3D Reconstruction for AR

Prof Victor Adrian Prisacariu

Active Vision Lab
Department of Engineering Science, University of Oxford

6D.ai

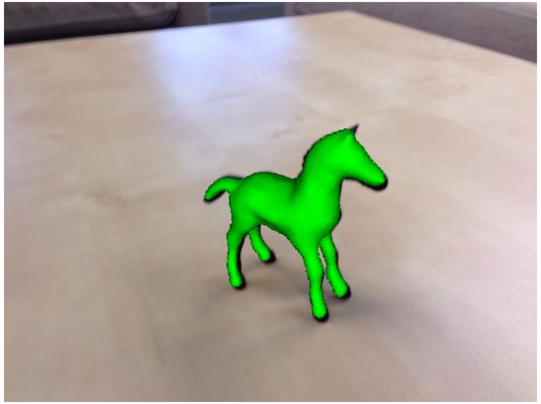
What do we want?

Track and reconstruct the world:

- in unstructured, real world environments;
- with little user intervention;
- in real time, on a mobile phone.
- without depth cameras.

3D Models for Objects and Scenes

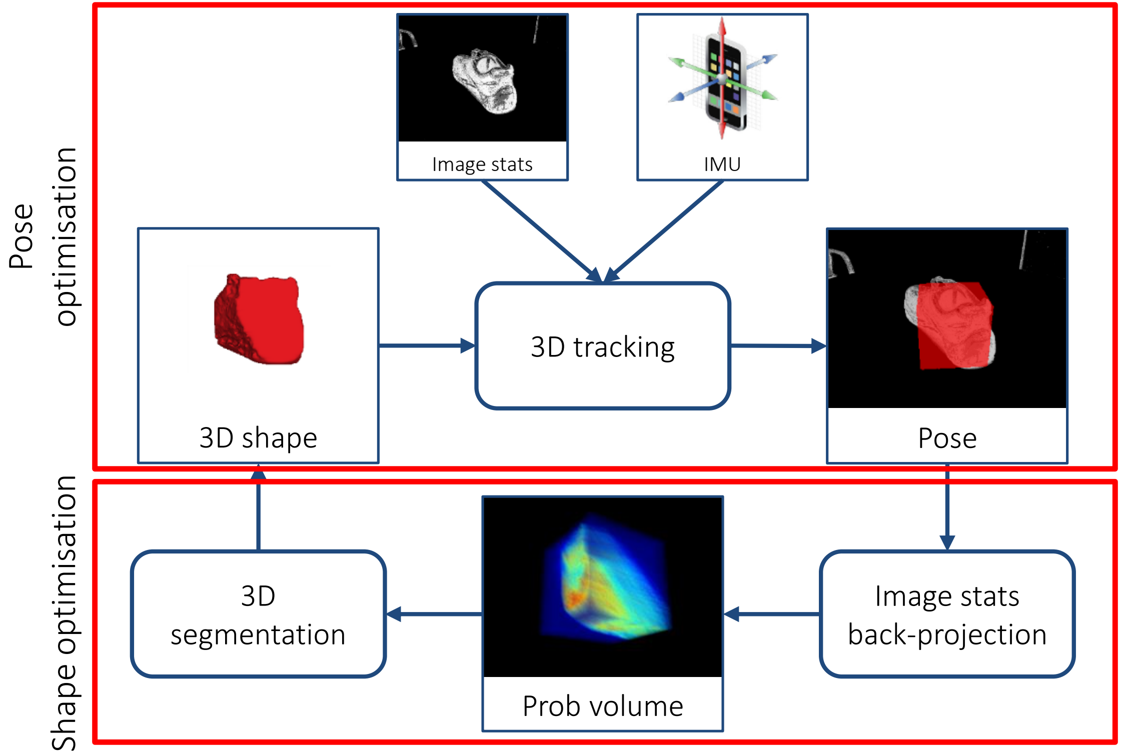
Objects



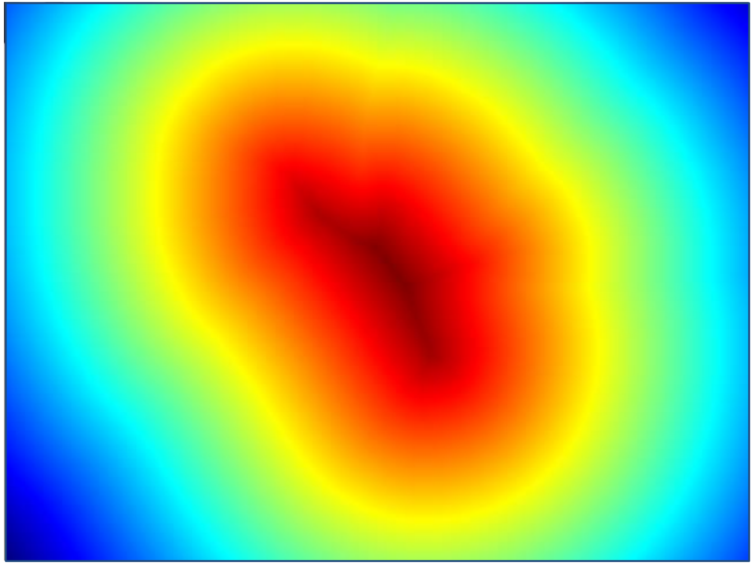
Full Scenes

3D Object Reconstruction and Tracking

How would we do it?



Pose Optimisation



Assumes known 3D shape and per-pixel image statistics.

P_f, P_b – image statistics (e.g. histograms)

Φ – contour embedding SDF.

H_e – Heaviside function

$$E(\Phi) = -\log \sum_{x \epsilon I} \{H_e(\Phi) P_f + (1 - H_e(\Phi)) P_b\}$$

2D Segmentation and 3D pose: $$\frac{\delta E(\Phi, pose)}{\delta pose}$$

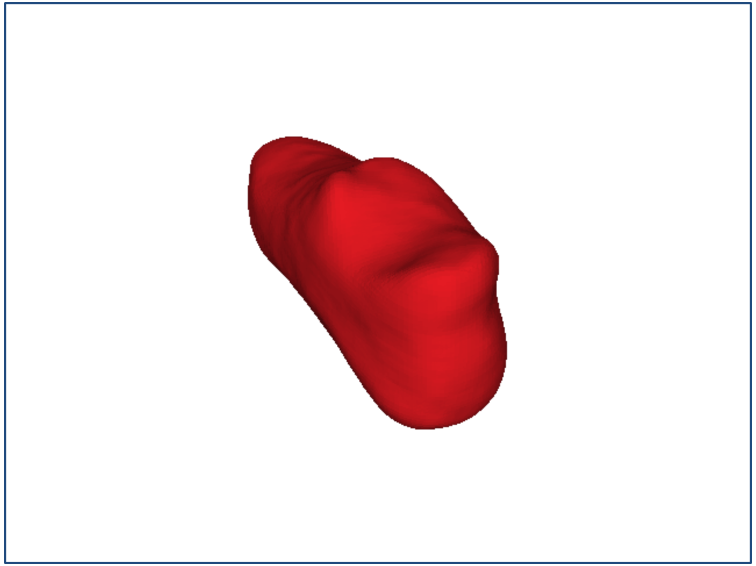
Pose Derivative

$$\frac{\partial E}{\partial \text{pose}} = \left(\text{term wrt } \mathrm{P}_f - P_b\right) \times \frac{\partial \text{SDF}}{\partial \text{position}} \times \frac{\partial \text{position}}{\partial \text{pose}}$$

Requires:

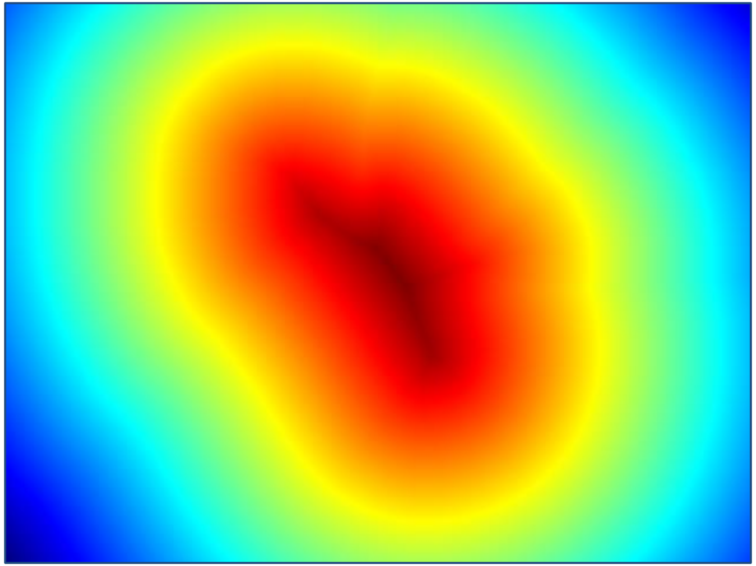
- Fast rendering of 3D shape
- Signed distance transform + derivative.

Fast Rendering



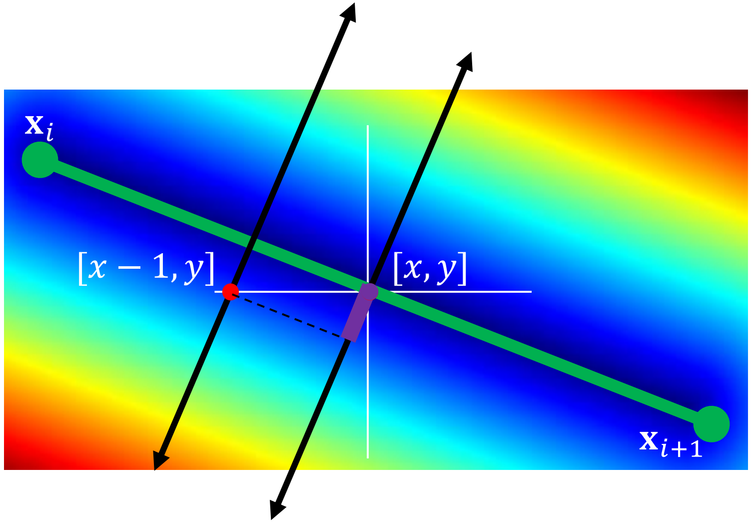
- Model is stored as 3D volume: standard rendering (raycasting) is very slow.
- We use a hierarchical binary raycaster.
  - Alternate between image resizing and raycasting around the contour.

SDF + Derivatives



- Computing full SDF + derivatives is very slow.
- We use per-contour-point local approximations.

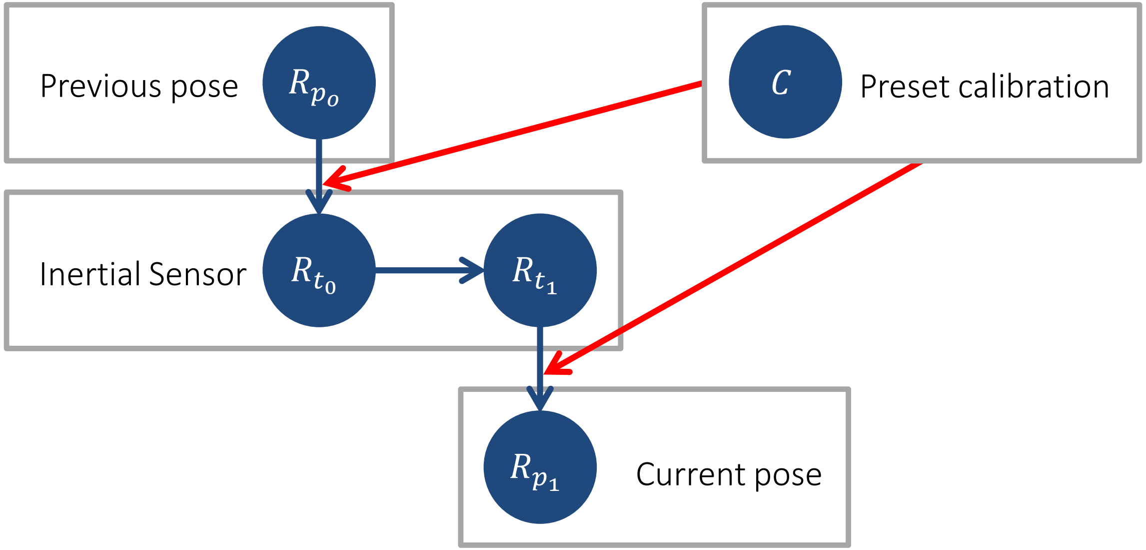
SDF + Derivatives



- Values of the SDF are obtained by following the per point normal.
- The SDF derivatives are computed using finite differences.

IMU Integration

- The mapping from shape to pose is ambiguous.
- We use the mobile phone IMU to provide disambiguated rotation at each frame.

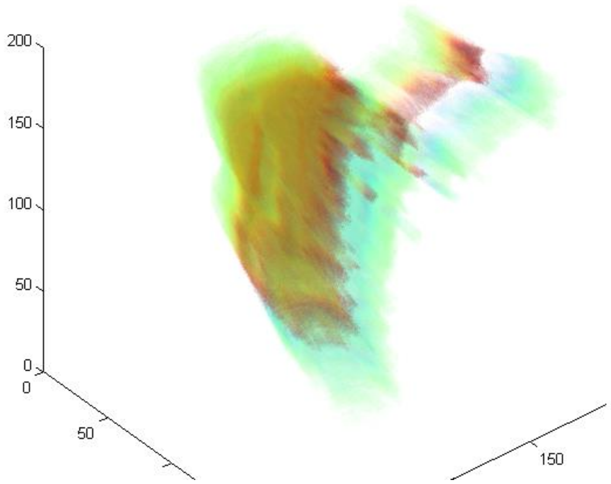


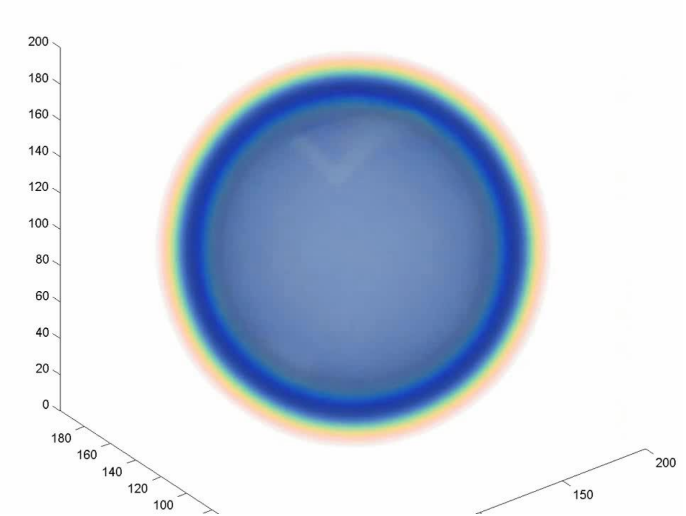
Tracking Results

We obtain speeds > 80 fps on a phone.

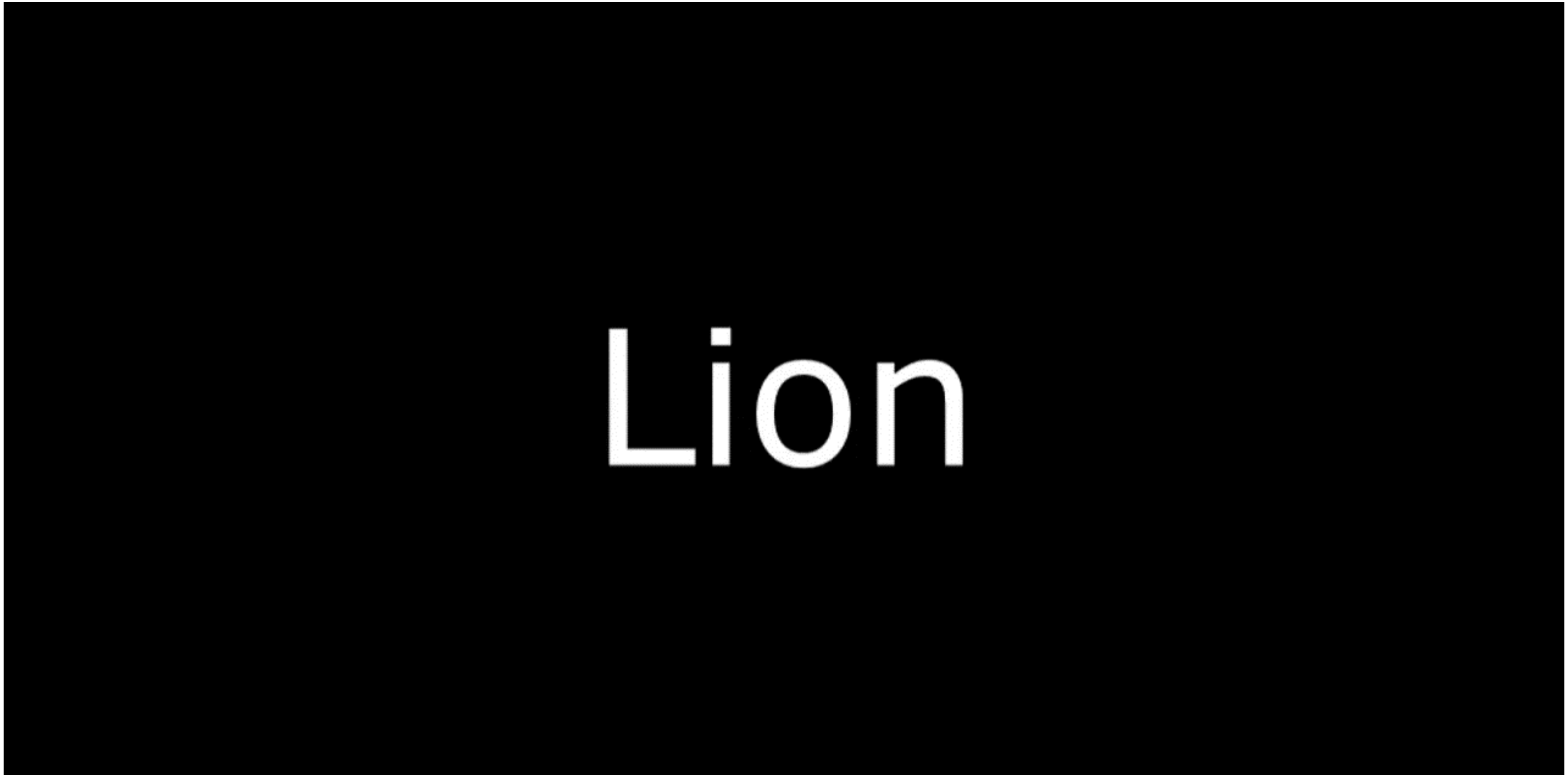
Shape Optimisation

- I assume known 3D pose and per-pixel image likelihoods.
- For a set of images, we build inside/outside membership functions.
- These represent the probability of a voxel being:
  - Inside of the shape (i.e. foreground).
  - Outside of the shape (i.e. background).
- Final shape obtained using a 3D segmentation optimisation.





Reconstruction Results



Reconstruction Results

Reconstruction Results

Reconstruction Results

Conclusions

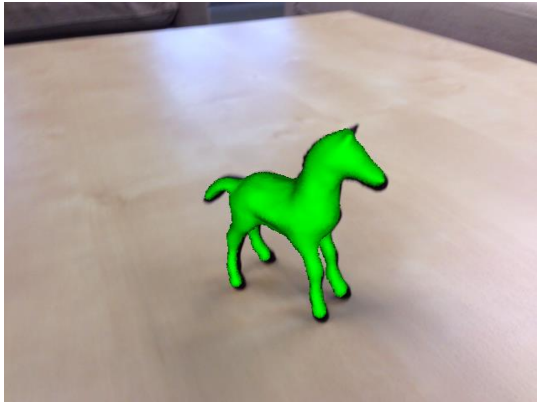
- I demoed an approach for 3D tracking and showed how you could connected it to a reconstruction stage.

- We can get :
  – state of the art 3D tracking speed.
  – state of the art space carving based reconstruction results.

- Processing is fast enough to run on a mobile phone at over 80fps.

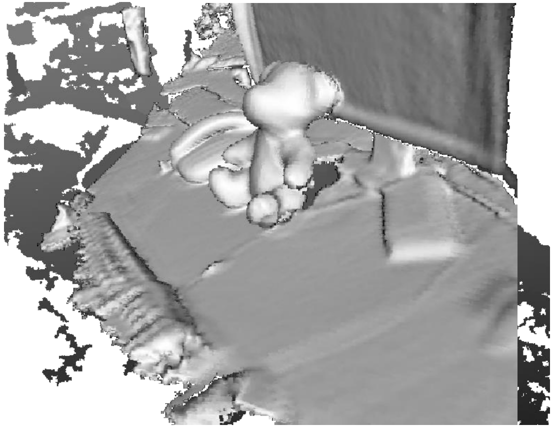
3D Models for Objects and Scenes

Objects



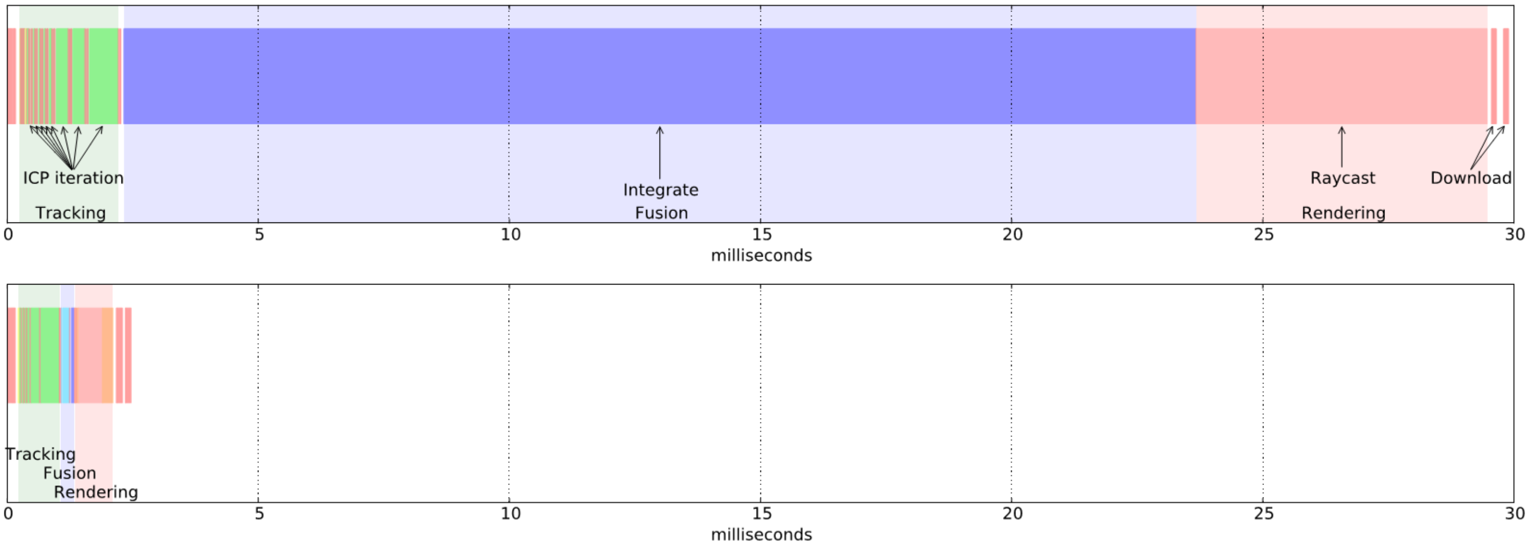
Full Scenes

Depth Fusion



Integration of depth images: **KinectFusion** [Newcombe et al, 2011]

Our Depth Fusion



Integration of depth images: **InfiniTAM**

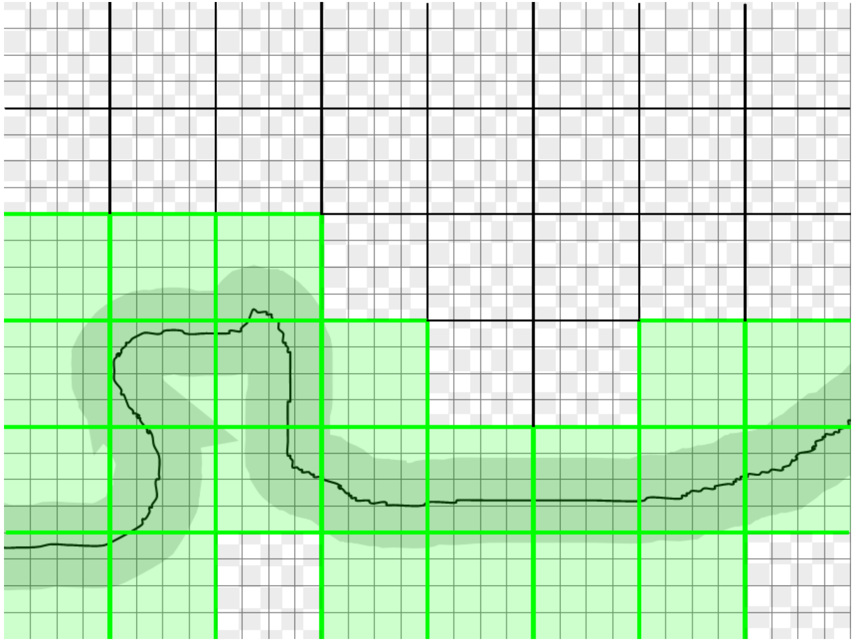
more than **10x speedup (up to kHz speeds).**
runs on **mobile devices.**

InfiniTAM

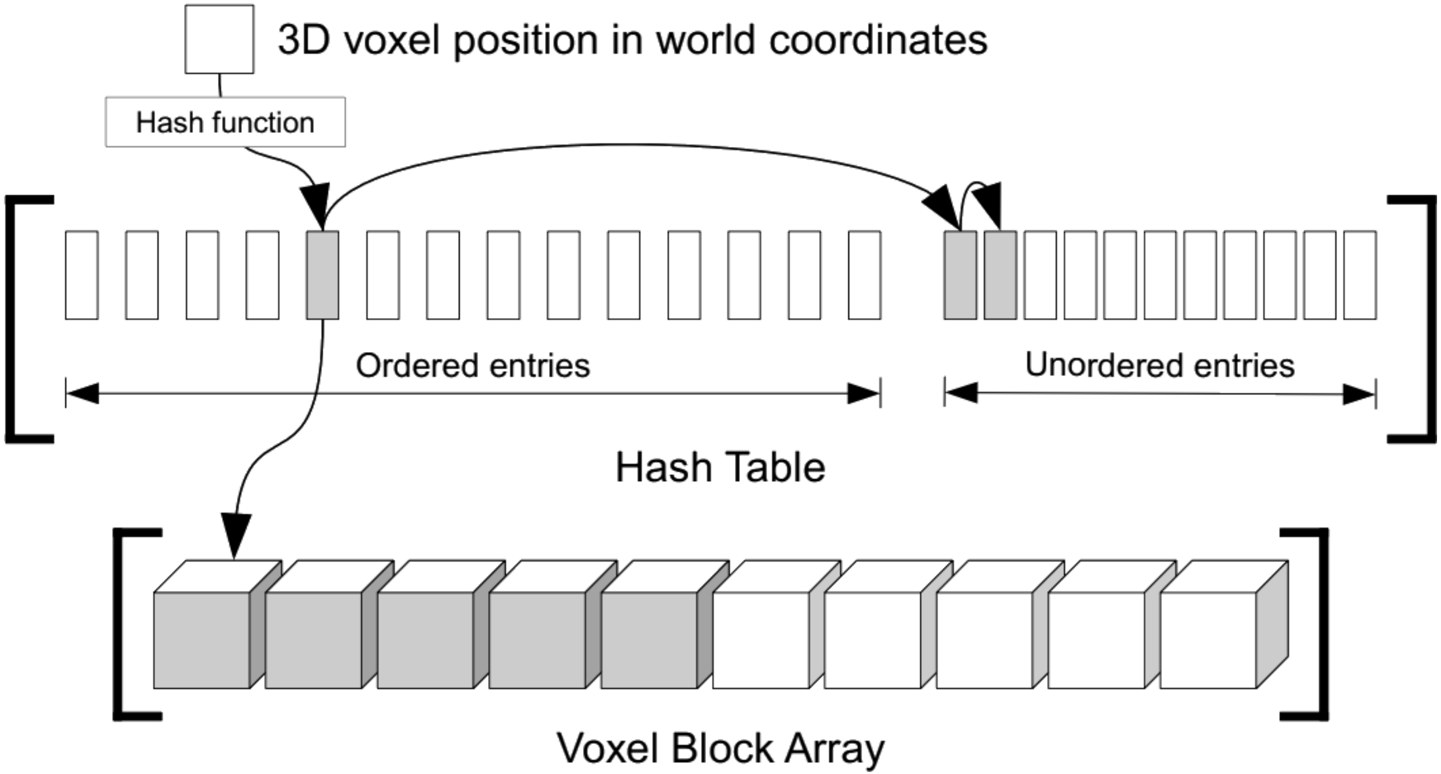
DEMO

Map Representation

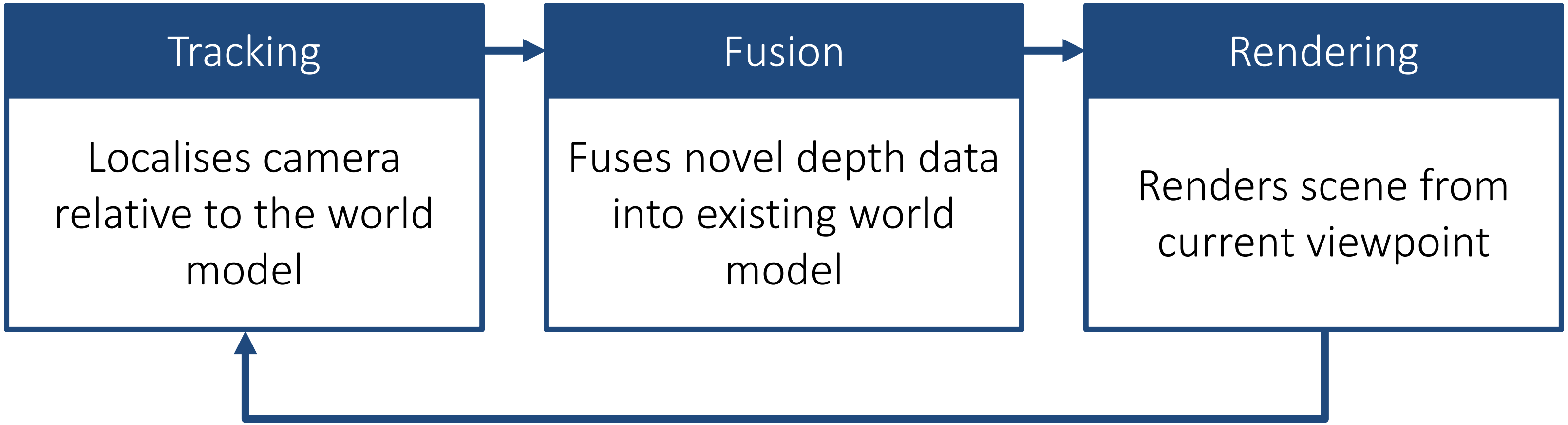
the maptbelyTSDFrisdisoxetizedoiateoVoxelsurfaces



Map Representation

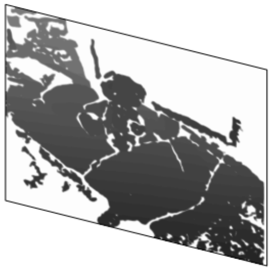


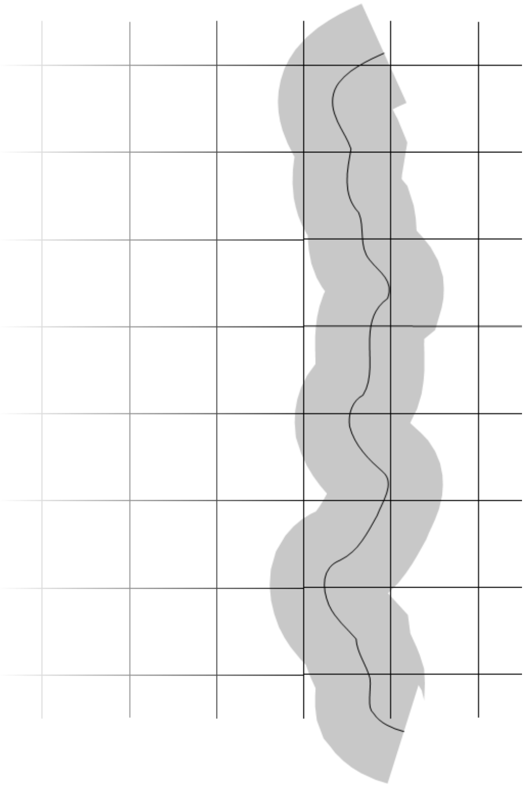
Main Processing Steps



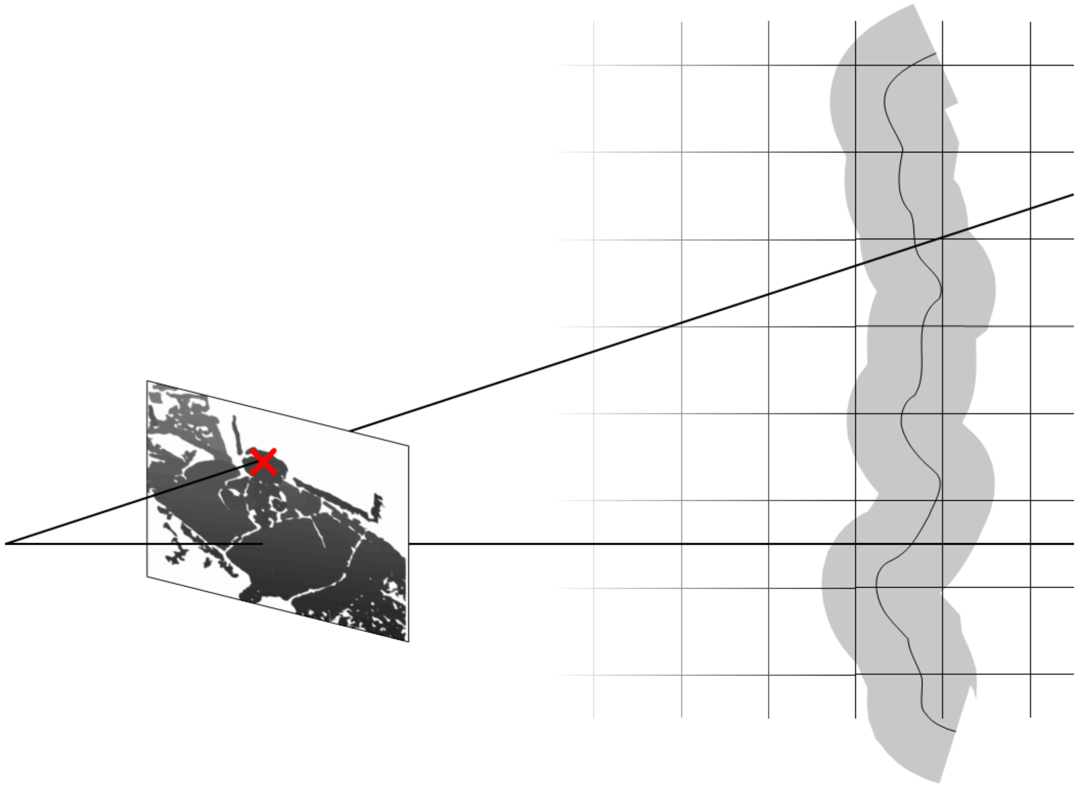
Tracking | **Fusion** | Rendering

1. Allocation
2. Visible List
3. Data Fusion
4. Swapping



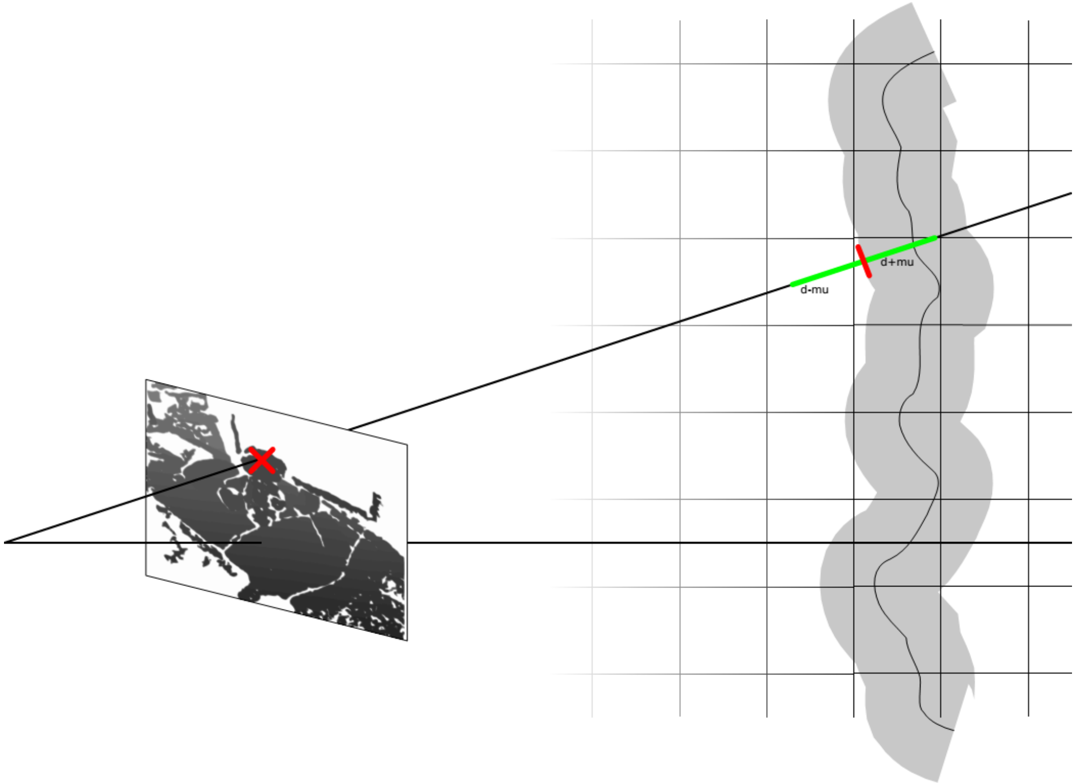


1. Allocation
2. Visible List
3. Data Fusion
4. Swapping



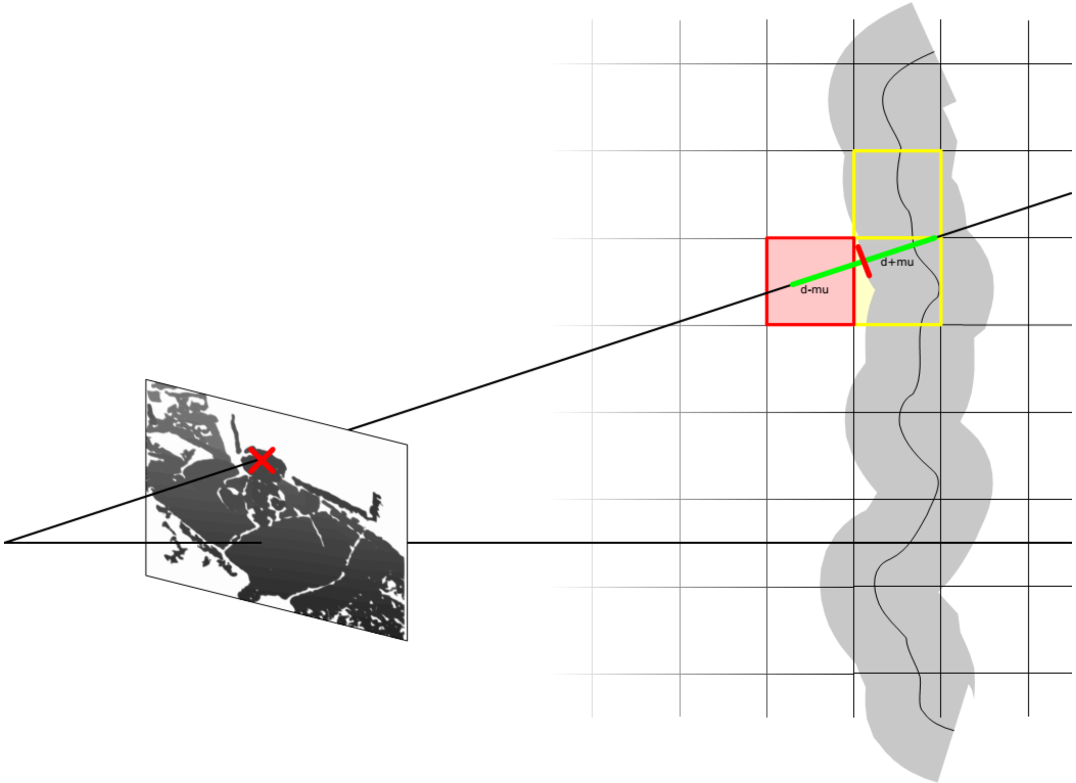
Tracking | Fusion | Rendering

1. Allocation
2. Visible List
3. Data Fusion
4. Swapping



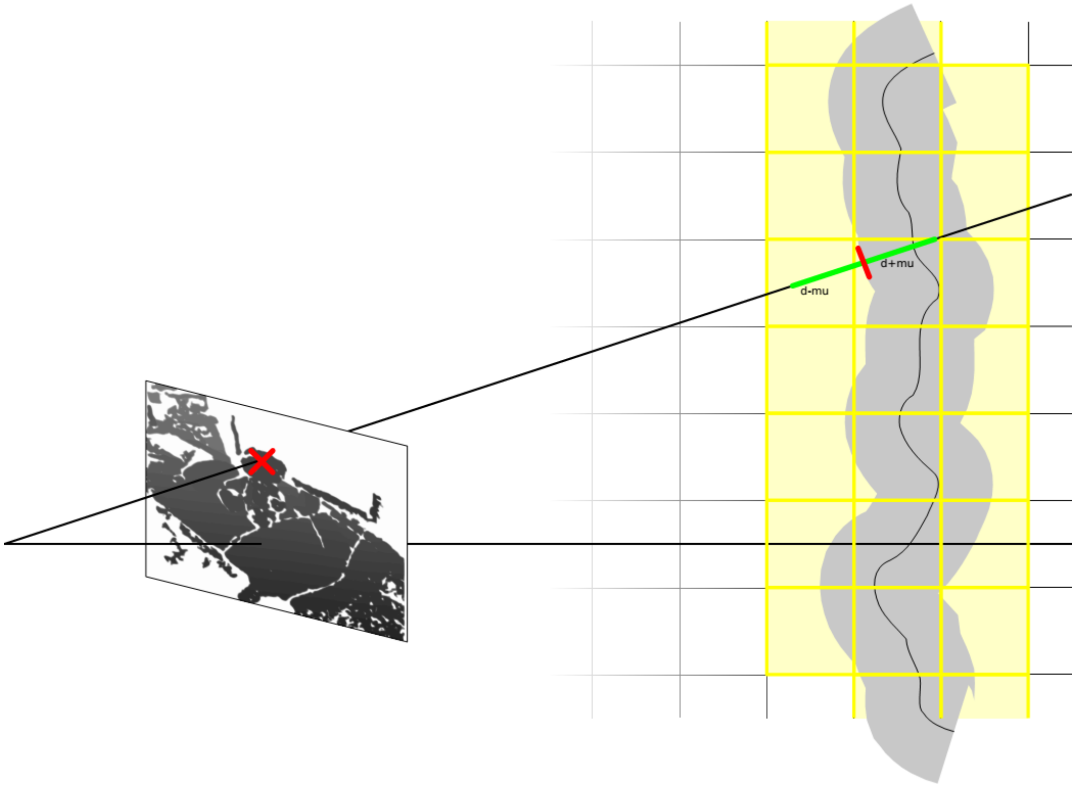
Tracking | **Fusion** | Rendering

1. **Allocation**
2. Visible List
3. Data Fusion
4. Swapping



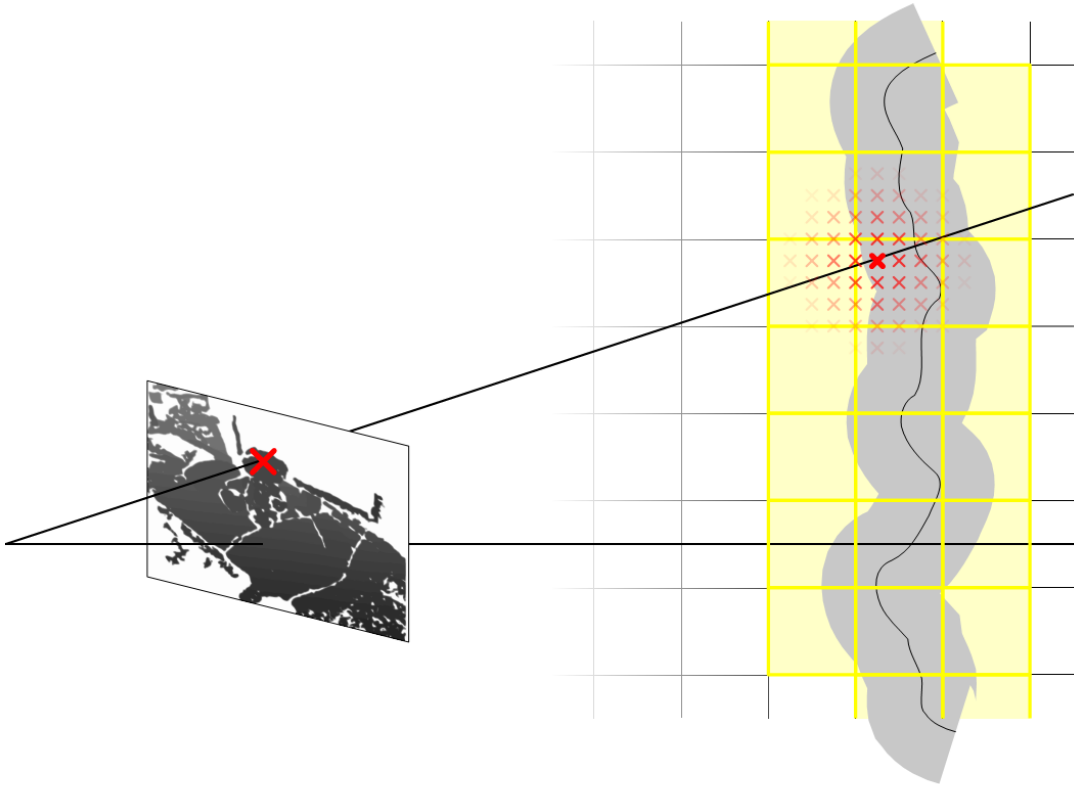
Tracking | **Fusion** | Rendering

1. Allocation
2. **Visible List**
3. Data Fusion
4. Swapping



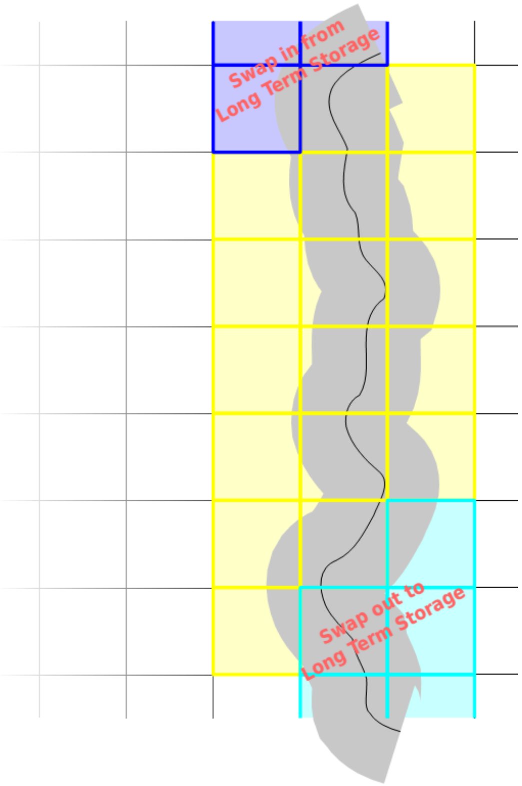
Tracking | Fusion | Rendering

1. Allocation
2. Visible List
3. **Data Fusion**
4. Swapping



Tracking | **Fusion** | Rendering

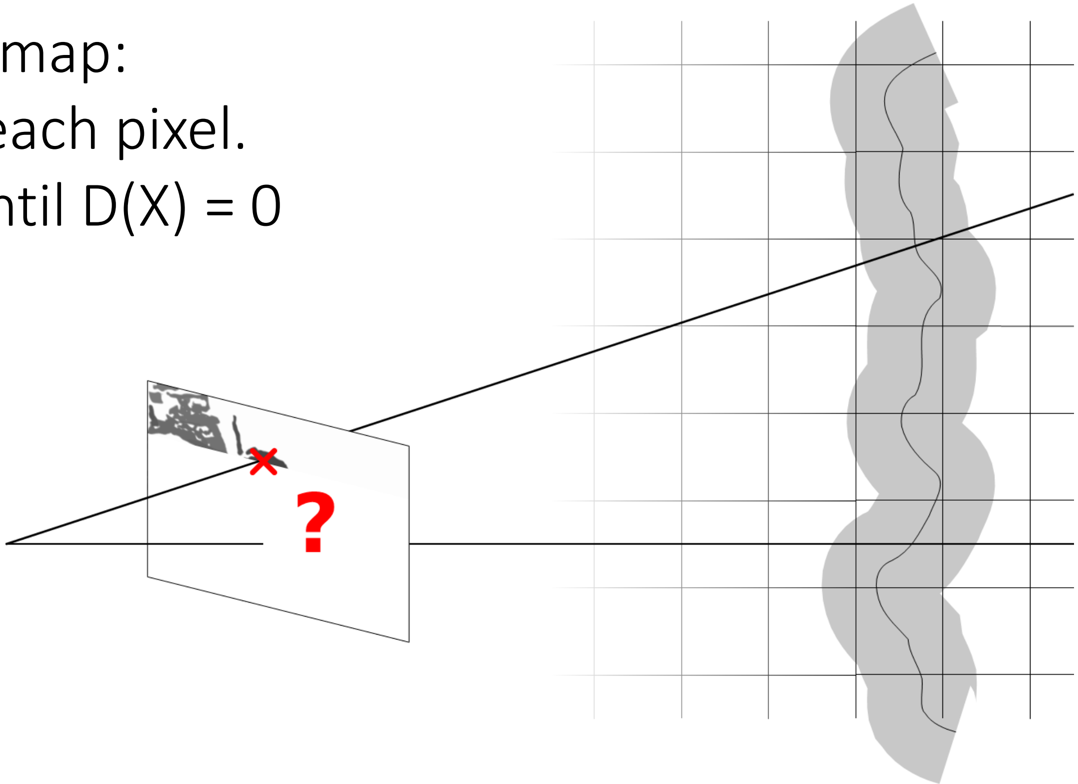
1. Allocation
2. Visible List
3. Data Fusion
4. **Swapping**



Tracking | Fusion | Rendering

Rendering the map:

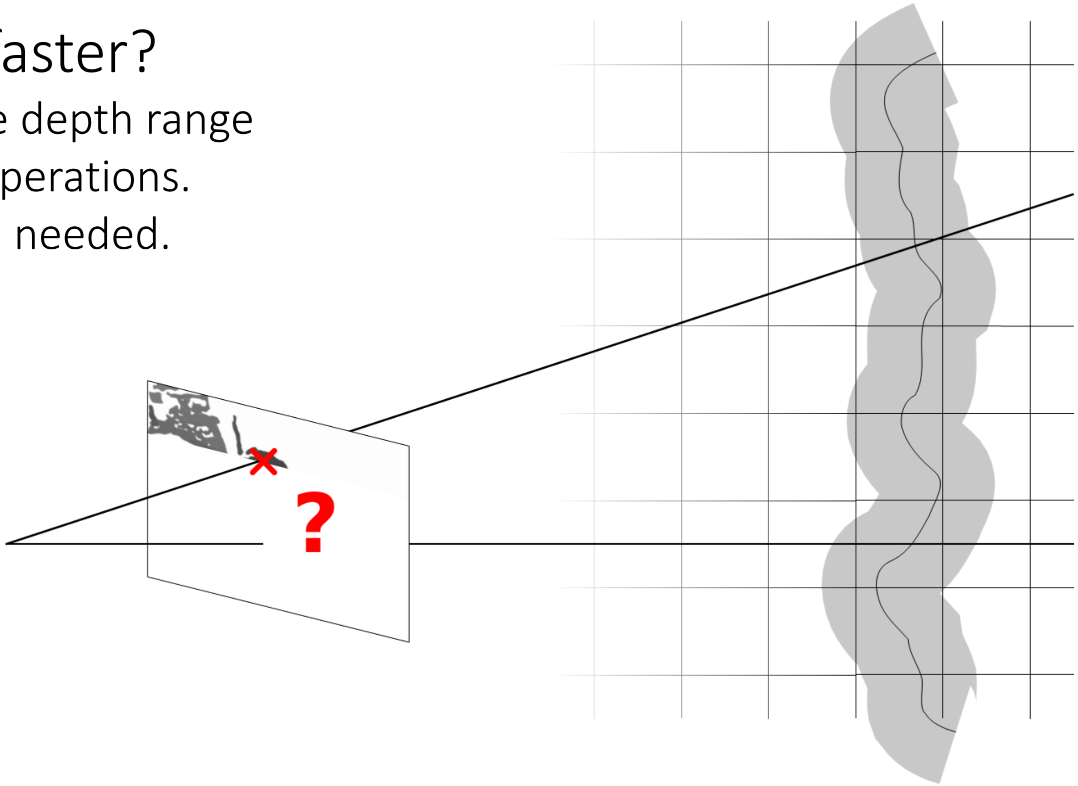
- Cast ray for each pixel.
- Take steps until D(X) = 0



Tracking | Fusion | **Rendering**

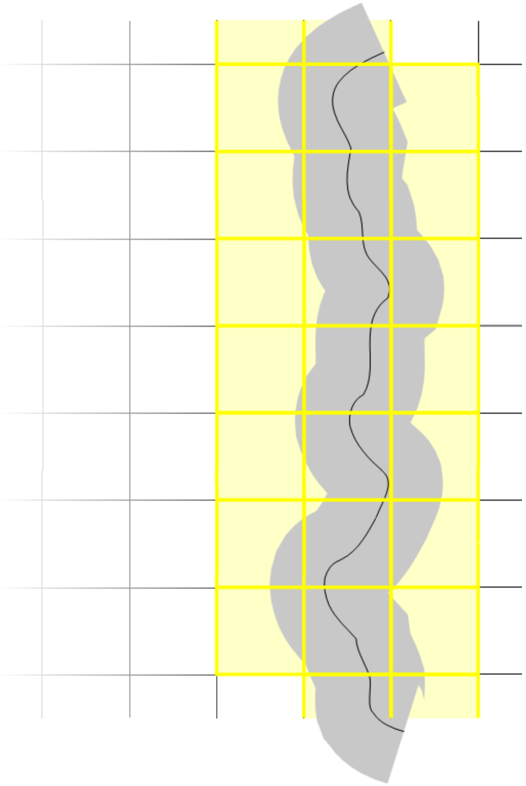
How to get it faster?

- Predict plausible depth range
- Optimise read operations.
- Draw only when needed.



How to get it faster?

- **Predict plausible depth range**
- Optimise read operations.
- Draw only when needed.



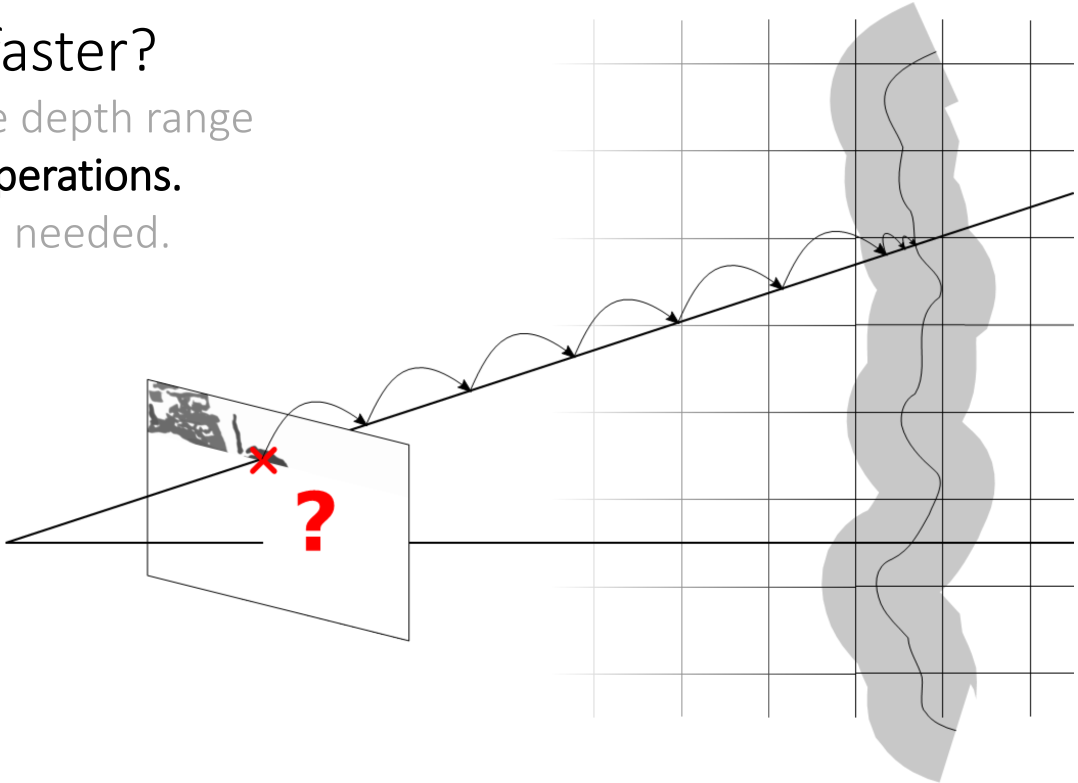
The visible list allows us to predict depth range:

- Forward project visible box
- Bounding box rather than full polygon.
- Low resolution projection

Tracking | Fusion | **Rendering**

How to get it faster?

- Predict plausible depth range
- **Optimise read operations.**
- Draw only when needed.

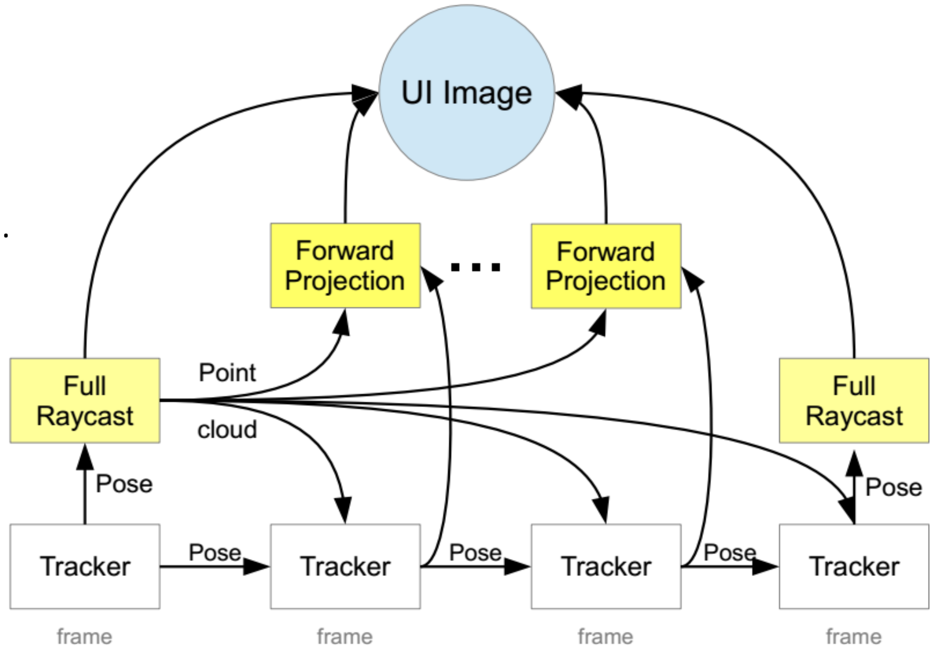


Tracking | Fusion | Rendering

How to get it faster?

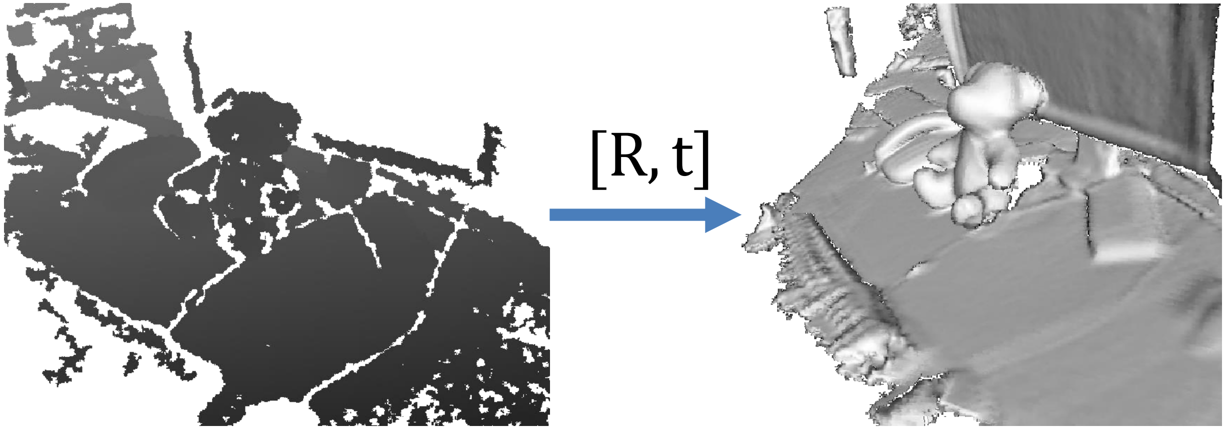
- Predict plausible depth range
- Optimise read operations.
- **Draw only when needed.**

- Full raycast every few frames.
- Forward projection otherwise (much).



Weighted ICP

$$\sum_{p} w_p \left(\left(\mathrm{R}p + \mathrm{t} - V(\bar{p}) \right)^T N(\bar{p}) \right)^2$$



p $V(\bar{p})$

Runtime Experiments

Runtime on different devices:

`teddy` sequence, 640×480 pixels

Device	full	forward	none	[Newcombe et al., 2011]	[Nießner et al., 2013]
Nvidia Titan X	1.91ms	1.74ms	1.38ms	26.15ms	25.87ms
Nvidia Tegra K1	36.53ms	31.38ms	26.79ms	-	-
Apple iPad Air 2	82.60ms	65.55ms	56.10ms	-	-
Intel Core i7-5960X	45.28ms	46.75ms	35.40ms	502.69ms	-

`couch` sequence, 320×240 pixels and IMU

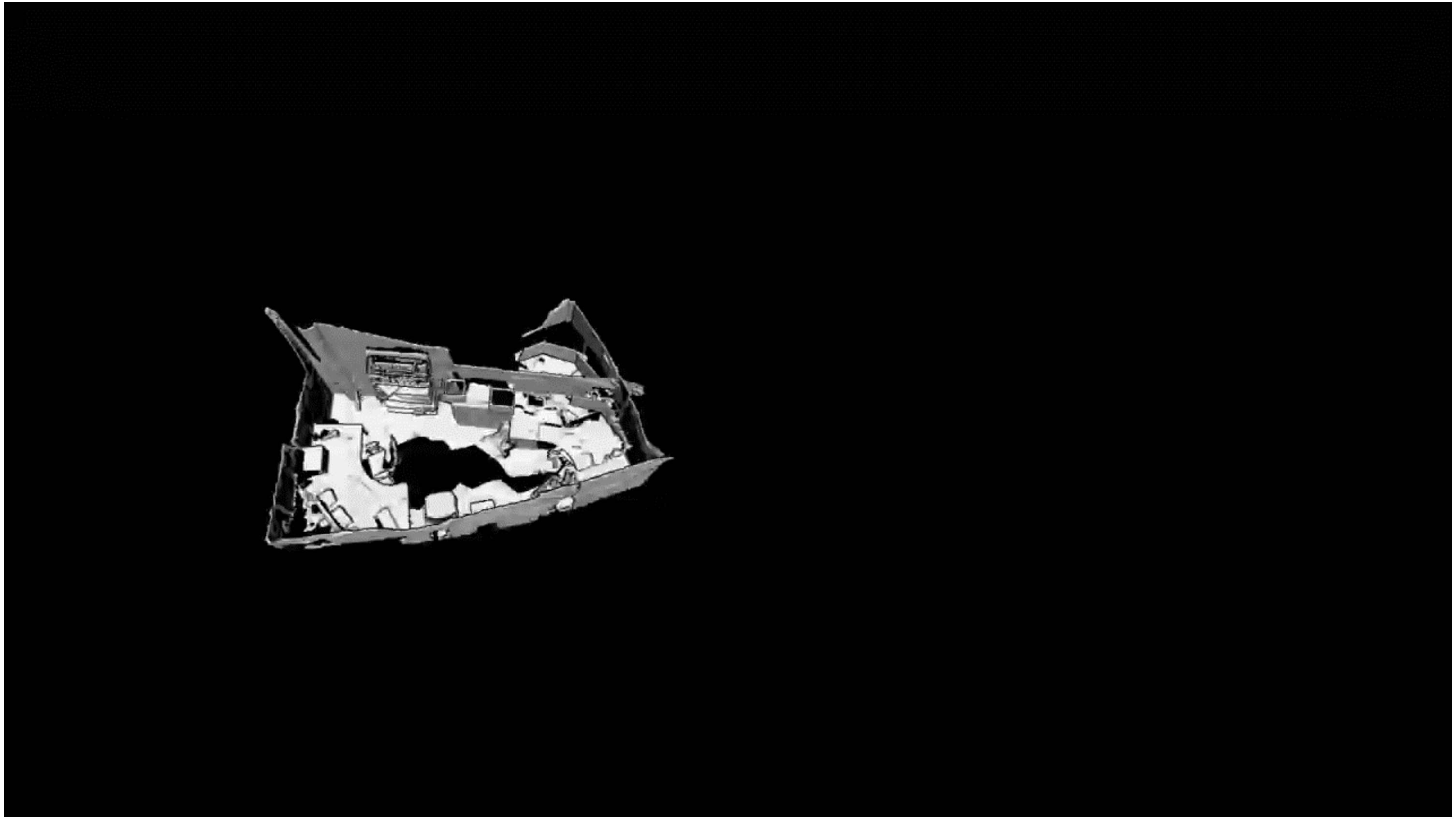
Device	full	forward	none	[Newcombe et al., 2011]	[Nießner et al., 2013]
Nvidia Titan X	1.17ms	1.10ms	0.87ms	19.34ms	15.18ms
Nvidia Tegra K1	25.58ms	21.04ms	19.38ms	-	-
Apple iPad Air 2	56.65ms	48.43ms	41.58ms	-	-
Intel Core i7-5960X	23.43ms	23.38ms	19.94ms	312.86ms	-

Is it perfect? – No ☺

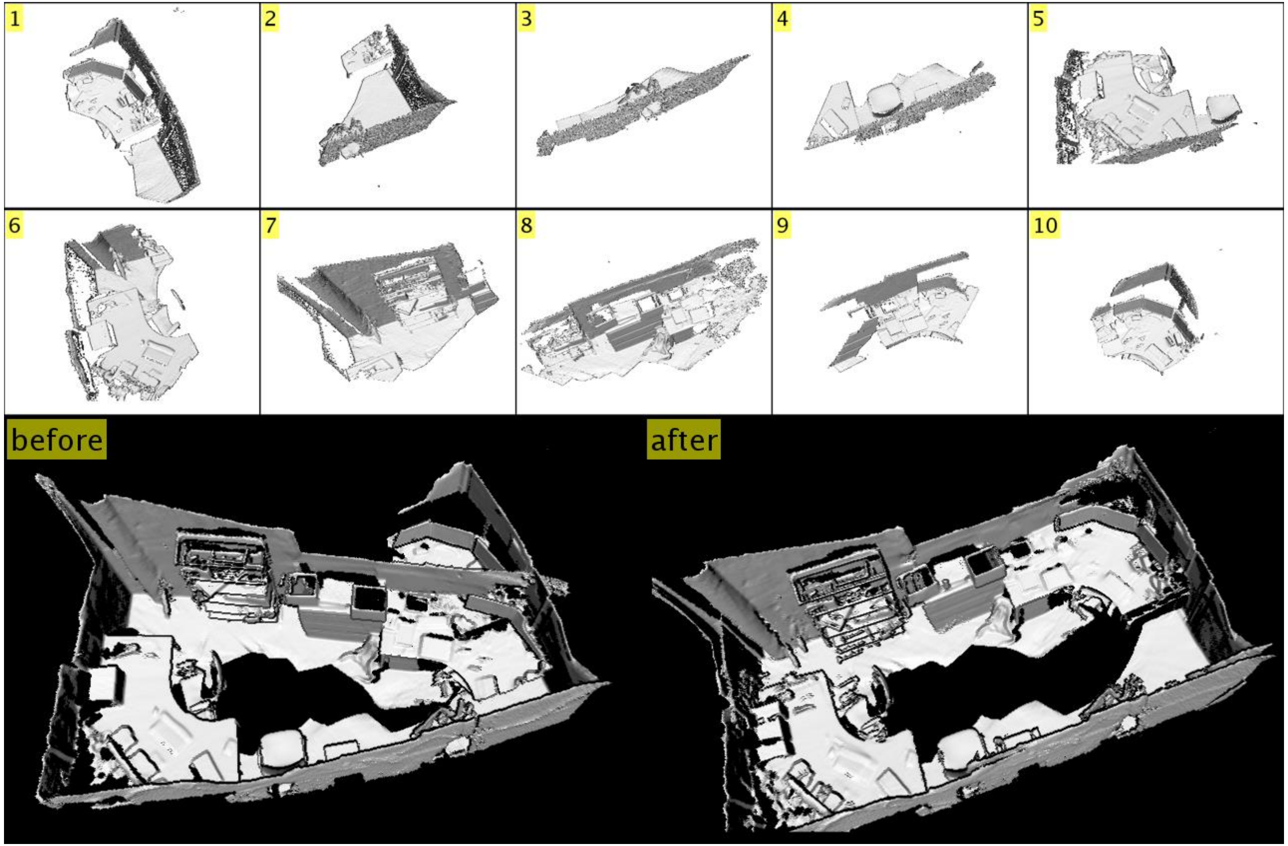
Biggest problem: **tracking drift**

loop closure

Loop Closure

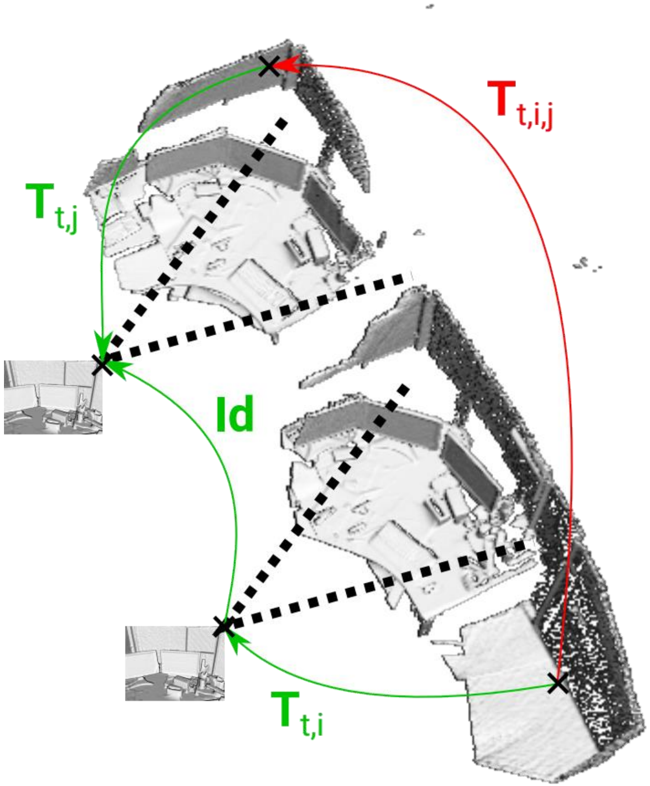


Graph of Submaps



Relative Constraints

- Track the same image in multiple submaps i and j: poses $\mathbf{T}_{t,i}$ and $\mathbf{T}_{t,j}$.
- Pose between submaps: $\mathbf{T}_{t,i,j} = \mathbf{T}_{t,j}^{-1}\mathbf{T}_{t,i}$
- Robustly aggregate over time t to get final estimate $\mathbf{T}_{i,j}$.
- Stop tracking old scene on tracker failure.
- Also add constraints on relocalisation.

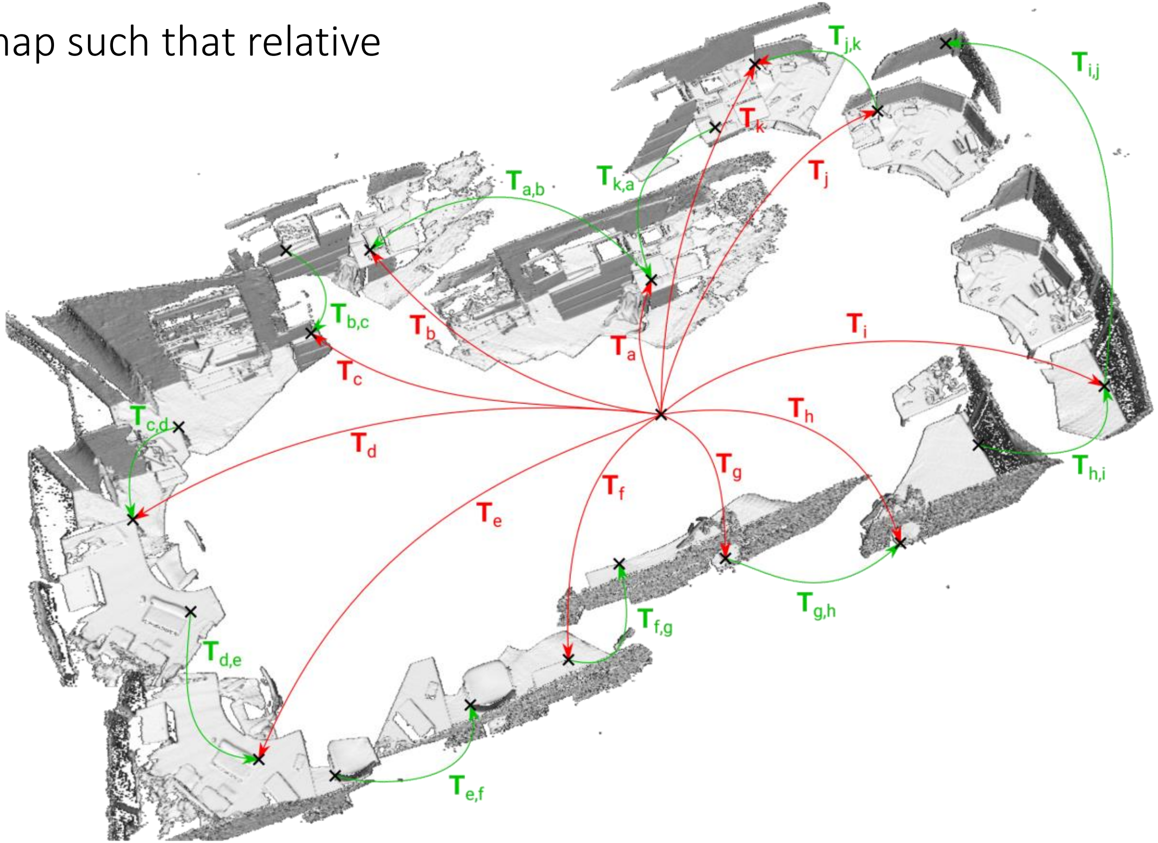


Pose Graph Optimisation

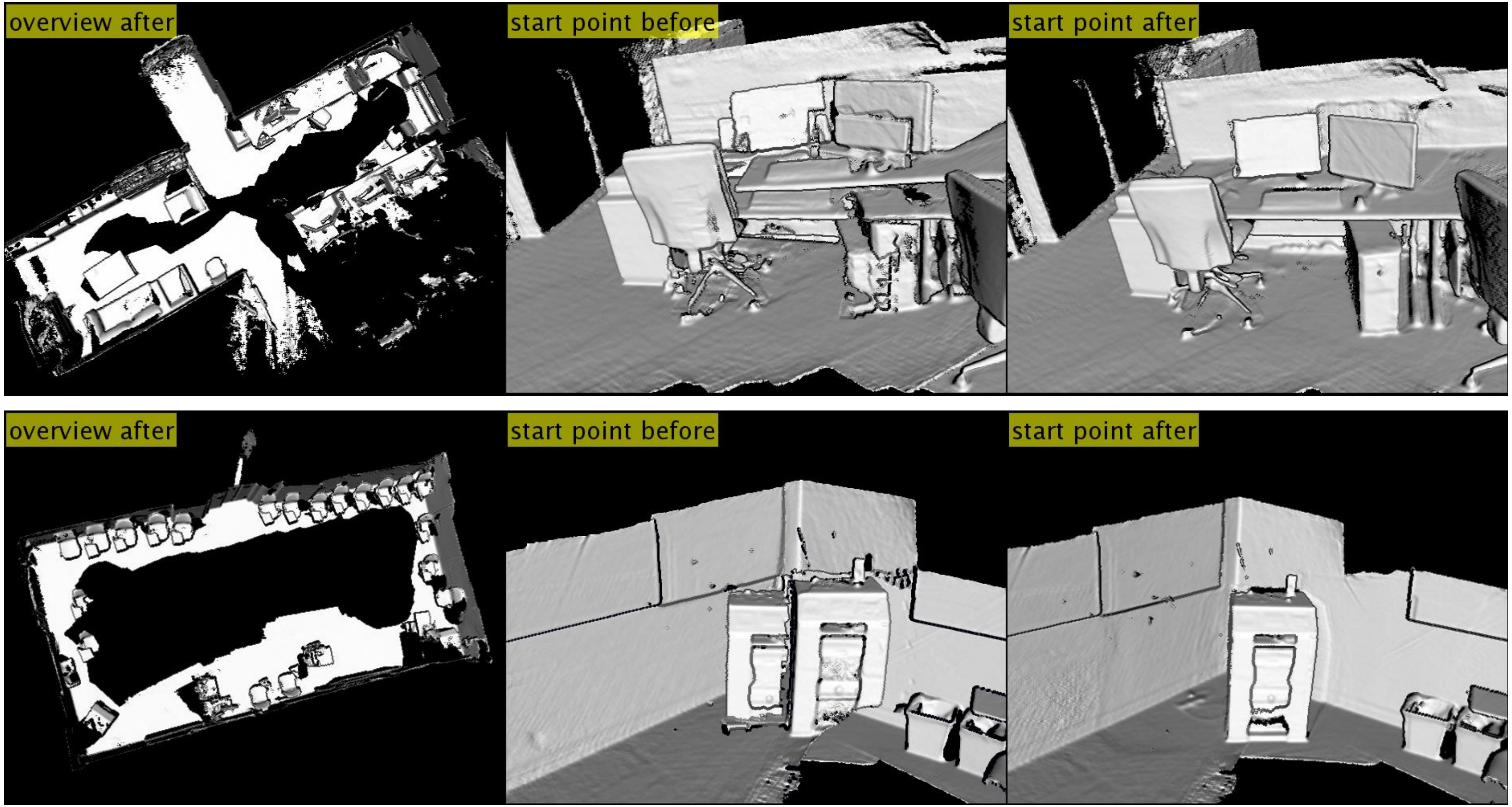
Find global pose for each submap such that relative constraints are satisfied.

$$\sum_{i,j} \left| \mathbf{v}(\mathbf{P}_\mathrm{i}, \mathbf{P}_\mathrm{j}, \mathbf{T}_\mathrm{i,j}) \right|$$

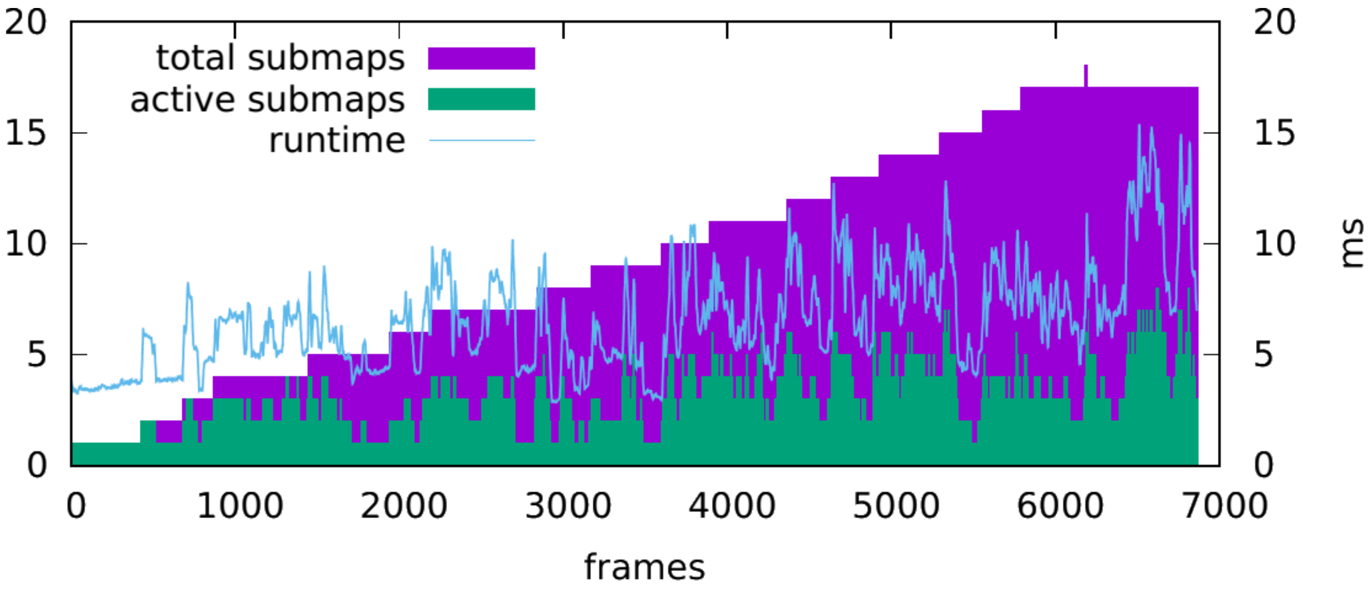
P pose of submap
T relative constraint
$\mathbf{v}(\mathbf{T}) = \left(\mathbf{q}(\mathbf{T}), \mathbf{t}(\mathbf{T})\right)^\mathrm{T}$



Result: Drift is compensated …



… and processing is still quick



- Processing time: **7.1 – 8.5 ms per frame**
- Remains constant

Depth cameras …

- Take space.
- Use lots of power.
- Do not work outside.

- We can do better ☺

6D.AI

Live Meshing Demo

DEMO

Underlying 3D Reconstruction

Active Vision
Laboratory

Underlying 3D Reconstruction

DEMO

Reconstructions can be very big …

DEMO

How does it work?

Unicorn magic ...
(and neural nets)

How do I use it?

- You'll need:
  - iOS 11.4+
  - Xcode 9.4.1 + ARKit 1.5+
  - Unity 2018.2+
  - iPhone 8 and higher.
- Sign-up and get username + SDK.
- Install the SDK in your app.
  - Standard drag and drop .framework on iOS.
  - Developer keys need to be specified in Info.plist.

How do I use it?

Init:

```
SixDegreesSDK_Initialize(EAGLContext*); // init with this
SixDegreesSDK_IsInitialized(); // wait until this returns true
```

Get pose:

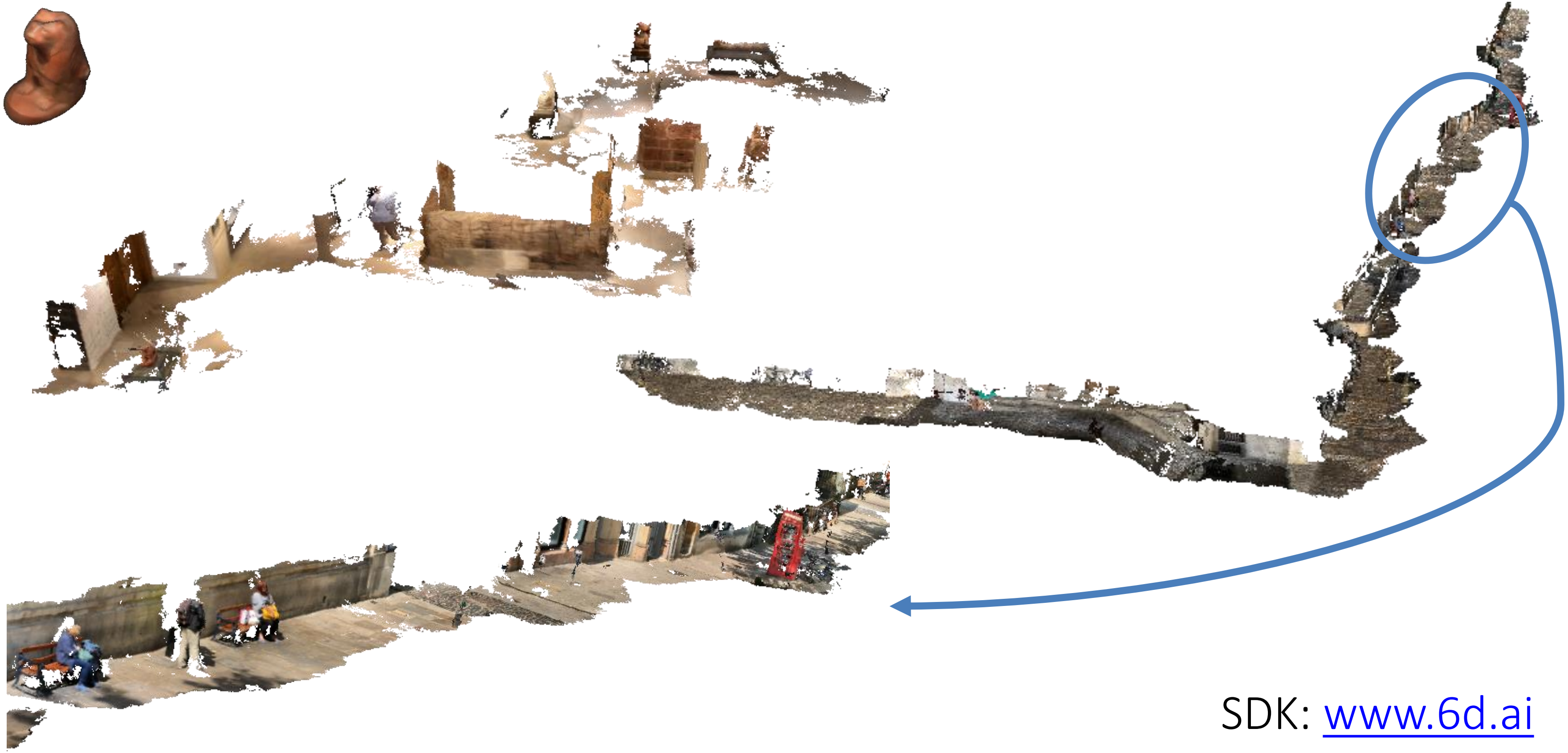
```
float pose[16];
SixDegreesSDK_GetPose(pose, 16); // get the pose here!
```

Get mesh:

```
int blockBufferSize, vertexBufferSize, faceBufferSize;
SixDegreesSDK_GetMeshBlockInfo(&blockBufferSize, &vertexBufferSize, &faceBufferSize); // gets the live mesh
info

int blockBuffer[blockBufferSize];
float vertexBuffer[vertexBufferSize];
int faceBuffer[faceBufferSize];
SixDegreesSDK_GetMeshBlocks(blockBuffer, vertexBuffer, faceBuffer, blockBufferSize, vertexBufferSize,
faceBufferSize); // gets the live mesh
```

Conclusion



SDK: www.6d.ai