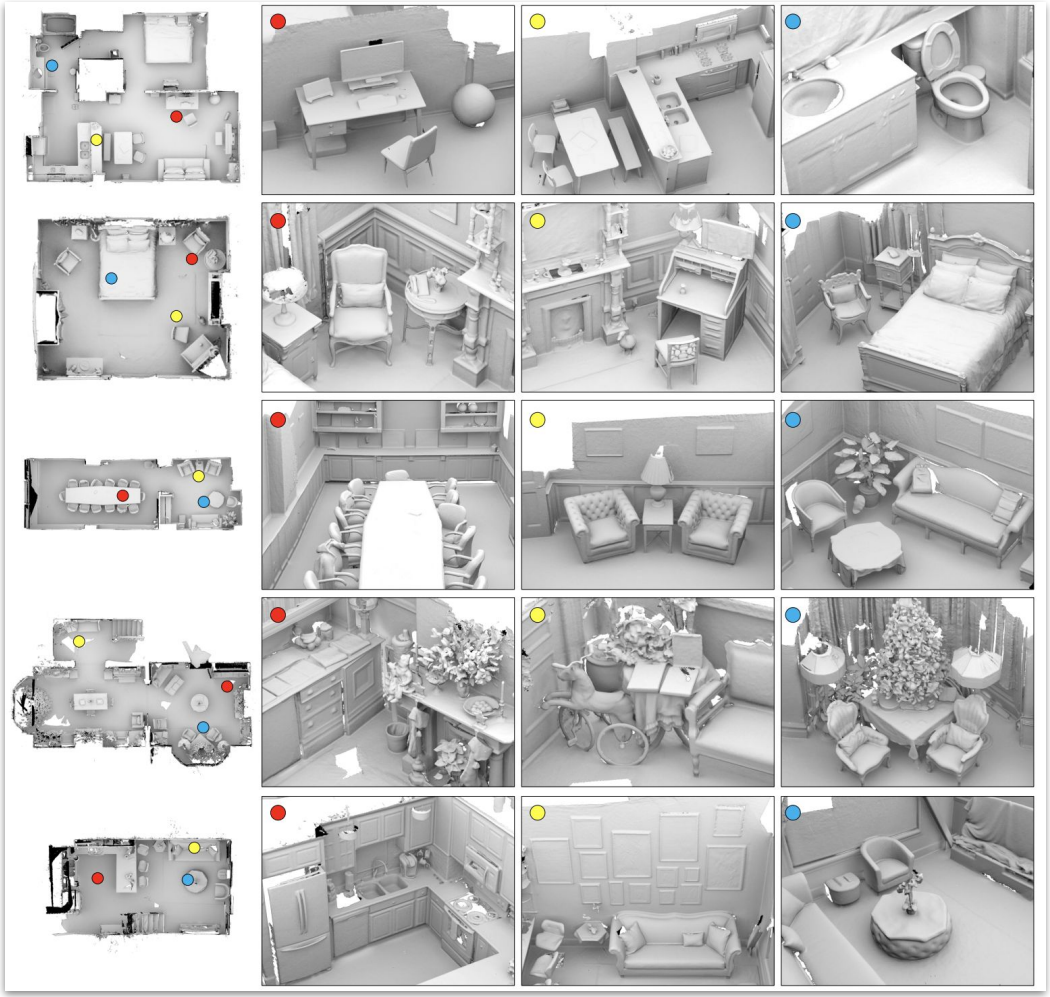
- None would work…

Maps we want

- 3D
- Preserve **surfaces**
- Can be **rendered** ‘realistically’
  - Your grandma can recognize



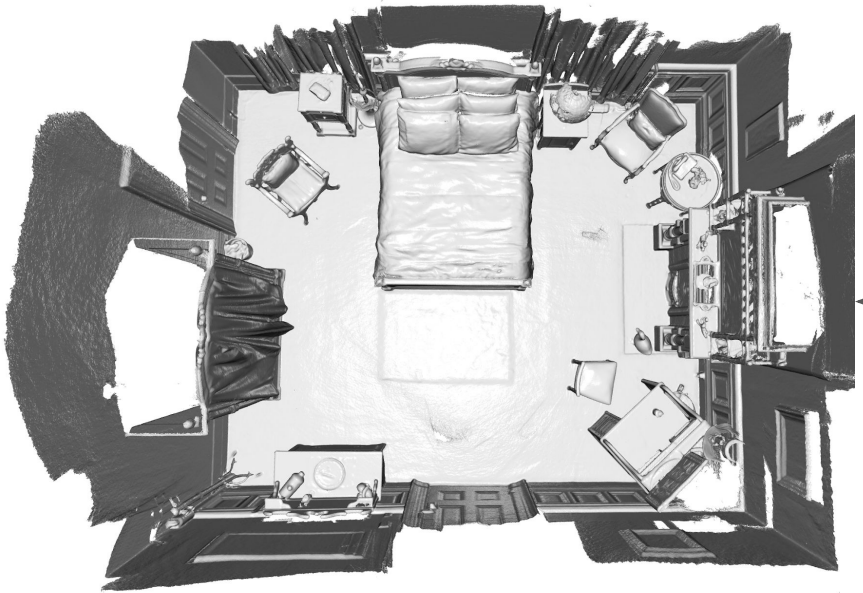
VLOAM, ICRA 2015
(Is it good enough for dense mapping?)



Colored Point Cloud Registration Revisited, ICCV 2017

Recap

What we want



Process

What we have

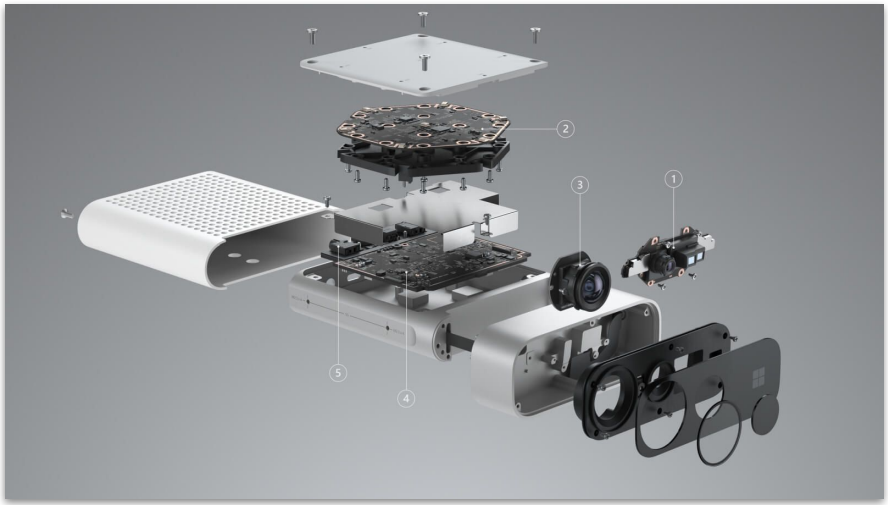
Densely reconstructed scenes for
MR: building Minecraft castles on your bed,
or VR: inviting our friends to play in the cyber bedroom while keeping social distance

LASER: > $10K

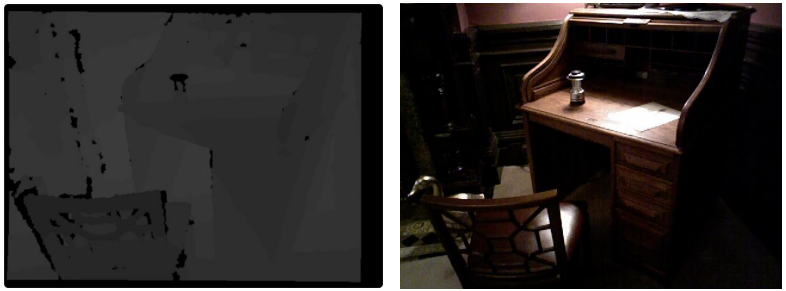
RGB-D: < $0.5K

RGB-D cameras and RGB-D data

- Conventional camera + ToF/structure light
- RGB video + depth video
    - Equivalent to point clouds under a projection



Microsoft Azure Kinect

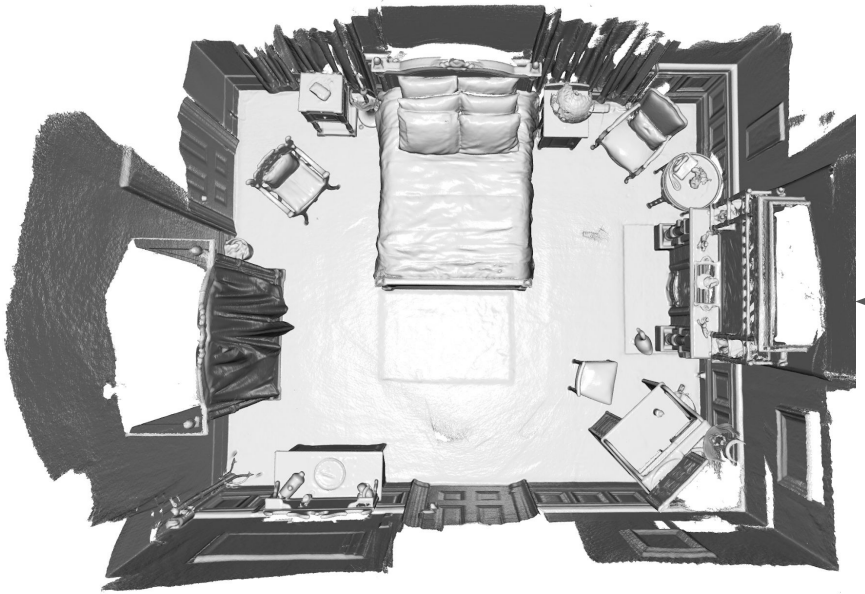


Raw data

Corresponding point cloud

Mapping: raw data to complete reconstruction

What we want: a clean, complete model



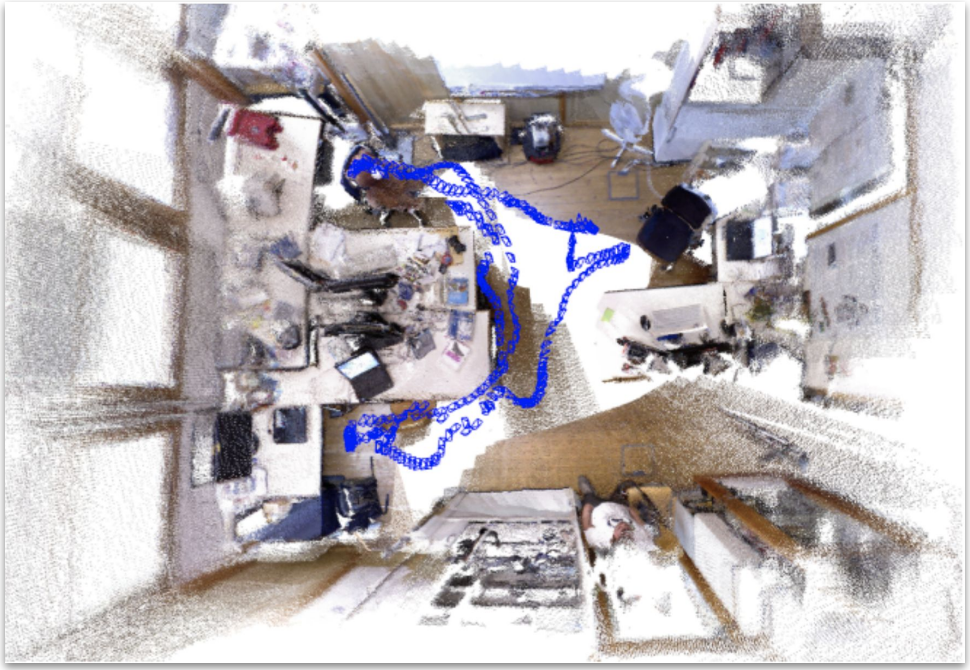
What we have: noisy, partial scans*

x 20K

*Assume poses are known at this stage!

Brute force

- Stack all the points and postprocess
- Application of ORB-SLAM2
  - First predict poses with the system
    - Only maintain feature maps
  - Then accumulate points
- Problems
  - 300K pts x 10K frames
  - Almost impossible to postprocess...



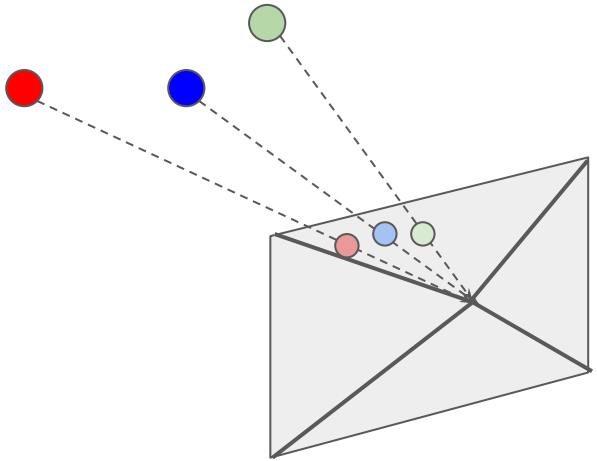
ORB SLAM2, TOR 2017

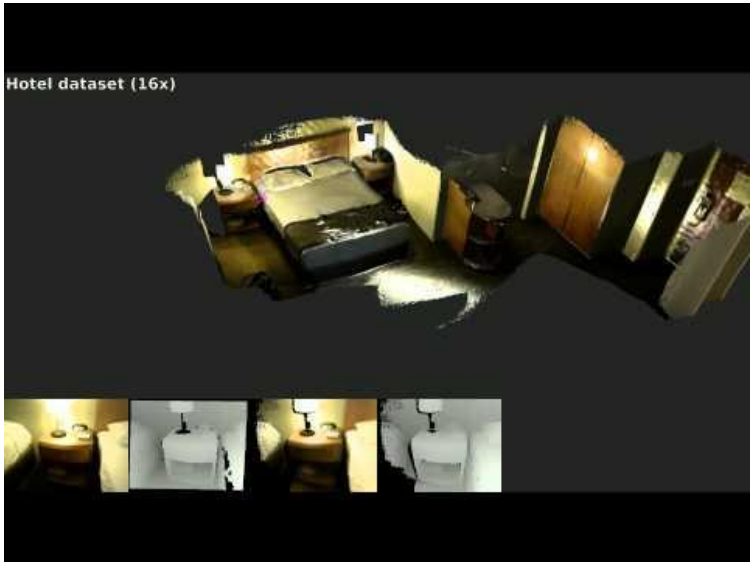
Outline

- Intro
- **Mapping: point-based fusion**
- Mapping: implicit function and SDF
- Tracking: RGB-D odometry
- Depth estimation: from TV to deep learning

Point-based fusion

- Process RGB-D frame + poses one by one
- Merge close points during accumulation
  - Project existing points to current image

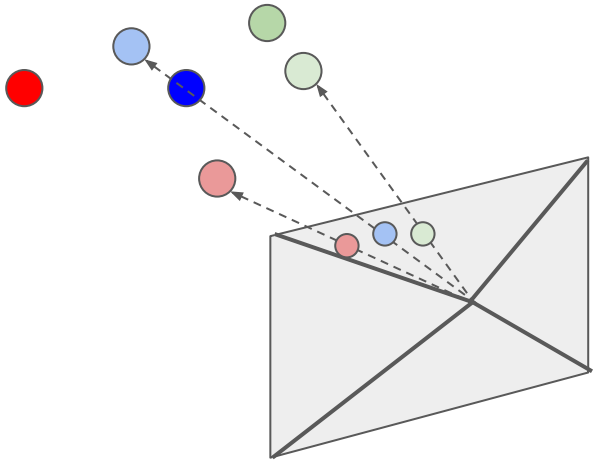


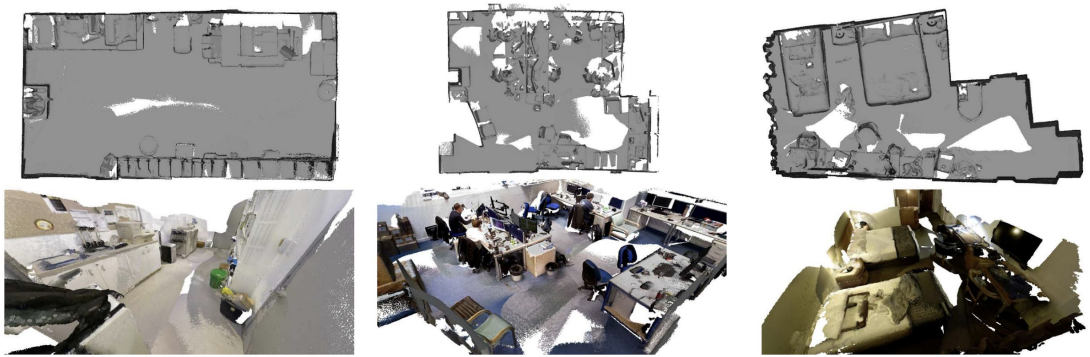


ElasticFusion, RSS 2015

Point-based fusion

- Merge close points during accumulation
  - Unproject matched RGB-D observations to 3D

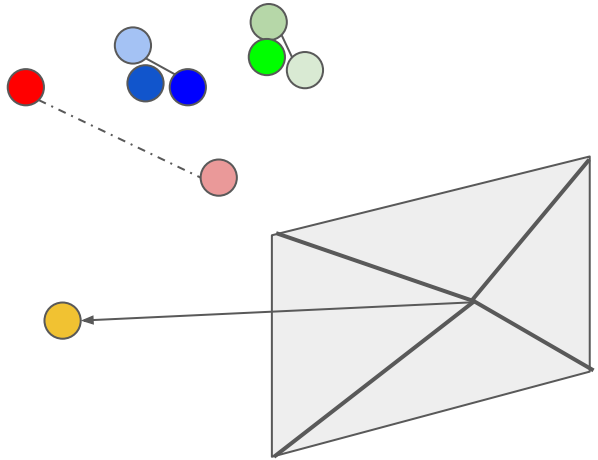


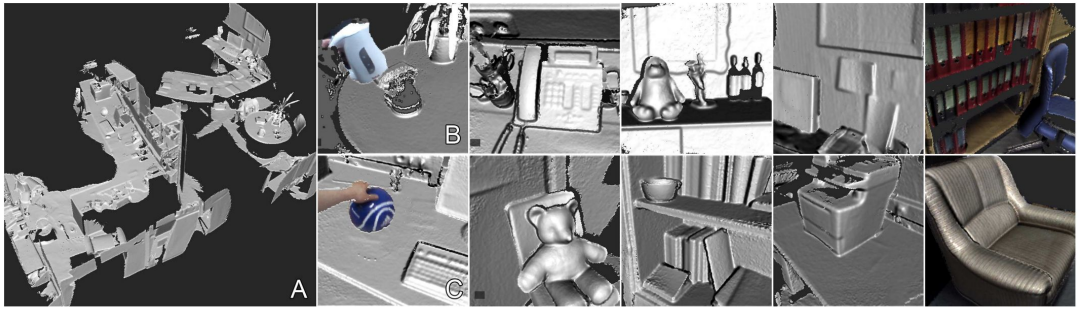


ElasticFusion, RSS 2015

Point-based fusion

- Merge close points during accumulation
  - Threshold and weight average **positions**
    - Threshold: remove outliers
  - Add other unmatched observations





Real-time 3D Reconstruction in Dynamic Scenes using Point-based Fusion, 3DV 2013

Point-based fusion: pros and cons

- Pros
  - Light weight: manipulating points is easy
  - Memory-friendly: map as array of points
  - BA-friendly: can integrate point-based optimizations bundle adjustment
    - *BAD SLAM, CVPR 2019*
- Cons
  - Outputs points are not surfaces (but can be converted via e.g. Poisson Reconstruction)
  - Position-based fusion is still (kind of) sensitive to outliers and noise

Outline

- Intro
- Mapping: point-based fusion
- **Mapping: implicit function and SDF**
- Tracking: RGB-D odometry
- Depth estimation: from TV to deep learning

Another perspective towards surfaces

- Surface as the zero crossing set of a continuous function

$$f(x, y) = x^2 + y^2 - 1$$

- Does not **explicitly** know surfaces
  - Tell me the surface vertices: I can't...
- Can **implicitly** decide surfaces by evaluating positions
  - Tell me if (2, 4) is on surface: No
  - Tell me if (0, 1) is on surface: Yes
  - A query function/machine

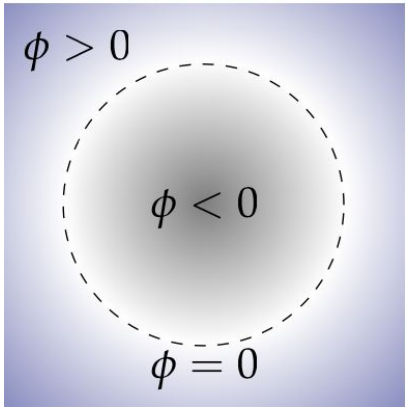


Image courtesy:
CMU 15858

Implicit to explicit

- Discretize, evaluate, and interpolate
- 3D: Marching Cubes

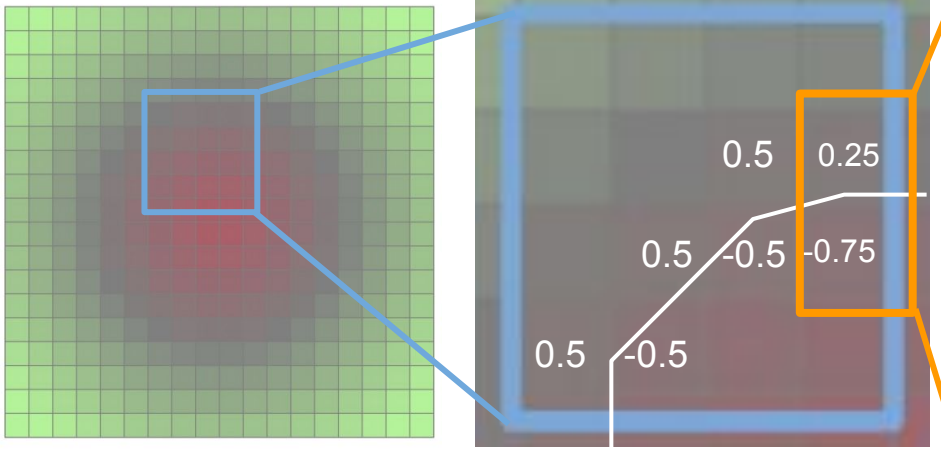
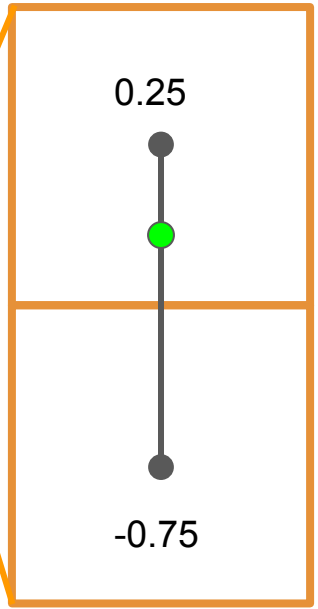
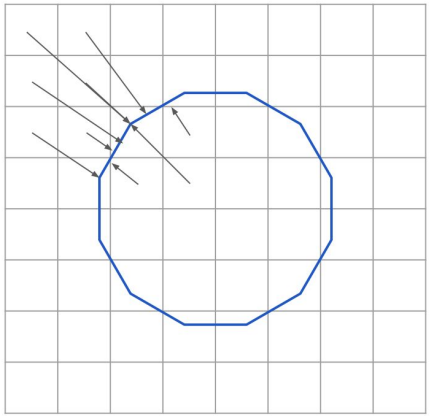


Image courtesy: UW CSE571

Local surface



Vertex extraction

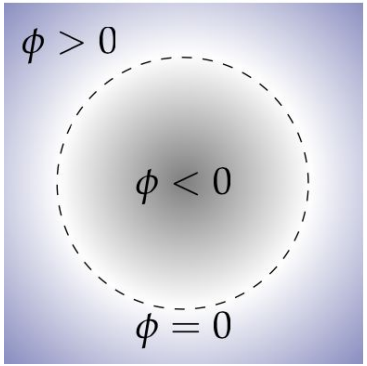


Overall surface

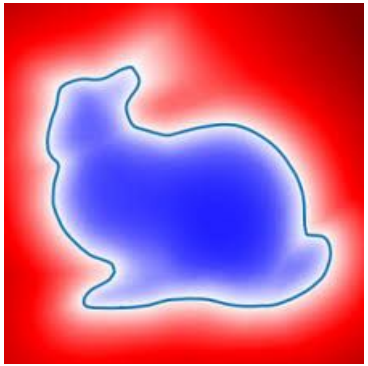
$$f(x, y) = x^2 + y^2 - 1$$

Real-world surface representation

- A circle has an analytical form
- Real-world surfaces do not …
- But all we need is:
  - An abstract **function definition**
  - An **evaluation method** for 3D positions



$$f(x, y) = x^2 + y^2 - 1$$



?

Function definition

- Signed distance function (SDF)
- **Oriented distance to surface**
- Query a 3D position, return a distance

$$\mathbb{R}^3 \rightarrow \mathbb{R}$$

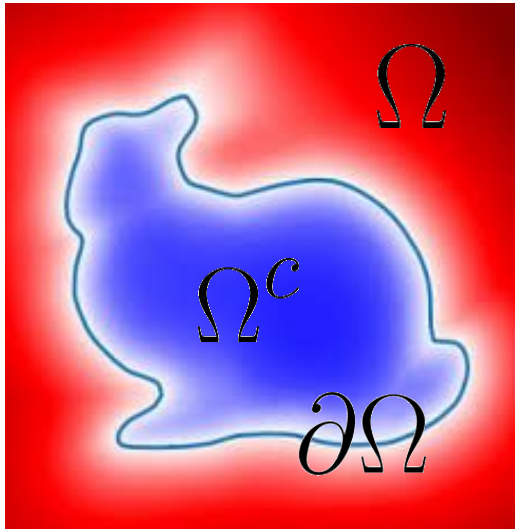
If Ω is a subset of a metric space, X, with metric, d, then the *signed distance function*, f, is defined by

$$f(x) = \begin{cases} d(x, \partial\Omega) & \text{if } x \in \Omega \\ -d(x, \partial\Omega) & \text{if } x \in \Omega^c \end{cases}$$

where $\partial\Omega$ denotes the boundary of Ω. For any $x \in X$,

$$d(x, \partial\Omega) := \inf_{y \in \partial\Omega} d(x, y)$$

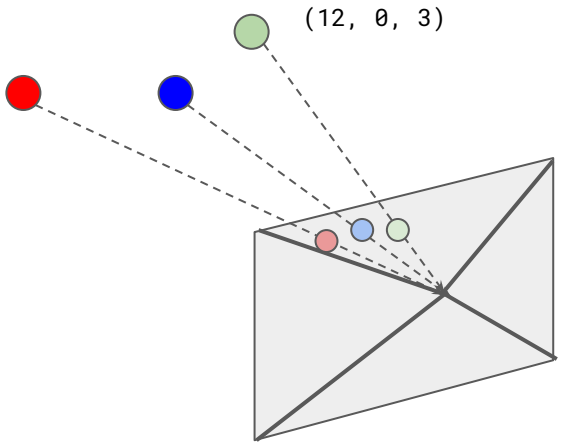
Wikipedia

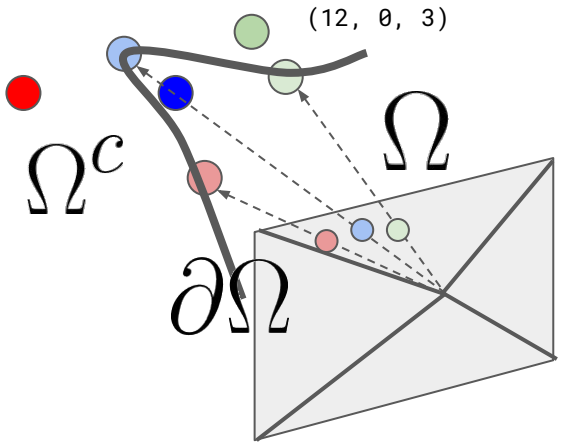


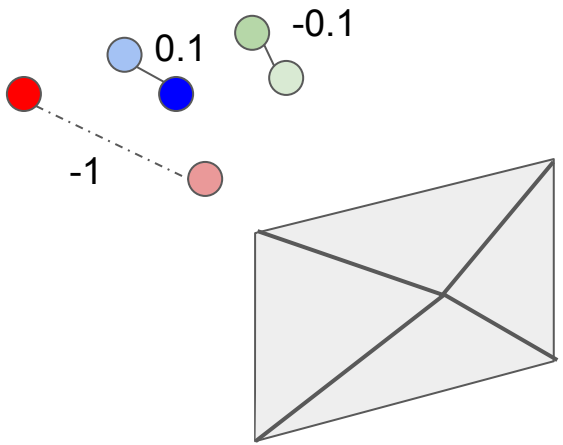
Evaluation method: single RGB-D input

- Tell me the distance value at (12, 0, 3)
  - Project query point to current image
  - Unproject matched RGB-D observations to 3D
  - Truncate and compute **distance**

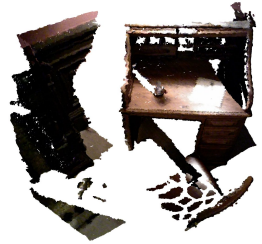
Intuition:
- RGB-D data approximate surface
- Distance from a **query point to its closest observation** approximates distance from the **query point to the real surface**



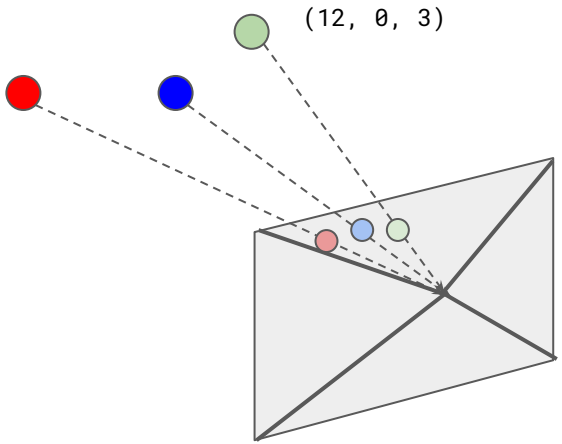


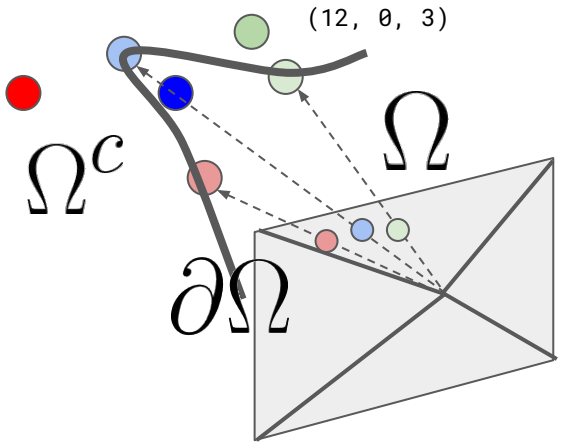


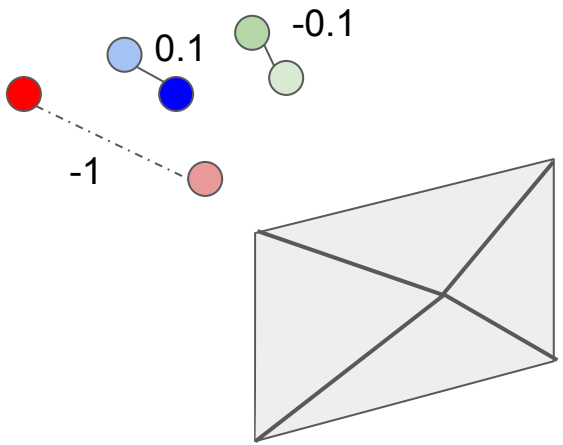
Evaluation method: multiple frames



- Tell me the distance value at (12, 0, 3) given all 20K frames
  - Project query point to **every** image
  - Unproject **every** matched RGB-D observations to 3D
  - Truncate and compute **mean distance (average)**
    - Truncate: remove outliers

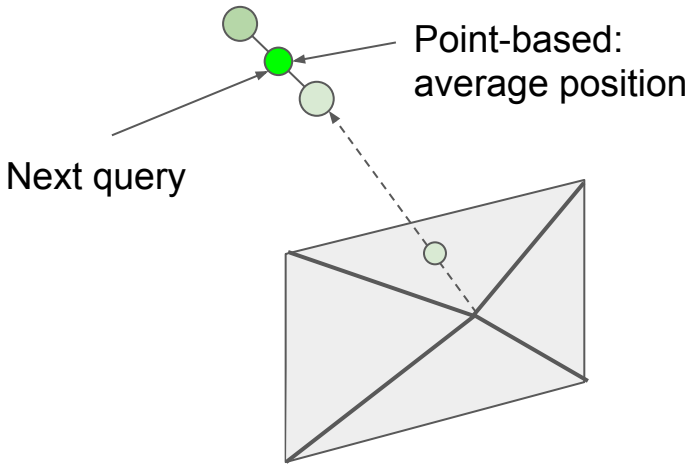


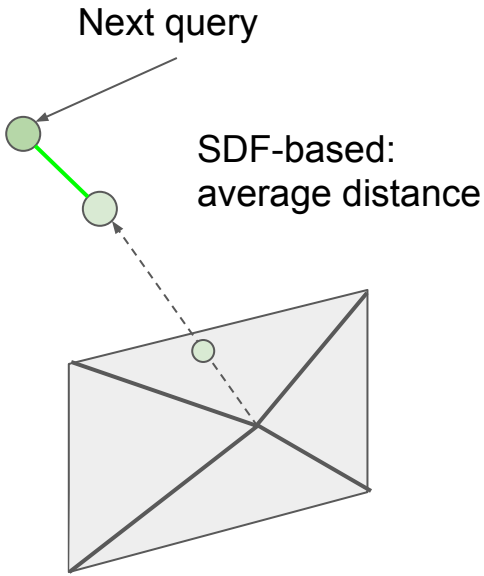




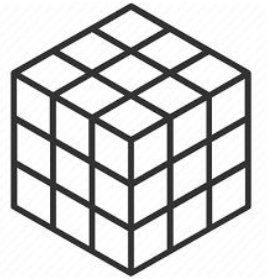
Difference to point-based fusion

- Point-based
  - Points are not fixed; average positions
- SDF-based
  - Query points are fixed; average all distances

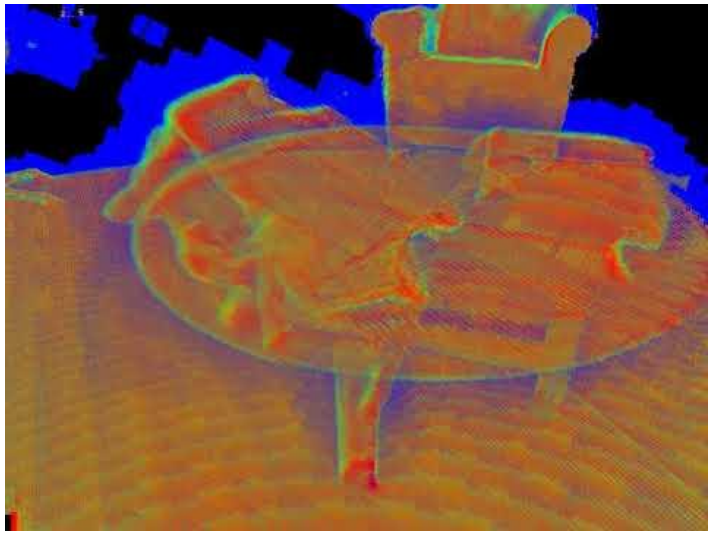




KinectFusion: SDF integration



- Discretize the space to a 512^3 volume; each grid is a query point
- For each RGB-D frame
  - For each query point in 512^3 query points
    - Compute weight average of SDF value

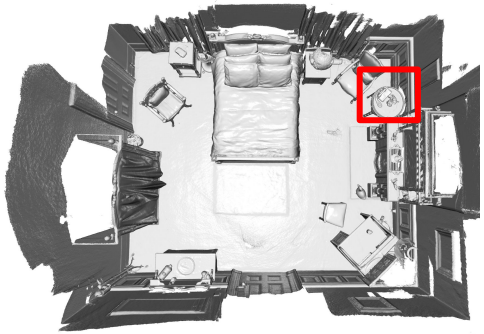


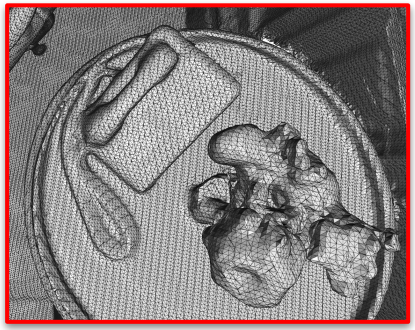
$$
\begin{aligned}
\mathrm{F}_{\mathrm{R}_k}(\mathbf{p}) &= \Psi\left(\lambda^{-1}\|(\mathbf{t}_{g,k}-\mathbf{p}\|_2-\mathrm{R}_k(\mathbf{x})\right), && \text{Distance}\\
\lambda &= \|\mathrm{K}^{-1}\dot{\mathbf{x}}\|_2, && \text{Unprojection}\\
\mathbf{x} &= \left\lfloor \pi\left(\mathrm{K}\mathrm{T}_{g,k}^{-1}\mathbf{p}\right)\right\rfloor, && \text{Projection}\\
\Psi(\eta) &= \begin{cases} \min\left(1,\frac{\eta}{\mu}\right)\mathrm{sgn}(\eta) & \text{iff } \eta \geq -\mu \\ \textit{null} & \textit{otherwise} \end{cases}
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{F}_k(\mathbf{p}) &= \frac{\mathrm{W}_{k-1}(\mathbf{p})\mathrm{F}_{k-1}(\mathbf{p})+\mathrm{W}_{\mathrm{R}_k}(\mathbf{p})\mathrm{F}_{\mathrm{R}_k}(\mathbf{p})}{\mathrm{W}_{k-1}(\mathbf{p})+\mathrm{W}_{\mathrm{R}_k}(\mathbf{p})}\\
\mathrm{W}_k(\mathbf{p}) &= \mathrm{W}_{k-1}(\mathbf{p})+\mathrm{W}_{\mathrm{R}_k}(\mathbf{p})
\end{aligned}
$$

KinectFusion: pros and cons

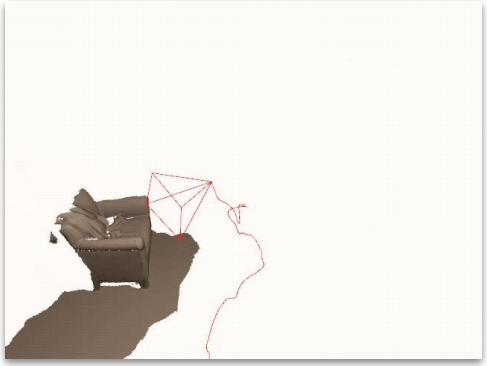
- Pros
  - Produces smooth yet detailed surfaces
  - Usually robust to noise
  - Directly generates triangle mesh!
- Cons
  - Limited spatial resolution: fixed resolution grid
  - Heavy computation: iterate over all query points. GPU!
  - Not friendly to global optimization in SLAM: how to update?

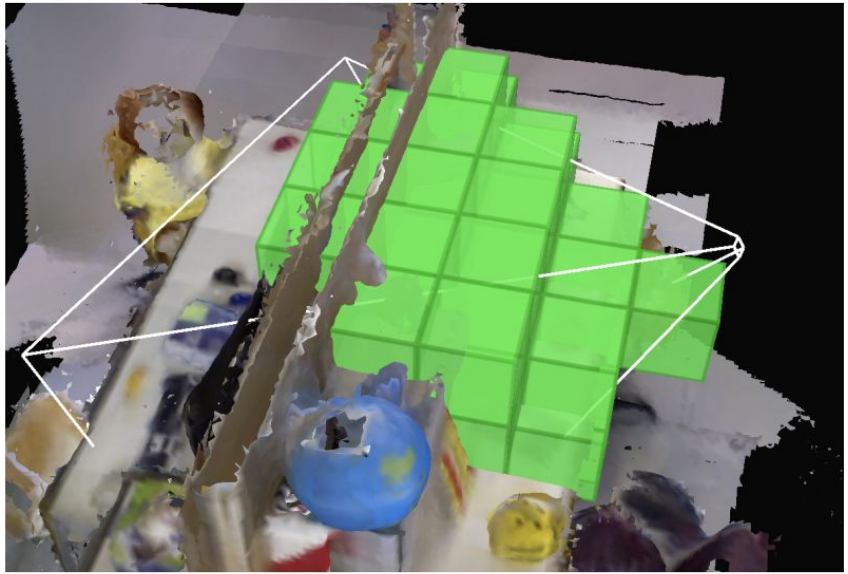




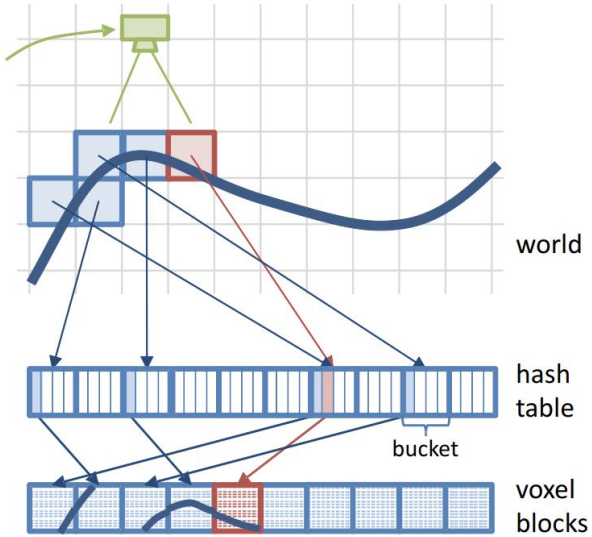
Extension to KinectFusion

- Allocate space on demand using spatial hashing
- Advanced discretization





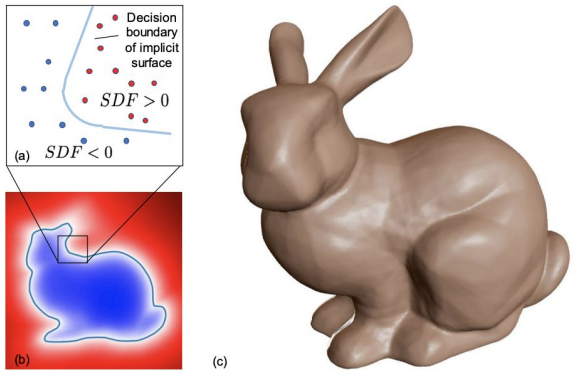
CHISEL, RSS 2015



VoxelHashing, TOG 2013

Modern implicit representations

- **Discrete** resolution has a limit
  - Practically, 4mm voxel size / query point resolution is the upper bound
- Can we really restore the underlying **continuous** function?
  - Representation power of neural networks
  - Learn $\mathbb{R}^3 \rightarrow \mathbb{R}$ functions for scenes
- Intuition
  - Surfaces are natural decision boundaries
  - Networks can fit very complex decision boundaries
  - Surface functions should be learnable!

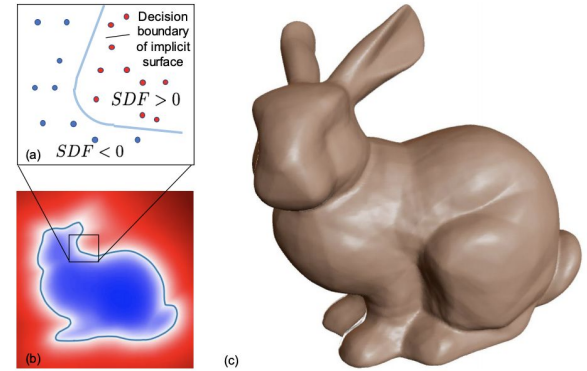


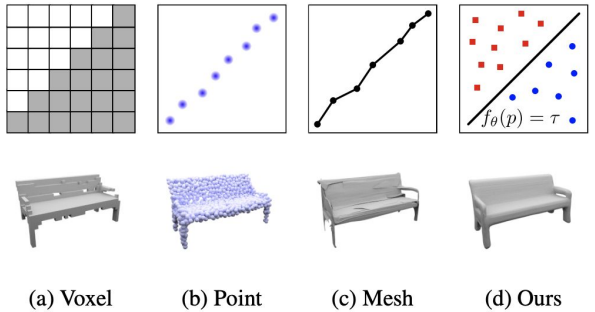
DeepSDF, CVPR 2019

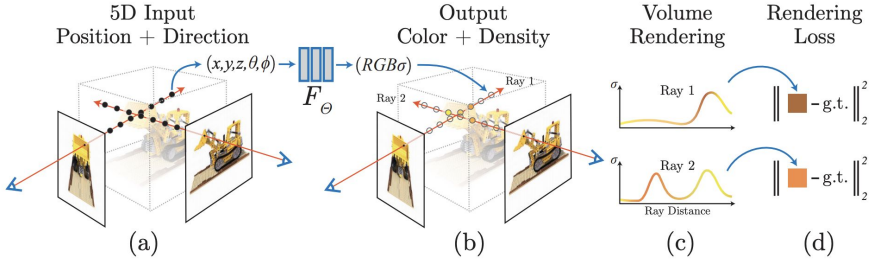
Recent works

- DeepSDF, CVPR 2019
  - 3D supervision
  - 3D positions + shape prior => SDF [-1, 1]
- Occupancy Networks, CVPR 2019
  - 3D supervision
  - 3D positions + shape prior => occupancy [0, 1]
- NeRF, ECCV 2020
  - **2D supervision**
  - 3D positions + 2D viewing angles =>

    3D color + 1D volume weight







Summary

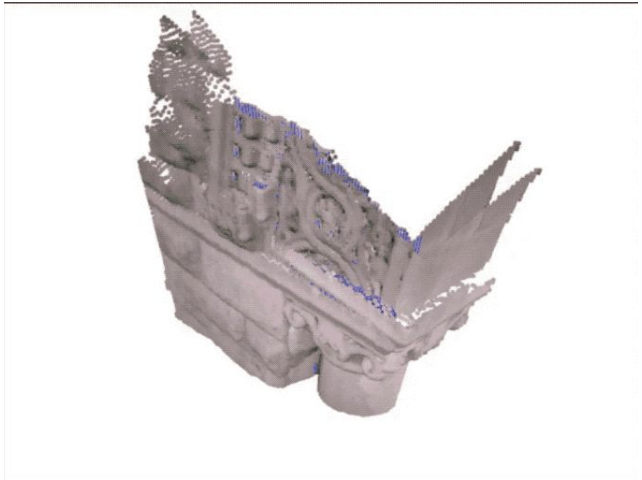
- Dense 3D mapping
  - noisy, partial data to clean, complete models
- Point-based fusion
  - Engineering perspective
    - Data, data structure, implementation
- SDF fusion
  - Mathematical perspective
    - Math representation, digitalization (discretization), implementation

Outline

- Intro
- Mapping: point-based fusion
- Mapping: implicit function and SDF
- **Tracking: RGB-D odometry**
- Depth estimation: from TV to deep learning

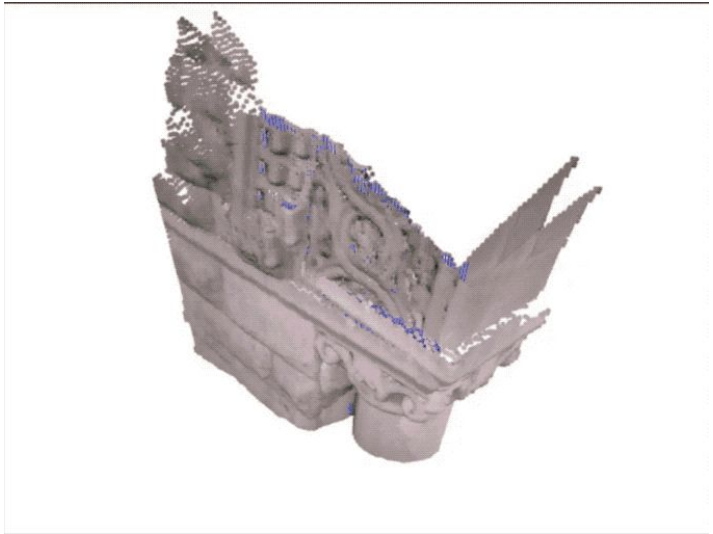
Tracking: raw data to poses

- We assume poses are known in integration
- We may use ORB-SLAM to acquire poses and perform integration
  - Feature-based mapping and tracking, another system...
- But we prefer the system to be self-contained!
  - Directly align RGB-D images/point clouds

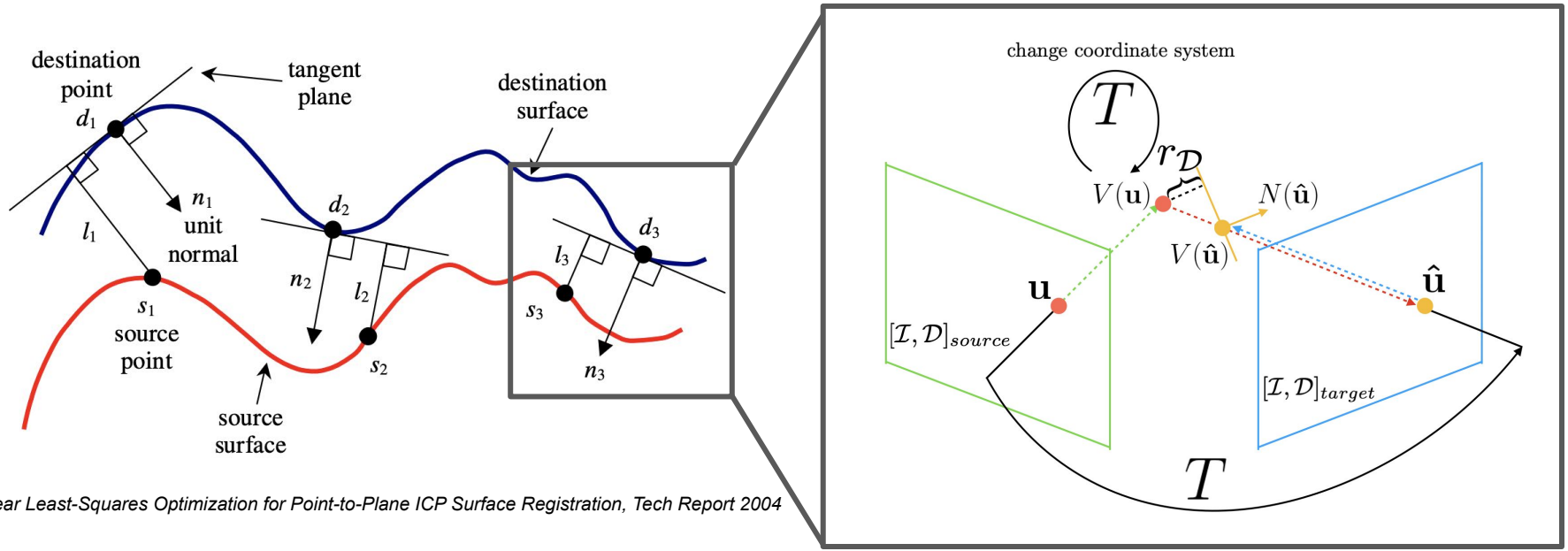


RGB-D odometry

- Align two RGB-D frames
- Also known as **registration/alignment**
- Intuition
  - **Overlap area** should be similar after the transformation
  - Geometry and appearance should look similar
- Formulation
  - $\min_{T} f(I_{src}, I_{tgt}, T)$



Viewpoints of odometry/registration



Linear Least-Squares Optimization for Point-to-Plane ICP Surface Registration, Tech Report 2004

General view

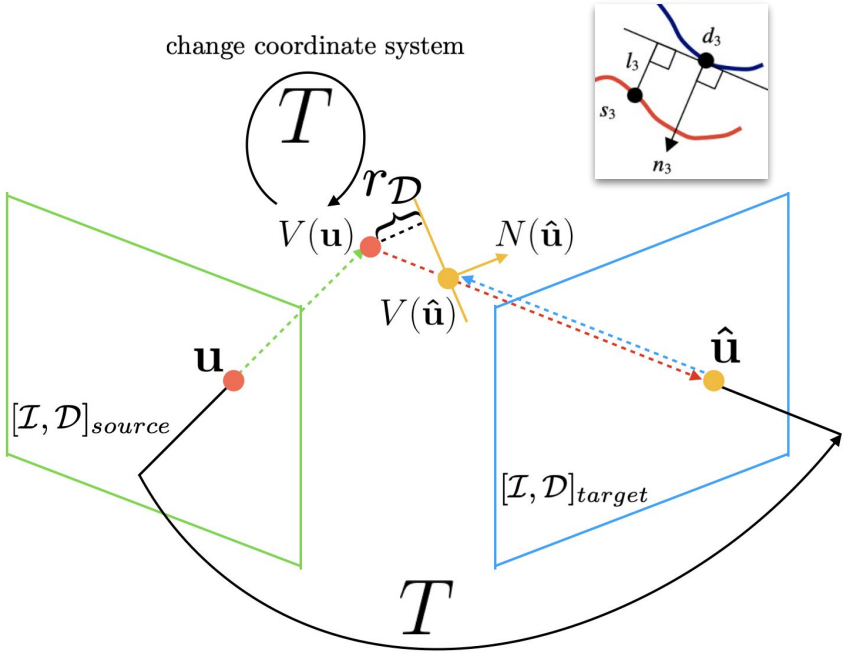
- Match shapes by minimizing point-to-(local)plane distance upon transformation

Sum

Pointwise view

- Find correspondence and compute distance upon transformation

Depth odometry: formulation



$$\mathbf{E}(\mathbf{T}_{g,k}) = \sum_{\substack{\mathbf{u}\in\mathcal{U} \\ \Omega_k(\mathbf{u})\neq\text{null}}} \left\| \left(\mathbf{T}_{g,k} \dot{\mathbf{V}}_k(\mathbf{u}) - \hat{\mathbf{V}}^g_{k-1}(\hat{\mathbf{u}}) \right)^\top \hat{\mathbf{N}}^g_{k-1}(\hat{\mathbf{u}}) \right\|_2$$

- Align two depth frames
- Minimize sum of point-to-plane distance
  - Unproject source points **u** to 3D **V(u)**
  - Transform **V(u)** with current **T**
  - Project **TV(u)** to target $\hat{\mathbf{u}}$
  - Sum up error
  - Optimize and update **T**

Depth odometry: optimization

$$\mathbf{E}(\mathrm{T}_{g,k}) = \sum_{\substack{\mathbf{u}\in\mathscr{U} \\ \Omega_k(\mathbf{u})\neq\text{null}}} \left\| \left(\mathrm{T}_{g,k}\dot{\mathbf{V}}_k(\mathbf{u}) - \hat{\mathbf{V}}^g_{k-1}(\hat{\mathbf{u}}) \right)^\top \hat{\mathbf{N}}^g_{k-1}(\hat{\mathbf{u}}) \right\|_2$$

- Parameterize **delta T with x**

$$\widetilde{\mathrm{T}}^z_{inc} = \left[\widetilde{\mathrm{R}}^z \middle| \widetilde{\mathbf{t}}^z \right] = \begin{bmatrix} 1 & \alpha & -\gamma & t_x \\ -\alpha & 1 & \beta & t_y \\ \gamma & -\beta & 1 & t_z \end{bmatrix}$$

$$\mathbf{x} = (\beta, \gamma, \alpha, t_x, t_y, t_z)^\top \in \mathbb{R}^6$$

- Rewrite the equation in terms of **x**

$$\widetilde{\mathrm{T}}^z_{g,k}\dot{\mathbf{V}}_k(\mathbf{u}) = \widetilde{\mathrm{R}}^z\widetilde{\mathbf{V}}^g_k(\mathbf{u}) + \widetilde{\mathbf{t}}^z = \mathbf{G}(\mathbf{u})\mathbf{x} + \widetilde{\mathbf{V}}^g_k(\mathbf{u}) \,,$$

$$\mathbf{G}(\mathbf{u}) = \left[\left[\widetilde{\mathbf{V}}^g_k(\mathbf{u}) \right]_\times \middle| \mathbf{I}_{3\times 3} \right]$$

- Solve the linear system of **x**!

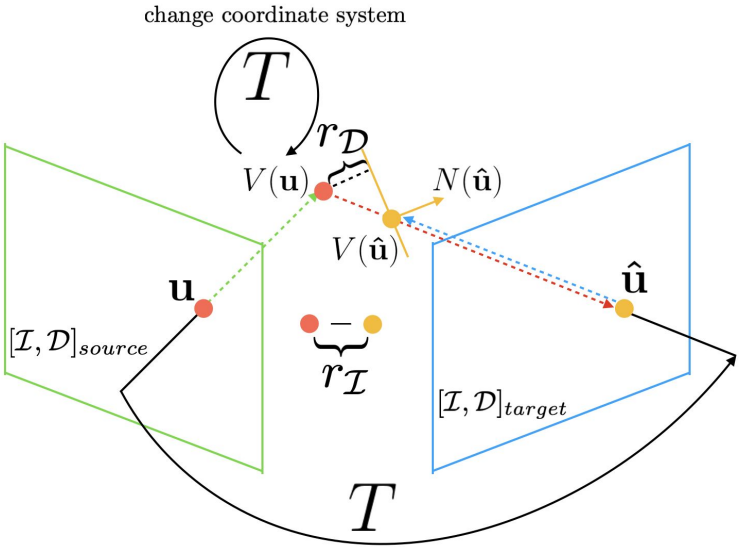
$$\min_{\mathbf{x}\in\mathbb{R}^6} \sum_{\Omega_k(\mathbf{u})\neq\text{null}} \|E\|_2^2$$

$$E = \hat{\mathbf{N}}^g_{k-1}(\hat{\mathbf{u}})^\top \left(\mathbf{G}(\mathbf{u})\mathbf{x} + \widetilde{\mathbf{V}}^g_k(\mathbf{u}) - \hat{\mathbf{V}}^g_{k-1}(\hat{\mathbf{u}}) \right)$$

$$\begin{aligned} \sum_{\Omega_k(\mathbf{u})\neq\text{null}} \left(\mathbf{A}^\top\mathbf{A}\right)\mathbf{x} &= \sum \mathbf{A}^\top\mathbf{b}, \\ \mathbf{A}^\top &= \mathbf{G}^\top(\mathbf{u})\hat{\mathbf{N}}^g_{k-1}(\hat{\mathbf{u}}), \\ \mathbf{b} &= \hat{\mathbf{N}}^g_{k-1}(\hat{\mathbf{u}})^\top \left(\hat{\mathbf{V}}^g_{k-1}(\hat{\mathbf{u}}) - \widetilde{\mathbf{V}}^g_k(\mathbf{u}) \right) \end{aligned}$$

RGB-D odometry: an extension

- Combine a color term
- Linear system is harder to build
  - T encoded in index
  - Need numerical image gradient
- Usually more stable



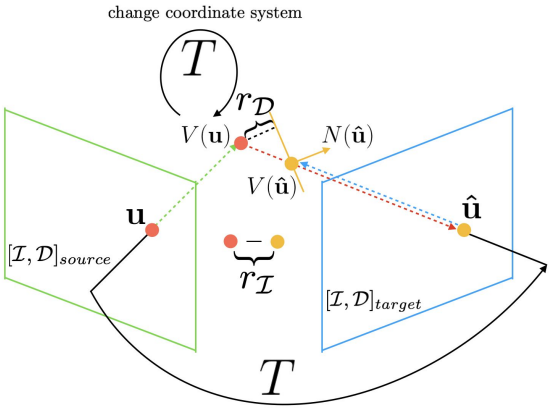
Real-time visual odometry from dense RGB-D images, ICCVW 2011
Colored point cloud registration revisited, ICCV 2017

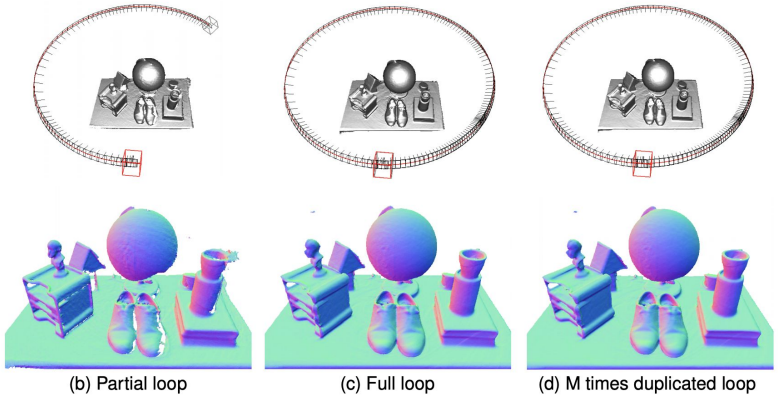
Frame-to-frame odometry: drift

- Frame-to-frame odometry suffers from drift
- Reminder: we cannot trivially apply global optimization on volumes…
  - iSAM is not applicable...

Frame-to-model: an attempt

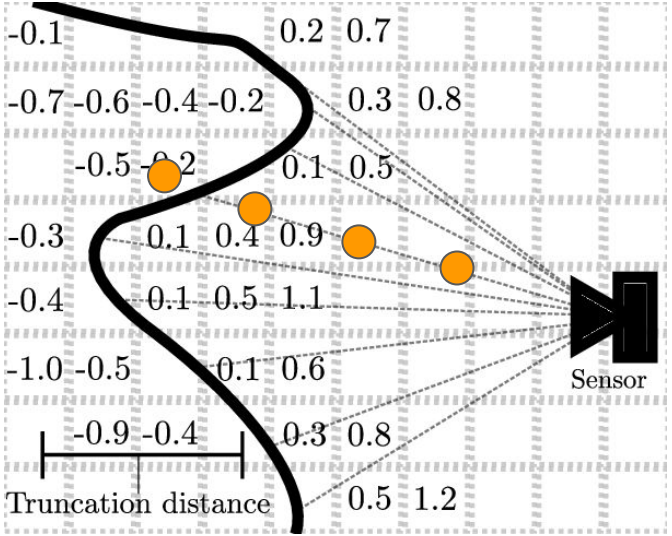
- Target depth:
  - **generated from map (volumes)** rather than previous frame
  - ‘virtual camera’ with previous pose
- Ideally: contains fused map information and can be more stable
- ...reality: usually only works well for object-centric scenes





Frame-to-model: depth generation

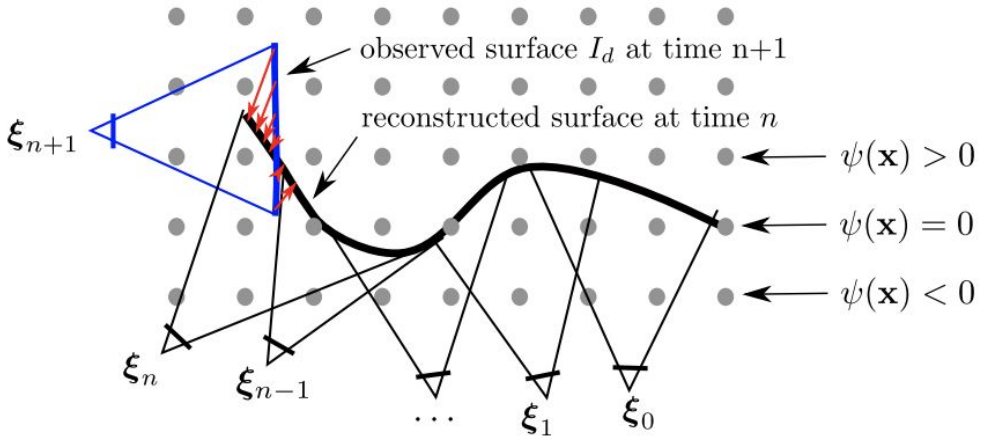
- Reuse the query function in SDF
- For each pixel in virtual camera:
  - Unproject a unit ray
  - Iterate with fixed step size and query SDF
  - Find zero crossing via interpolation
  - Return terminating depth
- **Ray marching / ray casting**
  - Also critical for Deep Implicit Functions inference



Real-time large-scale dense RGB-D SLAM with volumetric fusion, IJRR 2014

Frame-to-SDF

- Directly minimize **T** without ray marching
  - Unprojected and transformed (by **T**) points are generally on surfaces
  - SDF should all be close to 0!



$$\min_{\Delta\xi} \sum_{x \in S} w_t ||\mathcal{D}_t \Big(T(\xi; \Delta\xi) x \Big)||^2$$

Real-Time Camera Tracking and 3D Reconstruction Using Signed Distance Functions, RSS 2013

Brute force volumetric bundle adjustment

- Apply BA with features independently
- De-integrate affected volumes with old poses
- Re-integrate with optimized poses
- BundleFusion, TOG 2017

Summary

- RGB-D tracking aligns surfaces (optionally colors)
- Frame-to-model may reduce drift
- ‘Model’s are actually depth frames from SDF
- Map optimization in dense SLAM is still not fully solved

Outline

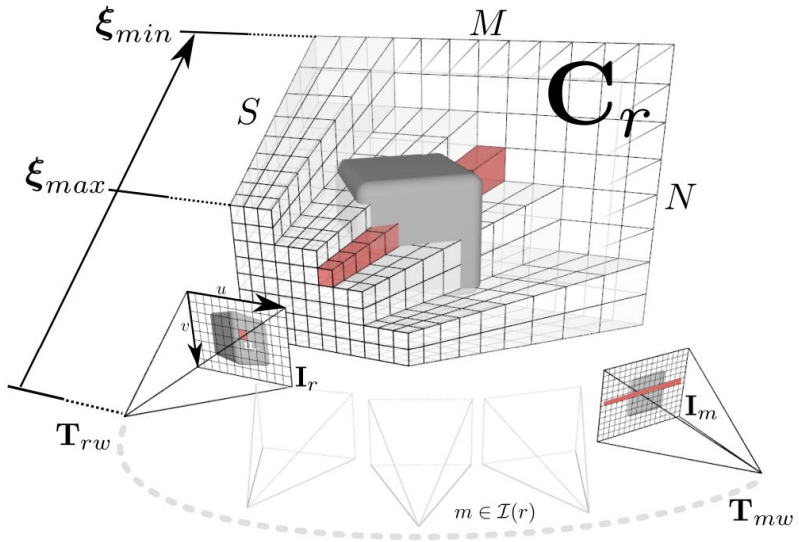
- Intro
- Mapping: point-based fusion
- Mapping: implicit function and SDF
- Tracking: RGB-D odometry
- **Depth estimation: from TV to deep learning**

DTAM

- How can we perform KinectFusion with monocular input?
  - **Estimate depth image for keyframes**
  - Perform KinectFusion on keyframes
- In essence: multi-view stereo
- Problem setting
  - Compute **pixel-wise dense depth** for one frame given an image and its neighbors
    - Epiploar geometry doesn’t work… we don’t have pixel-wise correspondences...
  - Assume poses are known (can be obtained by aforementioned method)

DTAM intuition

- Brute force enumeration!
- Each pixel has a range of candidate depths
  - [0.1, 0.2, 0.3, 0.4, …, 3.0]
- Find the best candidate for all pixels
- Problems
  - How to define error to minimize?
  - How to reduce computation?
    - Brute force search is intractable...

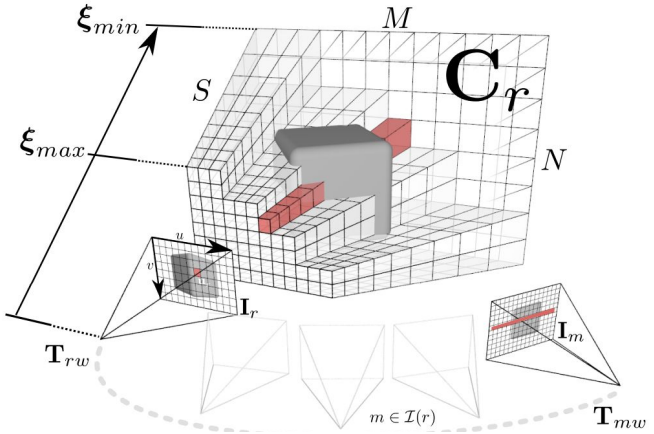


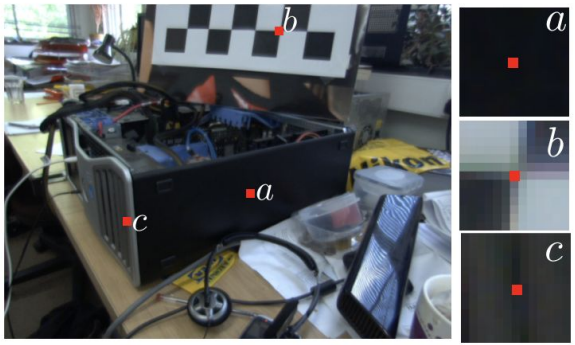
DTAM error formulation

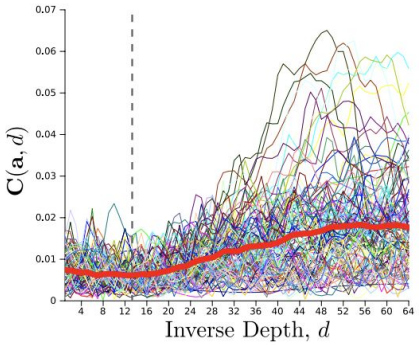
$$\mathbf{C}_r(\mathbf{u}, d) = \frac{1}{|\mathcal{I}(r)|} \sum_{m \in \mathcal{I}(r)} \|\rho_r\left(\mathbf{I}_m, \mathbf{u}, d\right)\|_1 ,$$

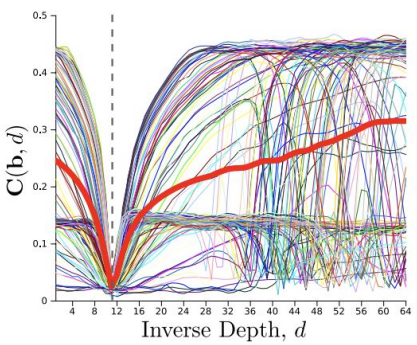
$$\rho_r\left(\mathbf{I}_m, \mathbf{u}, d\right) = \mathbf{I}_r\left(\mathbf{u}\right) - \mathbf{I}_m\left(\pi\left(\mathrm{KT}_{mr}\pi^{-1}\left(\mathbf{u}, d\right)\right)\right) .$$

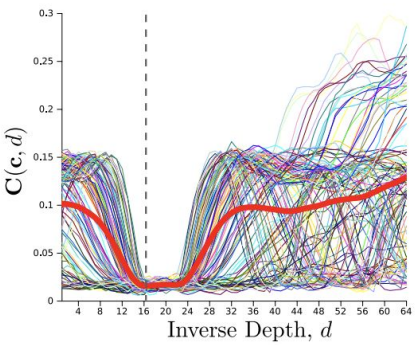
- Correct depth leads to accurate projections to other frames
- Accurate projections lead to similar appearance or small photometric error
- **Can be pre-computed!**
- Problem: images are highly non-convex





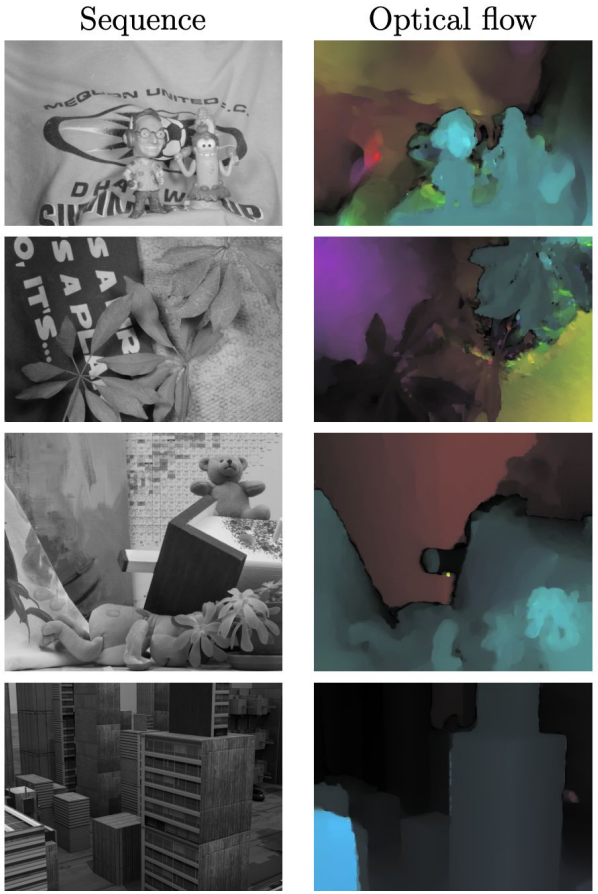






DTAM prior: Total Variation

- Total variation regularization: $||\nabla \mathbf{x}||_\epsilon$
- Piecewise constant prior
  - Prefers zero gradients: flat regions
  - Accepts sharp ‘delta gradients’: preserves boundaries
- Initially designed for optical flow
- Frequently used in the pre-deep learning era

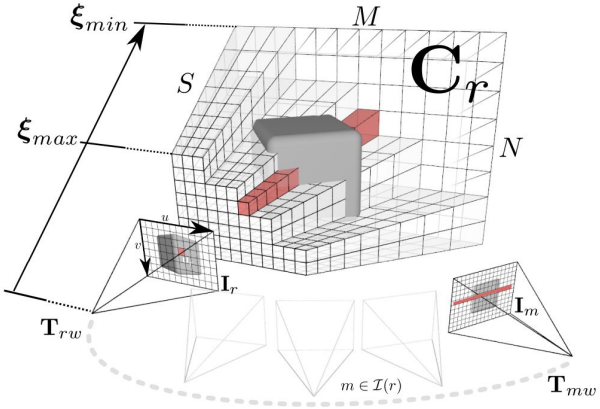


TV-L1 Optical Flow Estimation, IPOL 2013

DTAM loss: don’t be scared!

$$E_{\boldsymbol{\xi}} = \int_{\Omega} \left\{ g(\mathbf{u}) \|\boldsymbol{\nabla}\boldsymbol{\xi}(\mathbf{u})\|_{\epsilon} + \lambda \mathbf{C}\left(\mathbf{u}, \boldsymbol{\xi}(\mathbf{u})\right) \right\} \mathrm{d}\mathbf{u} \,.$$

- ‘Integration’ is just a sum up pixels
- **g(u)**: merely a weight to emphasize edges
- **Target: optimized depth per pixel**
- For each pixel with its depth value:
  - Compute sum of photometric error on all references
  - Evaluate local piecewise-constant loss



-1	0	+1
-1	0	+1
-1	0	+1

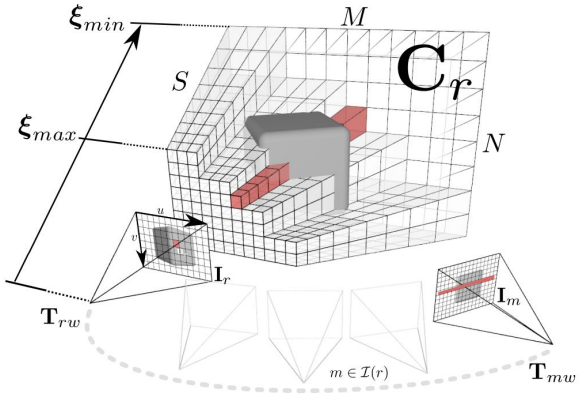
$\mathbf{G}_x$

+1	+1	+1
0	0	0
-1	-1	-1

$\mathbf{G}_y$

DTAM optimization: step 1

$$E_{\boldsymbol{\xi}} = \int_{\Omega} \left\{ g(\mathbf{u}) \|\boldsymbol{\nabla}\boldsymbol{\xi}(\mathbf{u})\|_{\epsilon} + \lambda \mathbf{C}\left(\mathbf{u}, \boldsymbol{\xi}(\mathbf{u})\right) \right\} \mathrm{d}\mathbf{u} \,.$$



- Relaxation
  - Optimize a differentiable continuous variable
  - But our discrete candidates are searchable but not differentiable
  - Optimize differentiable variable, minimize error between discrete candidates

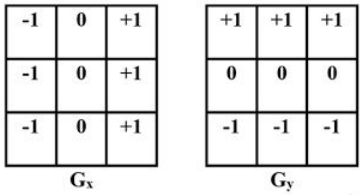
$$\mathbf{E}_{\boldsymbol{\xi},\boldsymbol{\alpha}} = \int_{\Omega} \left\{ g(\mathbf{u}) \|\boldsymbol{\nabla}\boldsymbol{\xi}(\mathbf{u})\|_{\epsilon} + \frac{1}{2\theta} \left(\boldsymbol{\xi}(\mathbf{u}) - \boldsymbol{\alpha}(\mathbf{u})\right)^2 + \lambda \mathbf{C}\left(\mathbf{u}, \boldsymbol{\alpha}(\mathbf{u})\right) \right\} \mathrm{d}\mathbf{u} \,.$$

DTAM optimization: step 2

Applications of Legendre-Fenchel transformation to computer vision problems

Ankur Handa, Richard A. Newcombe, Adrien Angeli, Andrew J. Davison

$$\mathbf{E}_{\boldsymbol{\xi},\boldsymbol{\alpha}} = \int_{\Omega} \left\{ g(\mathbf{u}) \|\nabla \boldsymbol{\xi}(\mathbf{u})\|_{\epsilon} + \frac{1}{2\theta} \left(\boldsymbol{\xi}(\mathbf{u}) - \boldsymbol{\alpha}(\mathbf{u})\right)^2 + \lambda \mathbf{C}\left(\mathbf{u}, \boldsymbol{\alpha}(\mathbf{u})\right) \right\} \mathrm{d}\mathbf{u} \, .$$



- Convex conjugate
  - Huber loss contains non-differentiable L1
  - Legendre-Fenchel transformation is a specific tool to resolve such problems
  - Huber loss becomes the complex combination with introduced dual **q** (but easier to optimize)

$$\|AG\mathbf{d}\|_{\epsilon} = \underset{\mathbf{q},\ \|\mathbf{q}\|_2 \leq 1}{\arg\max} \left\{ \langle AG\mathbf{d}, \mathbf{q} \rangle - \delta_q(\mathbf{q}) - \frac{\epsilon}{2} \|\mathbf{q}\|_2^2 \right\},$$

d: vectorized depth
AG: matricized gradient operator
q: dual variable

DTAM optimization: step 3

$$\arg\max_{\mathbf{q},\ \|\mathbf{q}\|_2 \leq 1} \{\arg\min_{\mathbf{d},\mathbf{a}} \ \mathbf{E}(\mathbf{d},\mathbf{a},\mathbf{q})\} \tag{9}$$

$$\mathbf{E}(\mathbf{d},\mathbf{a},\mathbf{q}) = \Big\{ \langle AG\mathbf{d},\mathbf{q}\rangle + \frac{1}{2\theta}\|\mathbf{d}-\mathbf{a}\|_2^2 + \lambda \mathbf{C}(\mathbf{a}) - \delta_q(\mathbf{q}) - \frac{\epsilon}{2}\|\mathbf{q}\|_2^2 \Big\}. \tag{10}$$

A Tutorial on Primal-Dual Algorithm

Shenlong Wang

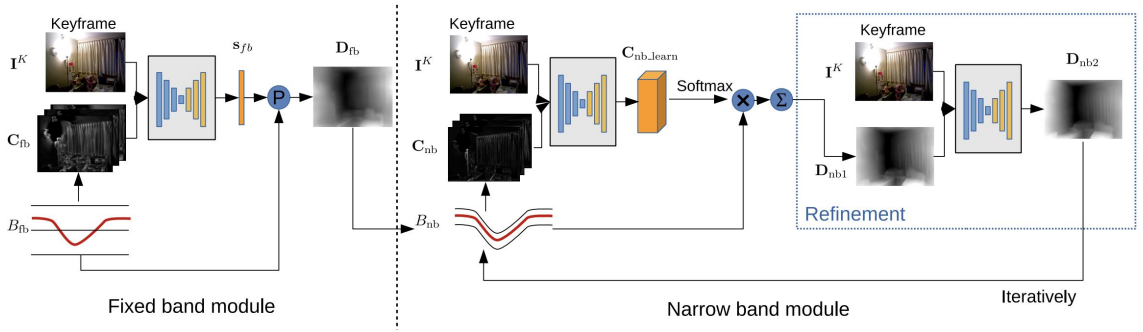
University of Toronto

March 31, 2016

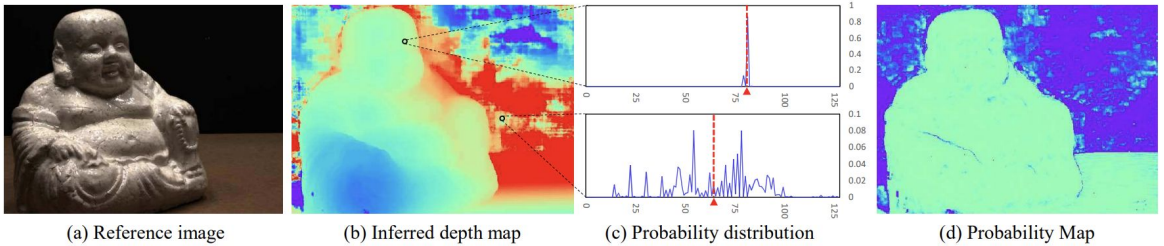
- Gradient ascent over **q**
- Gradient descent over **d**
- Discrete search discrete **a** around continuous **d**
- Iterate until convergence

DTAM problem

- Toooooo complex, never open-sourced, never reproduced :-(
- Revive at the deep learning age
  - Prior? Neural networks!
  - Optimization? SGD!

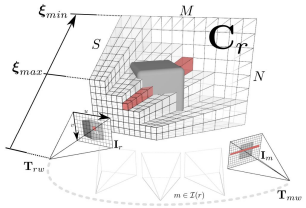


DeepTAM, ECCV 2018

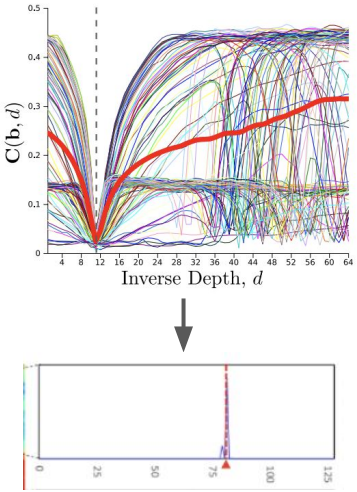


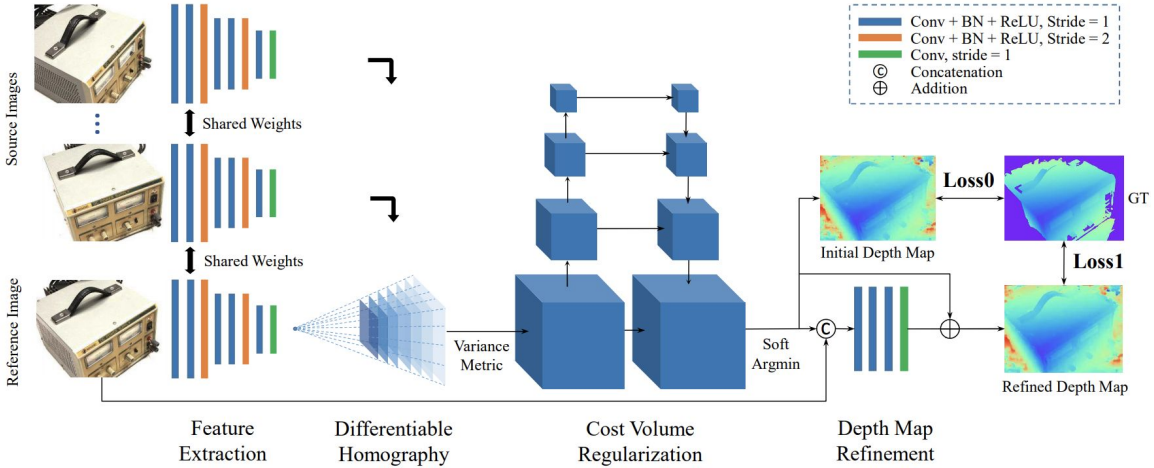
MVSNet, ECCV 2018

Deep variations



- Still maintain the cost volume **C** (colors can be replaced with deep features)
- Learn input cost -> output probability
  - Prior hidden in network weights
- Softmax: probability -> depth





MVSNet, ECCV 2018

Summary

- Exhaustive depth search can predict dense depth images
- Classical prior and optimization are hard to generalize
- Yet the DTAM problem formulation provides critical insights to deep depth estimation