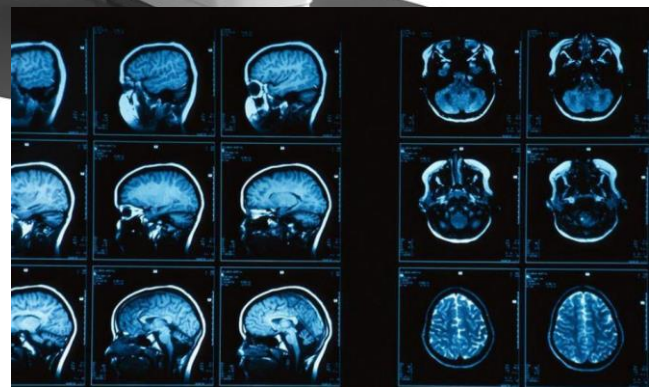
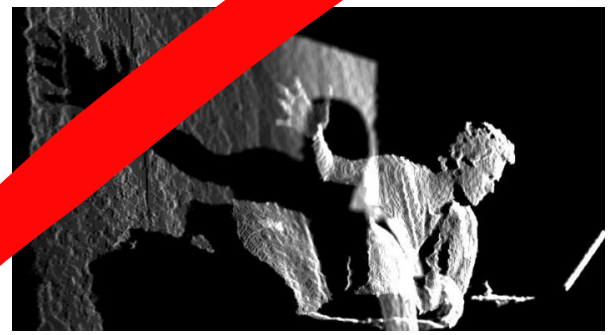
3D Scanning & Motion Capture

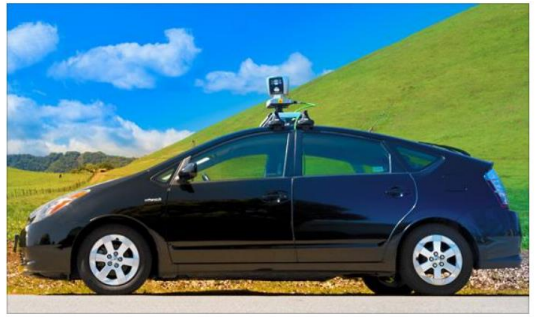
Rigid Surface Tracking & Reconstruction

Dr. Justus Thies, **Dr. Angela Dai**

Last Lecture: How to obtain “3D”?

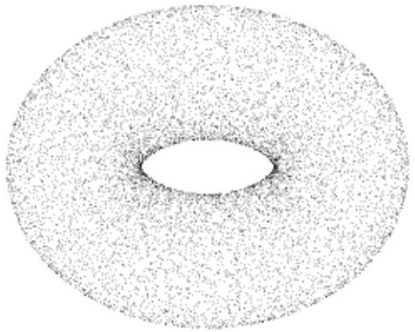




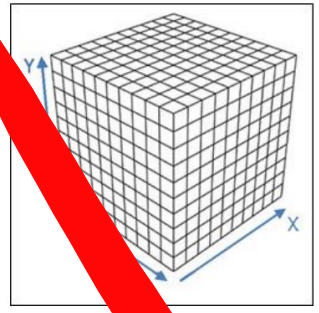


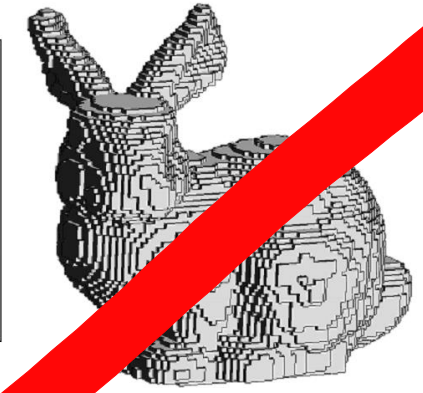
Last Lecture: Surface Representations

- Point Clouds

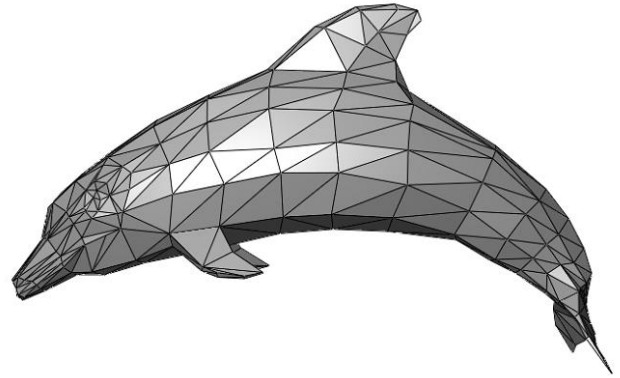


- Voxels

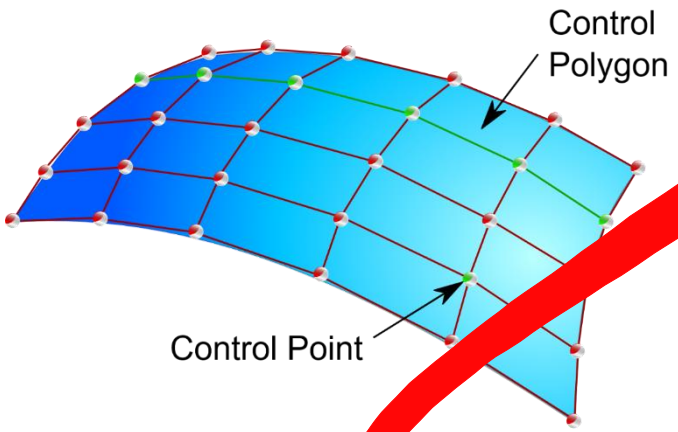




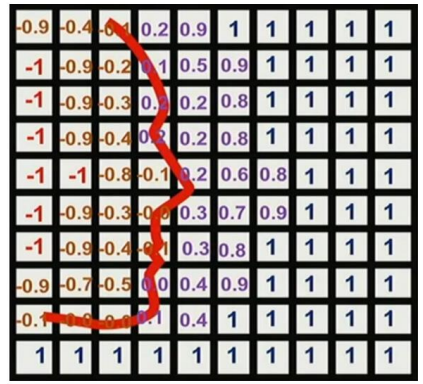
- Polygonal Meshes



- Parametric Surfaces



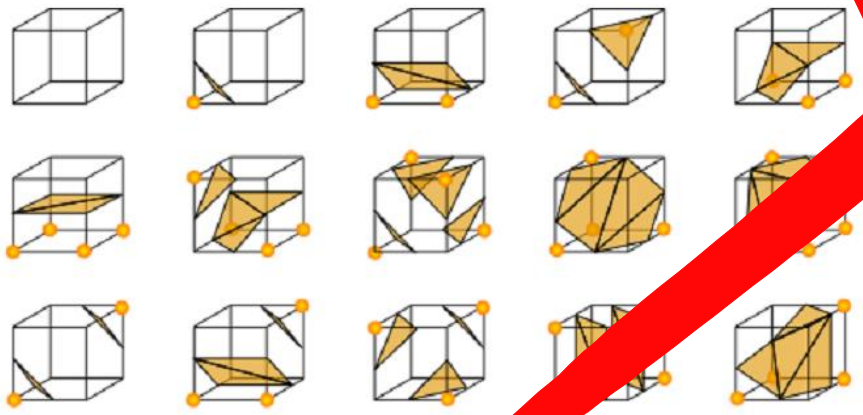
- Implicit Surfaces



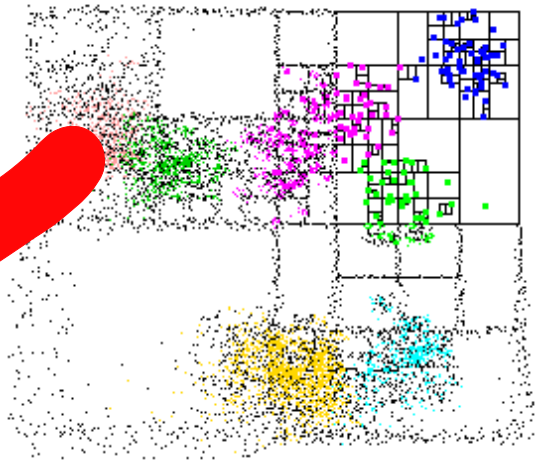
Truncated Signed Distance Field (TSDF)

Last Lecture: Surface Representations

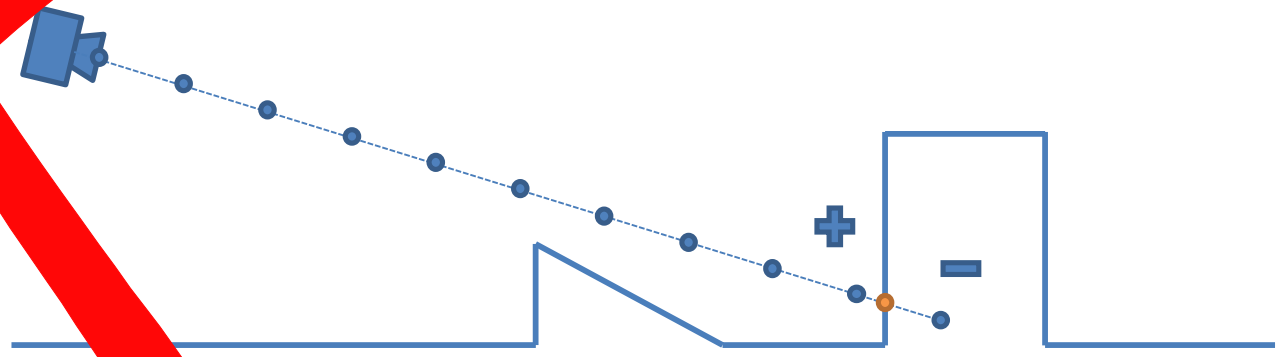
- Important Algorithms
  - Marching Cubes
  - Ray cast
  - Nearest neighbor lookups



Marching Cube table



ANN



Ray Casting

Last Lecture: Correspondence Finding / Matching

Last Lecture: Correspondence Finding / Matching

Last Lecture: Bundle Adjustment (SfM)

- m images
- n points in 3d

- $E_{re-proj}(\boldsymbol{T}, \boldsymbol{X}) =$

$$\sum_{i=1}^{m} \sum_{j=1}^{n} \left|\left| x_{ij} - \pi_i(T_i \cdot X_j) \right|\right|^2$$

over images

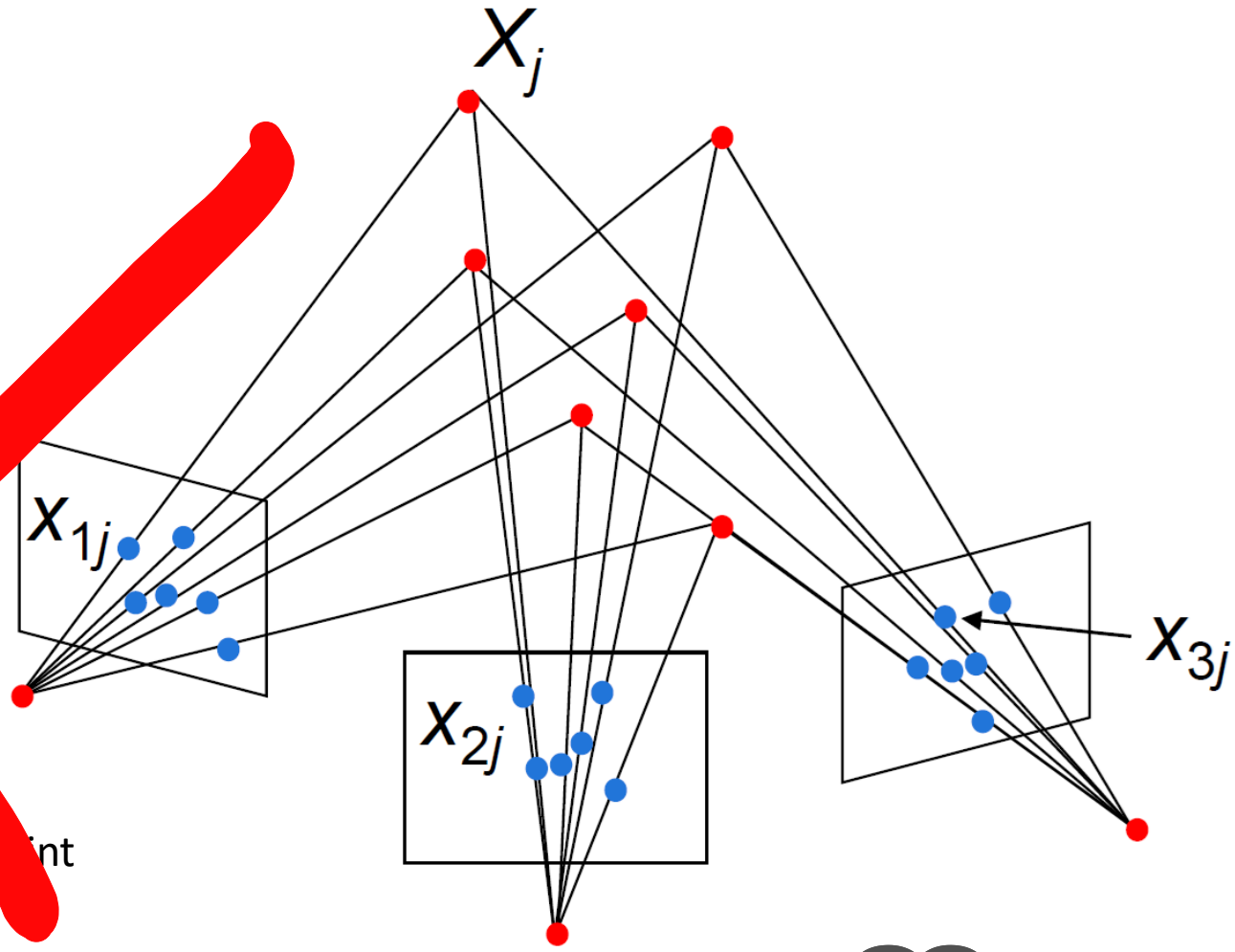
over 3d points

2d keypoint locations

3D-2D proj (intrinsics)

frame pose (extrinsics)

3D point



Last Lecture: RGB-D “Bundling”

$$E_{bundle}(T) = \sum_{i,j}^{\#frames} \sum_{k}^{\#corresp.} \left\| T_i p_{ik} - T_j p_{jk} \right\|_2^2$$

$$E_{depth}(T) = \sum_{i,j}^{\#frames} \sum_{k}^{\#pixels} \left\| \left(p_k - T_i^{-1} T_j \begin{pmatrix} u \\ v \\ d \end{pmatrix}^{-1} \left(D_j(\pi_d(T_j^{-1} T_i p_k)) \right) \right) \cdot n_k \right\|_2^2$$

$$E_{color}(T) = \sum_{i,j}^{\#frames} \sum_{k}^{\#pixels} \left\| \nabla I(\pi_c(p_k)) - \nabla I(\pi_c(T_j^{-1} T_i p_k)) \right\|_2^2$$

Last Lecture: How do we solve these objectives?

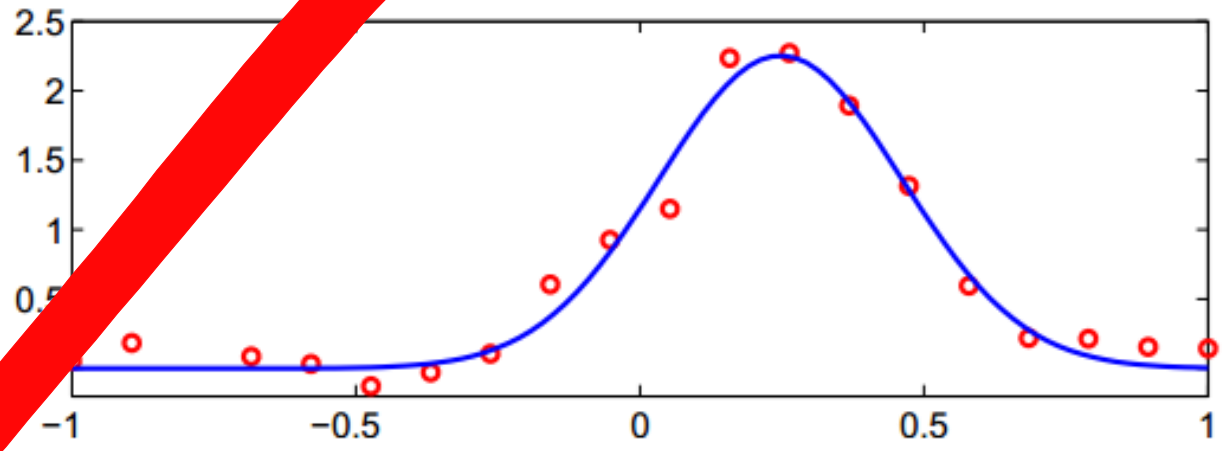
- Bundle Adjustment or RGB-D Bundling

$$E_{re-proj}(\boldsymbol{T}, \boldsymbol{X}) = \sum_{i=1}^{m} \sum_{j=1}^{n} \left|\left| x_{ij} - \pi_i(T_i \cdot X_j) \right|\right|_2^2$$

$$E_{keypoint}(T) = \sum_{i,j}^{\#\text{features corresp.}} \sum_{k} \left\| T_i p_{ik} - T_j p_{jk} \right\|_2^2$$

Last Lecture: Non-Linear Least Squares

- Find solution that minimizes the sum of squared residuals
  - $f(x) = \sum r_i(x)^2$
  - r_i non linear with respect to x
- Fitting a Gaussian model



- $M(x,t) = x_1 e^{-(t-x_2)^2/(2x_3)^2}$, $x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$
- Here: $r_i(x) = y_i - M(x, t_i)$

Last Lecture: NNLS -> Gauss-Newton and LM

- Newton's Method:

$$x_{k+1} = x_k - H_f(x_k)^{-1}\nabla f(x$$

- Gauss-Newton (GN):

$$x_{k+1} = x_k - [2J_F(x_k)^T J_F(x_k)]^{-1}\nabla f(x_k)$$

Solve linear system (again inverting a matrix is unstable):

$$2\left(J_F(x_k)^T J_F(x_k)\right)(x_k - x_{k+1}) = \nabla f(x_k)$$

Solve for delta vector

- Levenberg-Marquardt (LM)

$$\left(J_F(x_k)^T J_F(x_k) + \lambda \cdot diag(J_F(x_k)^T J_F(x_k))\right) \cdot (x_k - x_{k+1}) = \nabla f(x_k)$$

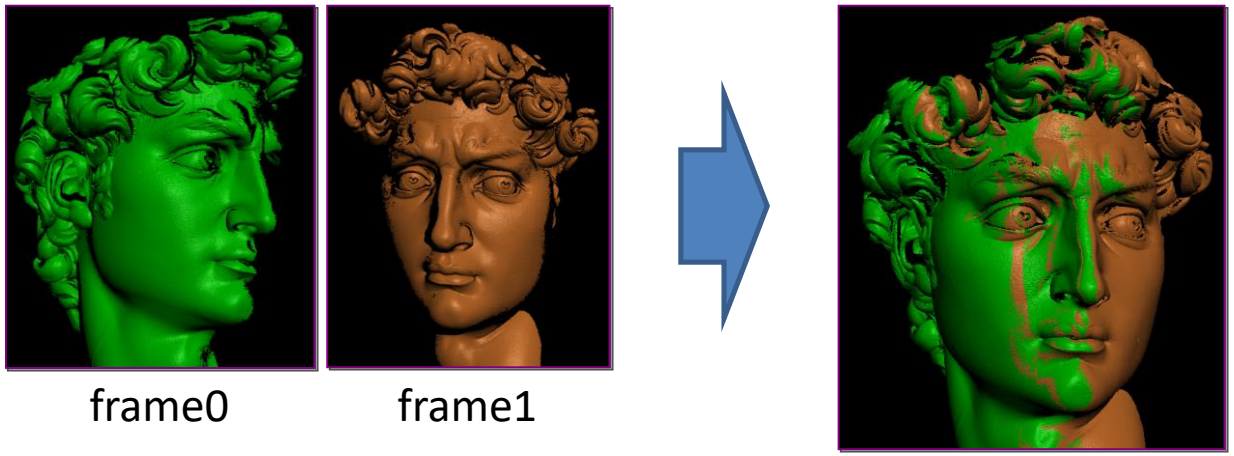
Today: Rigid Surface Tracking & Reconstruction

How do we solve these non-linear terms?

- Frame-to-frame alignment (RGB-D case)
- $E_{frame-to-frame}(T) = \sum \|p_k - Tq_k\|_2^2$
- How to align two RGB-D frames?
  - Iterative Closest Points (ICP)!

Problem Statement: RGB-D Tracking

- Given RGB-D camera, move it over time and we want to align it
- Analog statement: given two point clouds, find rigid transformation between them
- Think about minimizing the distance between two implicit functions; ICP approximates functions



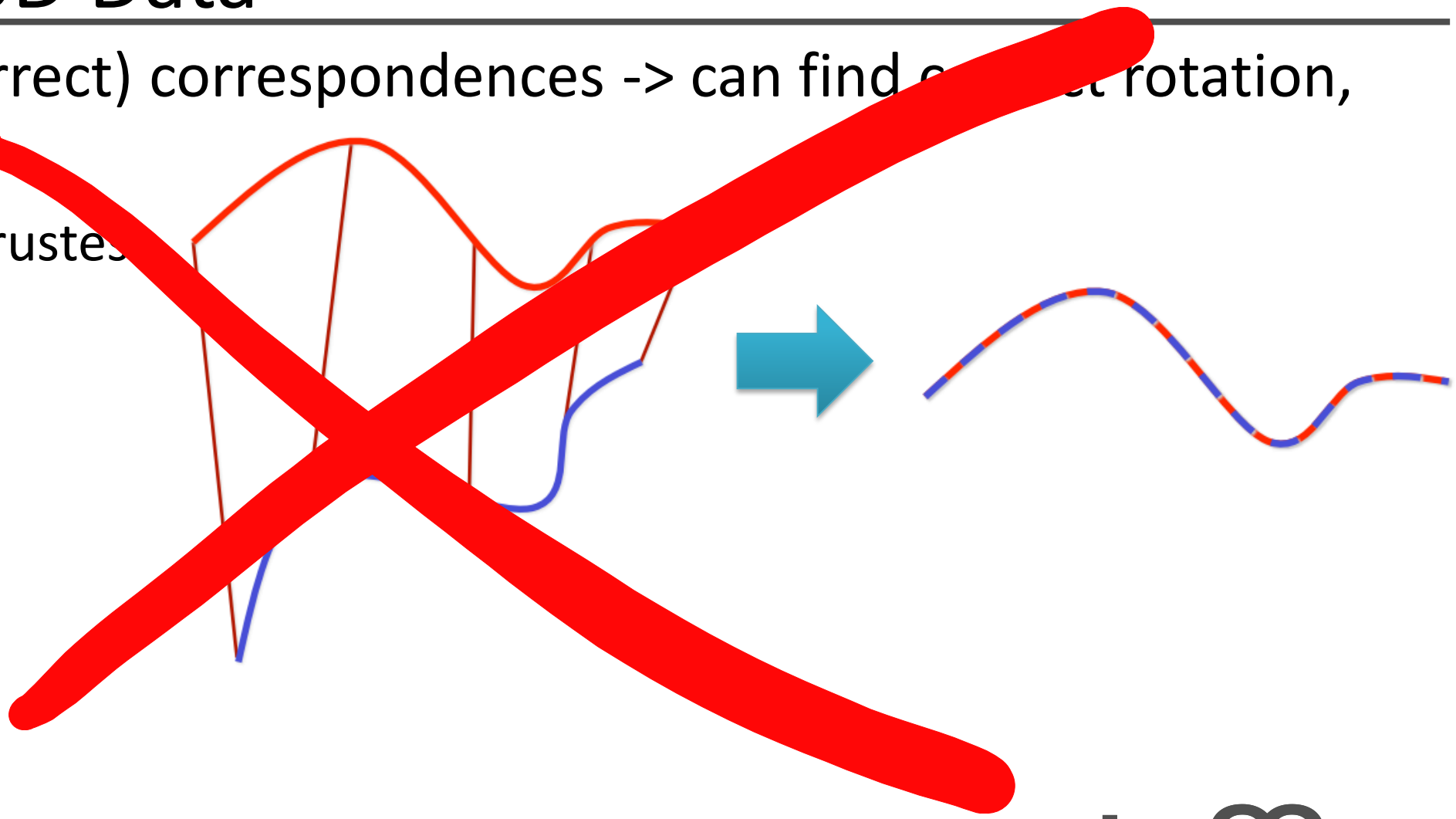
frame0 frame1

High-Level ICP Overview

- Data association (Correspondence Search)
- Outlier removal / pruning
- Optimization
- Iterate

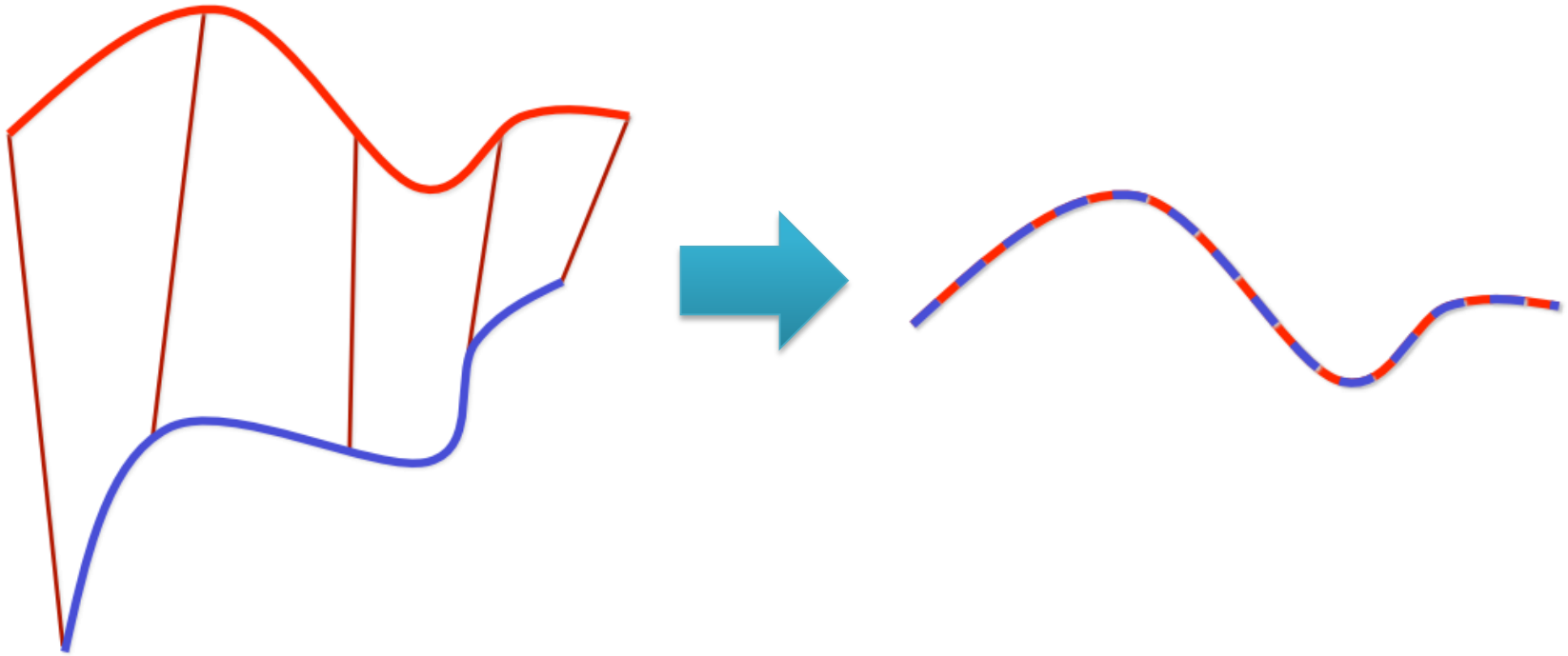
Aligning 3D Data

- Known (correct) correspondences -> can find c… rotation, translation
  - E.g., Procrustes



Aligning 3D Data

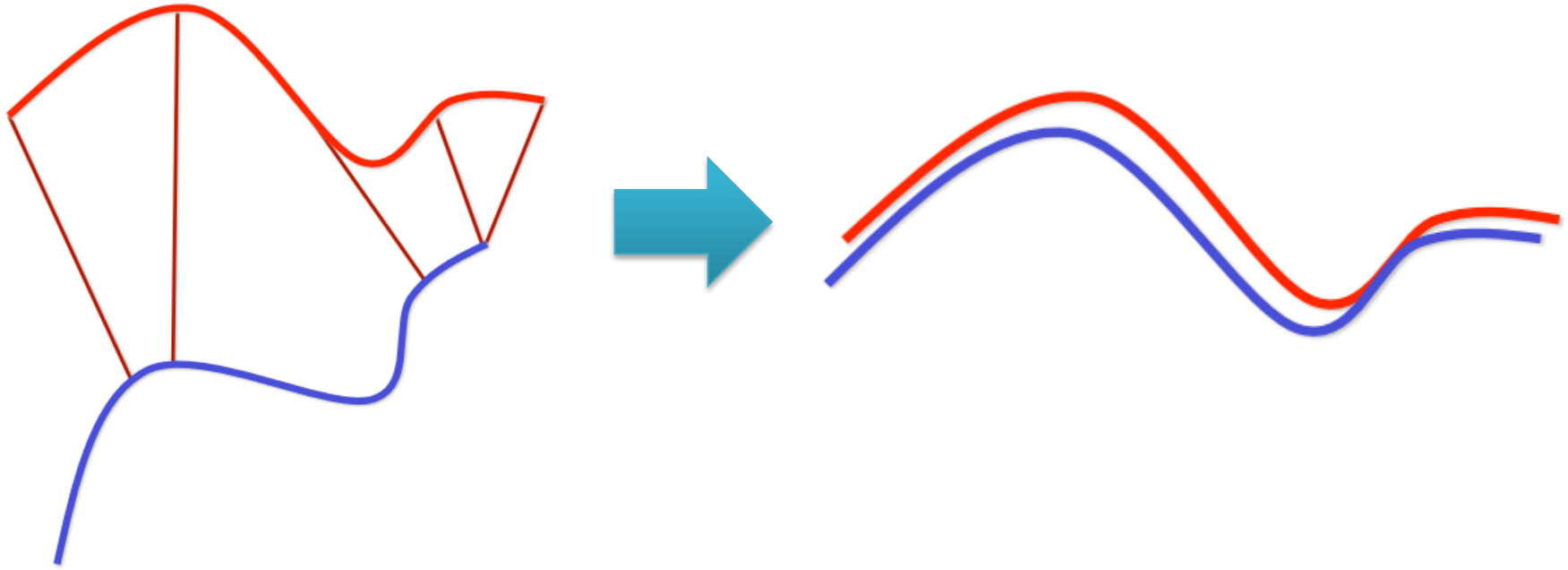
- Known (correct) correspondences -> can find correct rotation, translation
  - E.g., Procrustes



- How to find correspondences?

Iterative Closest Points (ICP) *[Besl and McKay 92]*

- Assume closest points are corresponding



Iterative Closest Points (ICP) *[Besl and McKay 92]*

- Iterate to find alignment
- Converge if poses are “close enough”

Basic ICP

- **Select** (e.g., 1000) random points
- **Match** each point to closest point on other scan
  - using efficient search structure, e.g., kd-tree
  - various libraries: ANN, FLANN, nanoflann
- **Reject** outlier pairs with distance $> k \times$ median
- **Error Function**:

$$E := \sum_i (R\mathbf{p}_i + t - \mathbf{q}_i)^2$$

- **Minimize** (closed form solution in [Horn 87])

Variants of Basic ICP

- **Selecting** source points
- **Matching** to points in other scan
- **Weighting** the correspondences
- **Rejecting** outlier point pairs
- Assigning an **error metric** to the current transform
- **Minimizing** the error metric w.r.t transformation

Performance of ICP Variants

- Analyze:
  - Speed
  - Stability
  - Tolerance to noise and/or outliers
  - Maximum initial misalignment
- Comparison of many ICP variants:
  - [Rusinkiewicz and Levoy, 3DIM, 2001]

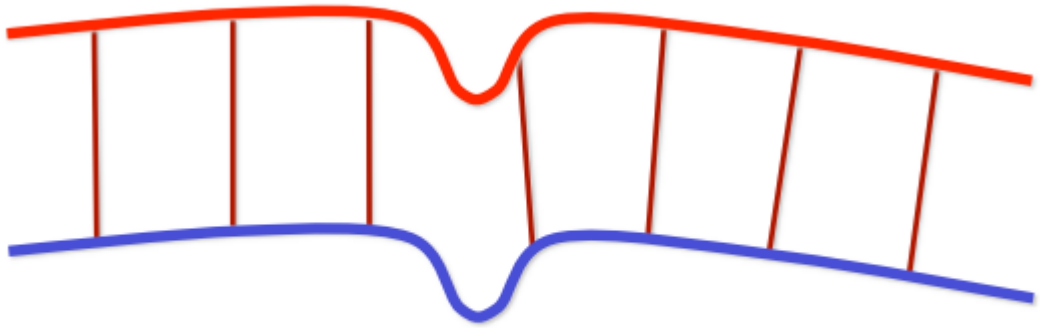
Variants of Basic ICP

- **Selecting** source points
- **Matching** to points in other scan
- **Weighting** the correspondences
- **Rejecting** outlier point pairs
- Assigning an **error metric** to the current transform
- **Minimizing** the error metric w.r.t transformation

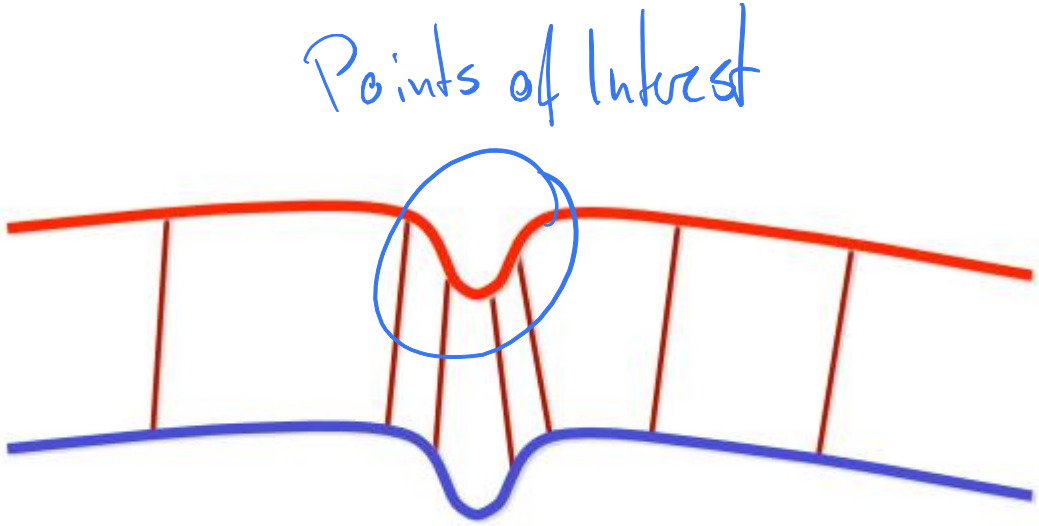
Selecting Source Points

- Use all points → Only for small #
- Uniform subsampling → ↯ Aliasing
- Random sampling
- Stable sampling [Gelfand et al. 2003]
  - Select samples that constrain all degrees of freedom of the rigid-body transformation

Stable Sampling



Uniform Sampling



Stable Sampling

Variants of Basic ICP

- **Selecting** source points
- **Matching** to points in other scan
- **Weighting** the correspondences
- **Rejecting** outlier point pairs
- Assigning an **error metric** to the current transform
- **Minimizing** the error metric w.r.t transformation

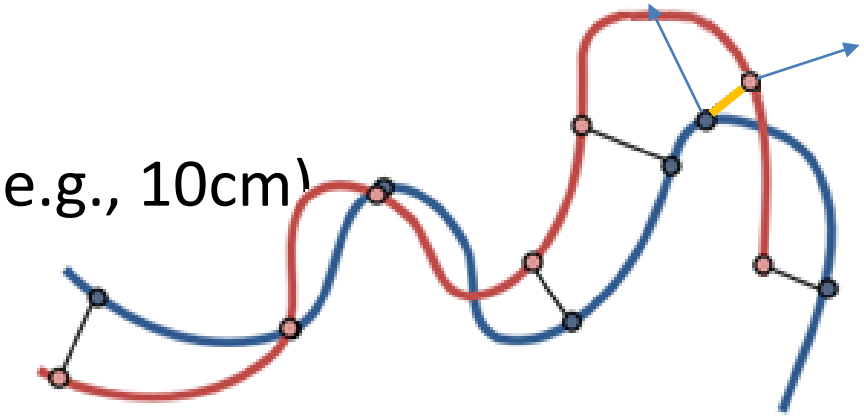
Closest Compatible Point

- Closest points are often bad as corresponding points
- Improve matching effectiveness by restricting to **compatible** points
  - Color compatibility [Godin et al. 94]
  - Normal compatibility [Pulli 99]
  - Other possibilities: curvature, higher-order derivatives, other local features

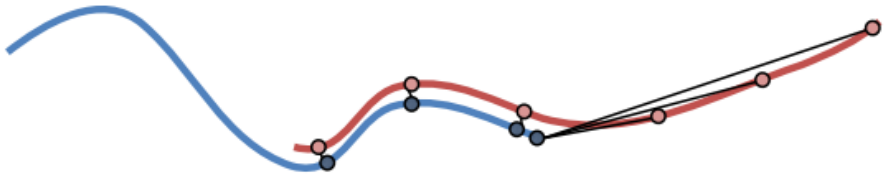
ICP: Data Association

- Nearest neighbor (use ANN, FLANN, nanoflann)
- Projective (often much better and faster!!)
  - Only works with structured data; e.g., depth images (not on raw point clouds)

- Rejection based on outliers
  - Distance between depth values > threshold (e.g., 10cm)
  - Variation between normals



- Pruning
  - Prune correspondences to borders

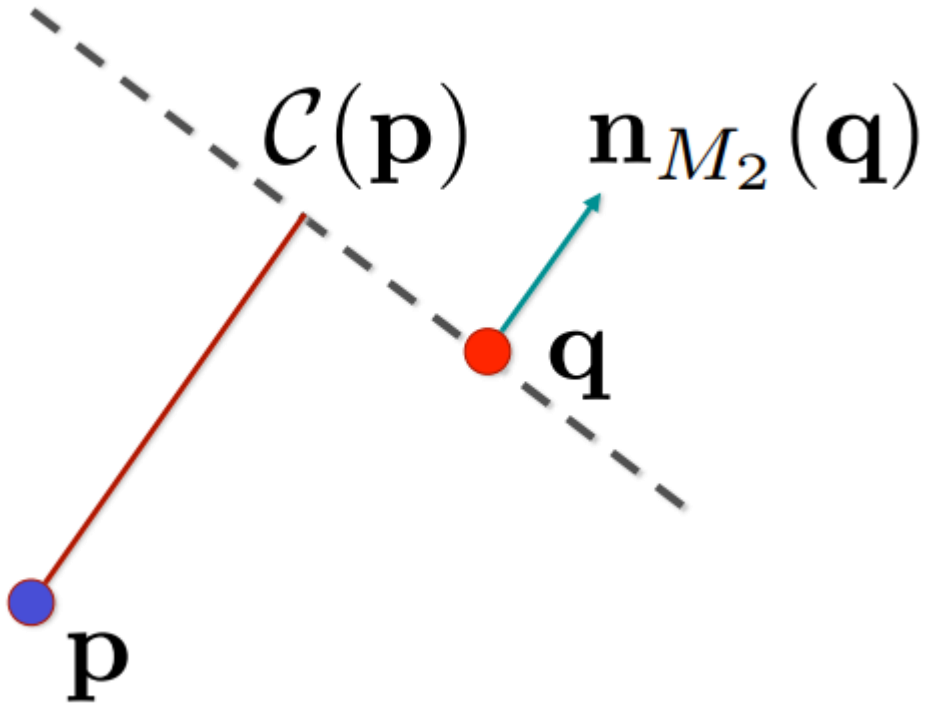


Variants of Basic ICP

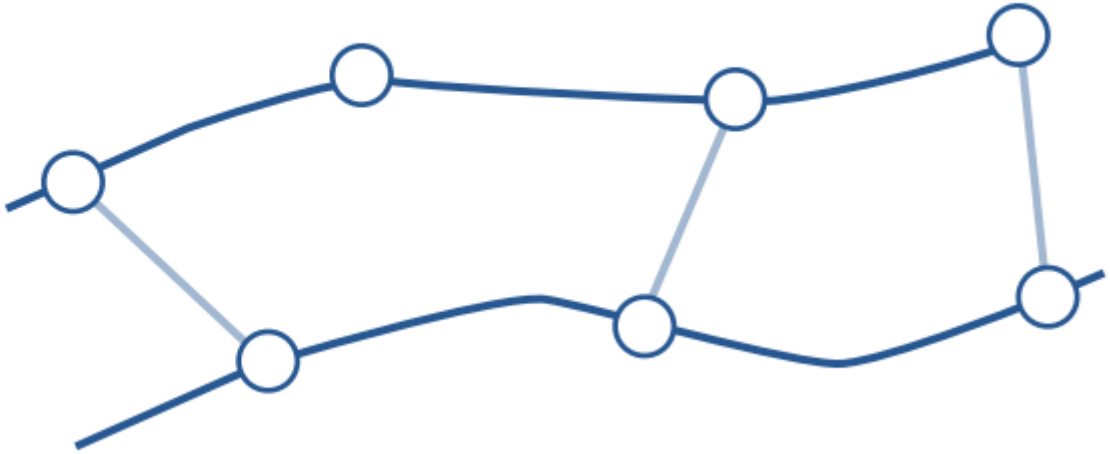
- **Selecting** source points
- **Matching** to points in other scan
- **Weighting** the correspondences
- **Rejecting** outlier point pairs
- Assigning an **error metric** to the current transform
- **Minimizing** the error metric w.r.t transformation

Point-to-Plane Error Metric

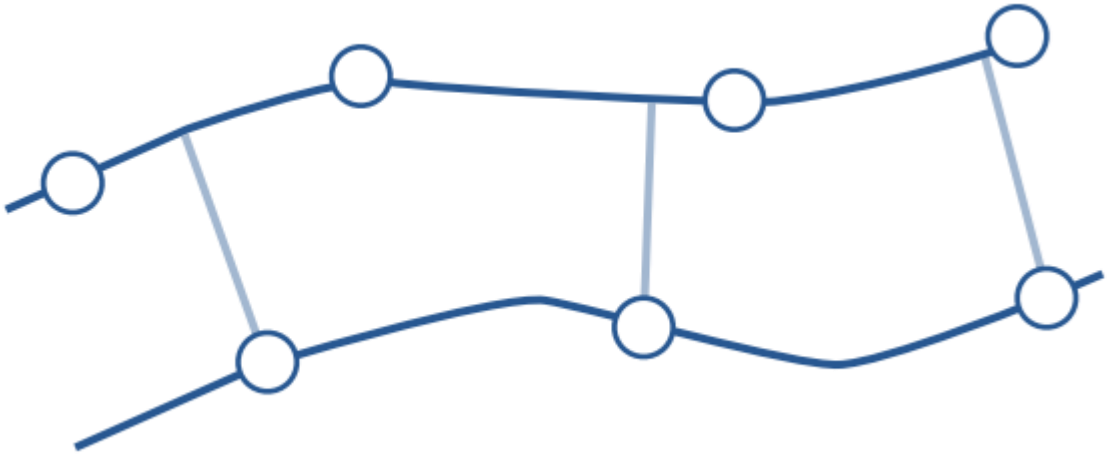
- Allows flat regions to slide along each other, in contrast to point-to-point distance



Point-to-Plane Error Metric



Point-to-Point



Point-to-Plane

Variants of Basic ICP

- **Selecting** source points
- **Matching** to points in other scan
- **Weighting** the correspondences
- **Rejecting** outlier point pairs
- Assigning an **error metric** to the current transform
- **Minimizing** the error metric w.r.t transformation

Point-to-Plane Error Metric - Optimization

- Error function:

$$E := \sum_i \left((R\mathbf{p}_i + t - \mathbf{q}_i) \cdot \mathbf{n}_i \right)^2$$

where R is a rotation matrix, t is a translation vector

Camera frames only small differences!

- Linearize (i.e., assume $\sin\theta \approx \theta, \cos\theta \approx 1$):

$$E \approx \sum ((\mathbf{p}_i - \mathbf{q}_i) \cdot \mathbf{n}_i + \mathbf{r} \cdot (\mathbf{p}_i \times \mathbf{n}_i) + \mathbf{t} \cdot \mathbf{n}_i)^2 \quad \text{where } \mathbf{r} = \begin{pmatrix} r_x \\ r_y \\ r_z \end{pmatrix}$$

Point-to-Plane Error Metric - Optimization

- Overconstrained linear system

$$\mathbf{A}x = b,$$

$$\mathbf{A} = \begin{pmatrix} \leftarrow & \mathbf{p}_1 \times \mathbf{n}_1 & \rightarrow & \leftarrow & \mathbf{n}_1 & \rightarrow \\ \leftarrow & \mathbf{p}_2 \times \mathbf{n}_2 & \rightarrow & \leftarrow & \mathbf{n}_2 & \rightarrow \\ & \vdots & & & \vdots & \end{pmatrix}, x = \begin{pmatrix} r_x \\ r_y \\ r_z \\ t_x \\ t_y \\ t_z \end{pmatrix}, b = \begin{pmatrix} -(\mathbf{p}_1 - \mathbf{q}_1) \cdot \mathbf{n}_1 \\ -(\mathbf{p}_2 - \mathbf{q}_2) \cdot \mathbf{n}_2 \\ \vdots \end{pmatrix}$$

- Solve using least squares

$$\mathbf{A}^T\mathbf{A}x = \mathbf{A}^T b$$

Point-to-Plane Error Metric - Optimization

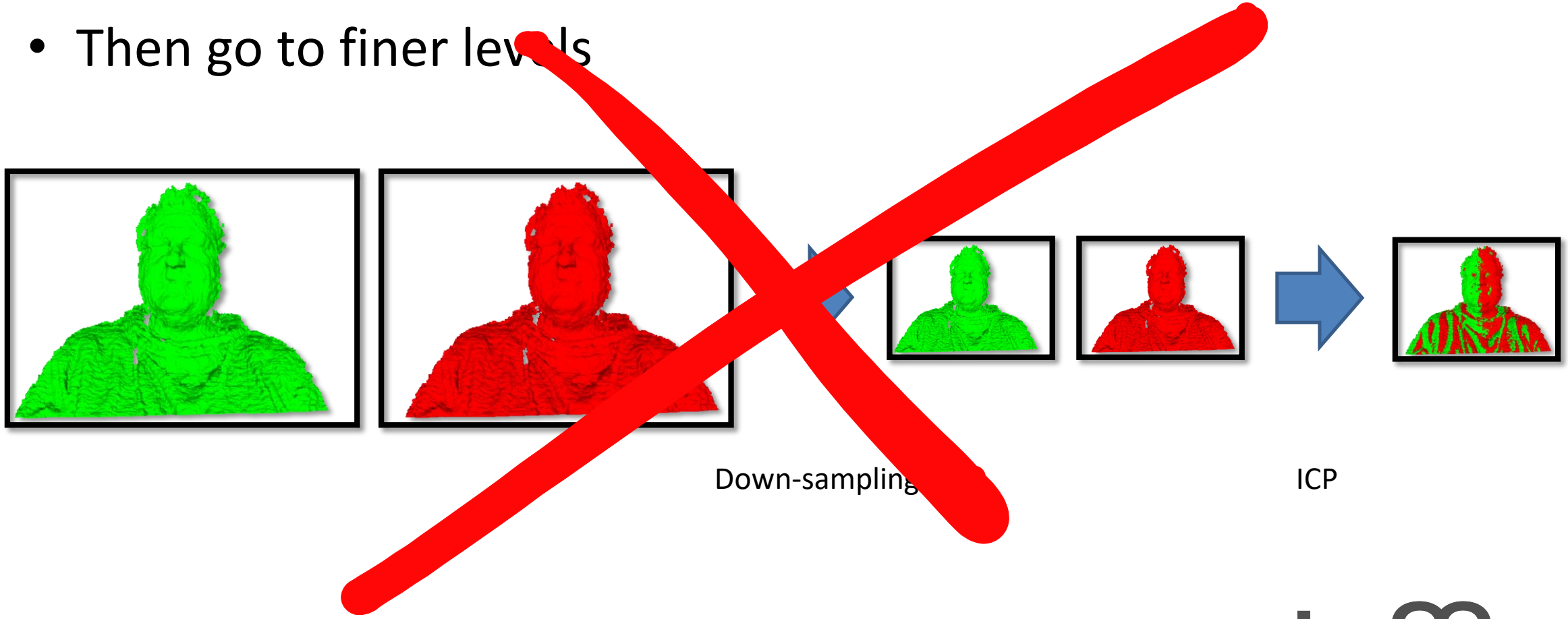
- Linearized Point-to-Plane Optimization [Low]
  - Only valid for small rotations (linear approximation of the rotation!)

- If you don't know what to do, always use Levenberg-Marquardt!
  - 2nd order method → **quadratic convergences** -> rules them all

$$E_{frame-to-frame}(T) = \sum_i [(Tp_i - q_i) \cdot n_i]^2$$

Coarse-to-fine Alignment

- Start on low-resolution depth map
- Then go to finer levels



Coarse-to-fine Alignment

- Start on low-resolution depth map
- Then go to finer levels

Propagate solution
to finer level

ICP

Coarse-to-fine Alignment

- Start on low-resolution depth map
- Then go to finer levels

➢ Faster convergence

➢ Avoids local minima (down-sampling → smoothing)

➢ Allows faster motions

ICP: Conclusion

- Because the correspondence finding is not differentiable, there is the hack of alternating
- Ideally, you would want to directly align a continuous surface, and differentiate it

- Many ICP variants (e.g., adding a color term)
  - Can also do similar algorithm for RGB-only alignment

3D Reconstruction

1. Compute poses between all frames

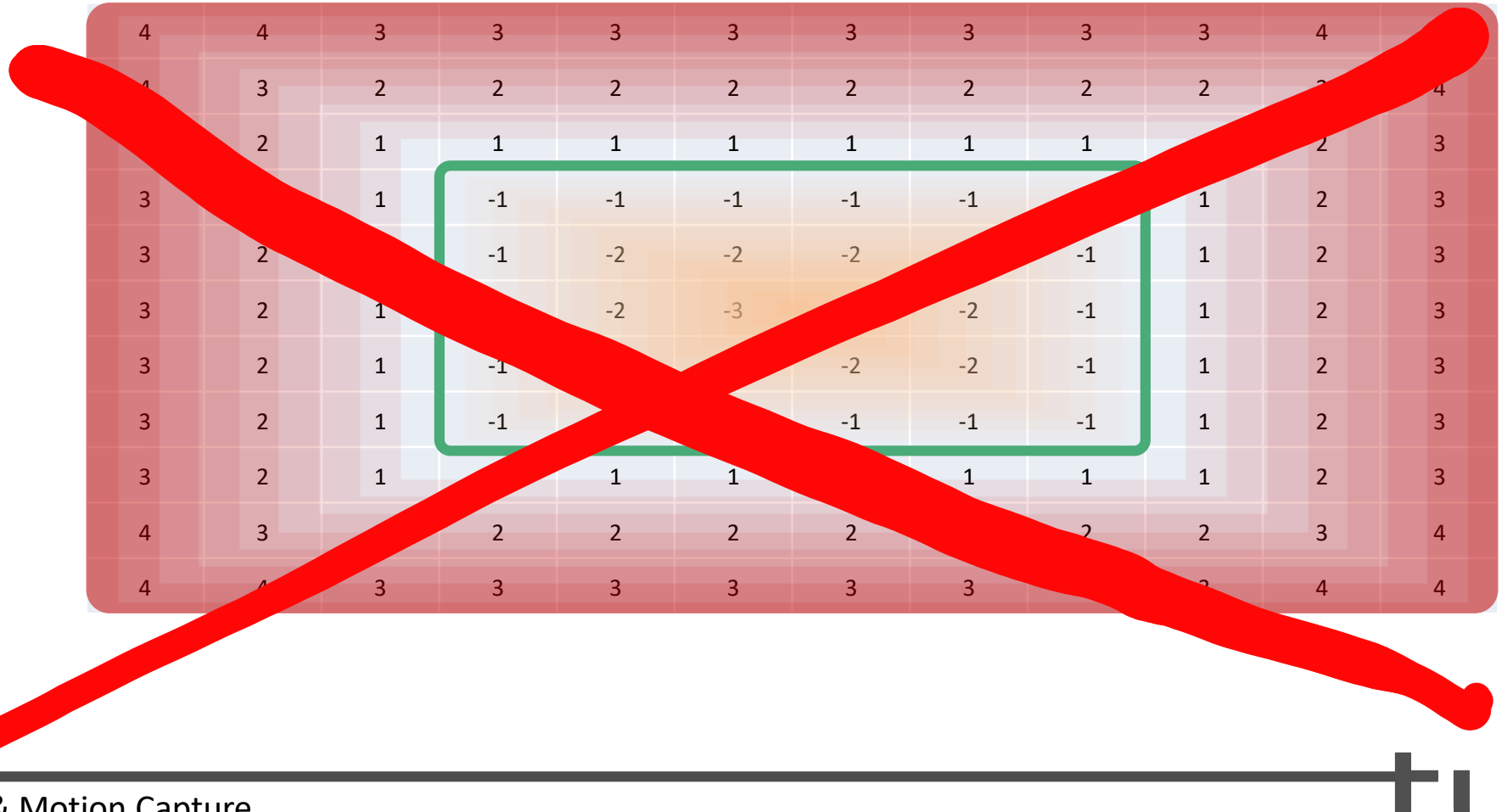
   e.g., frame-to-frame ICP, or frame-to-model ICP

2. Integrate data into a shared model

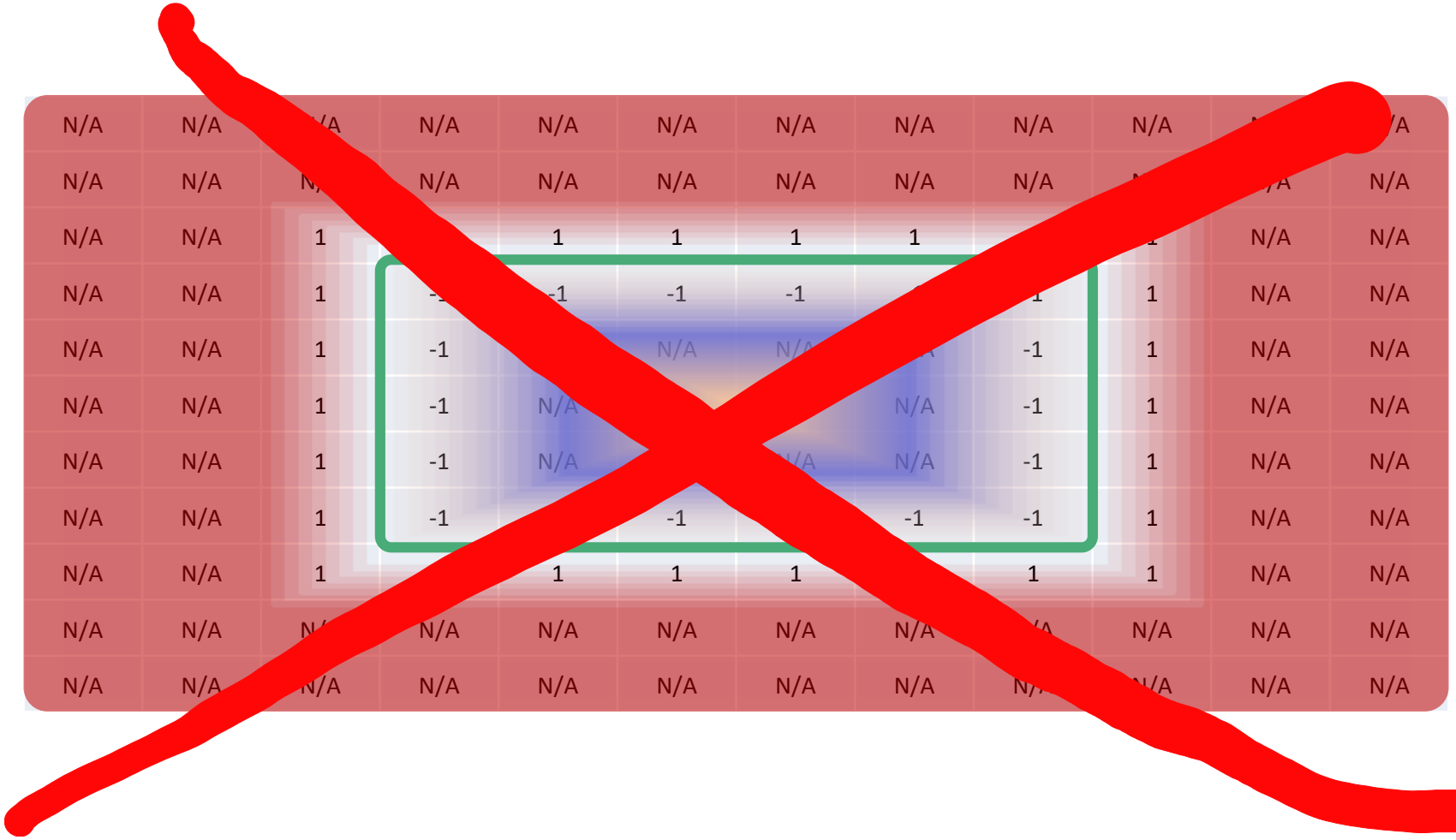
   e.g., accumulate in a distance field

3. Extract Mesh from implicit representation

Signed Distance Field (SDF)

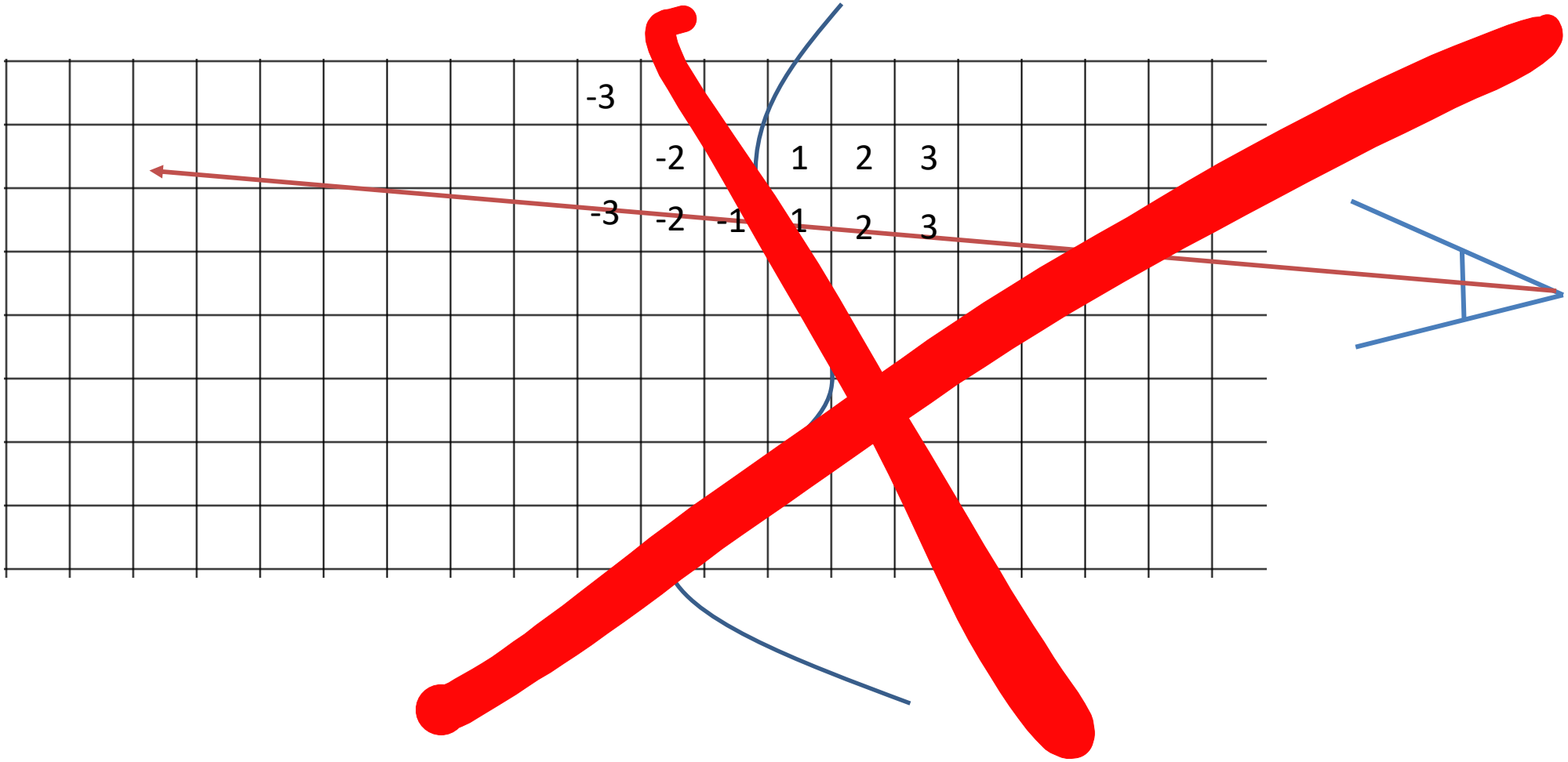


Truncated Signed Distance Field (TSDF)

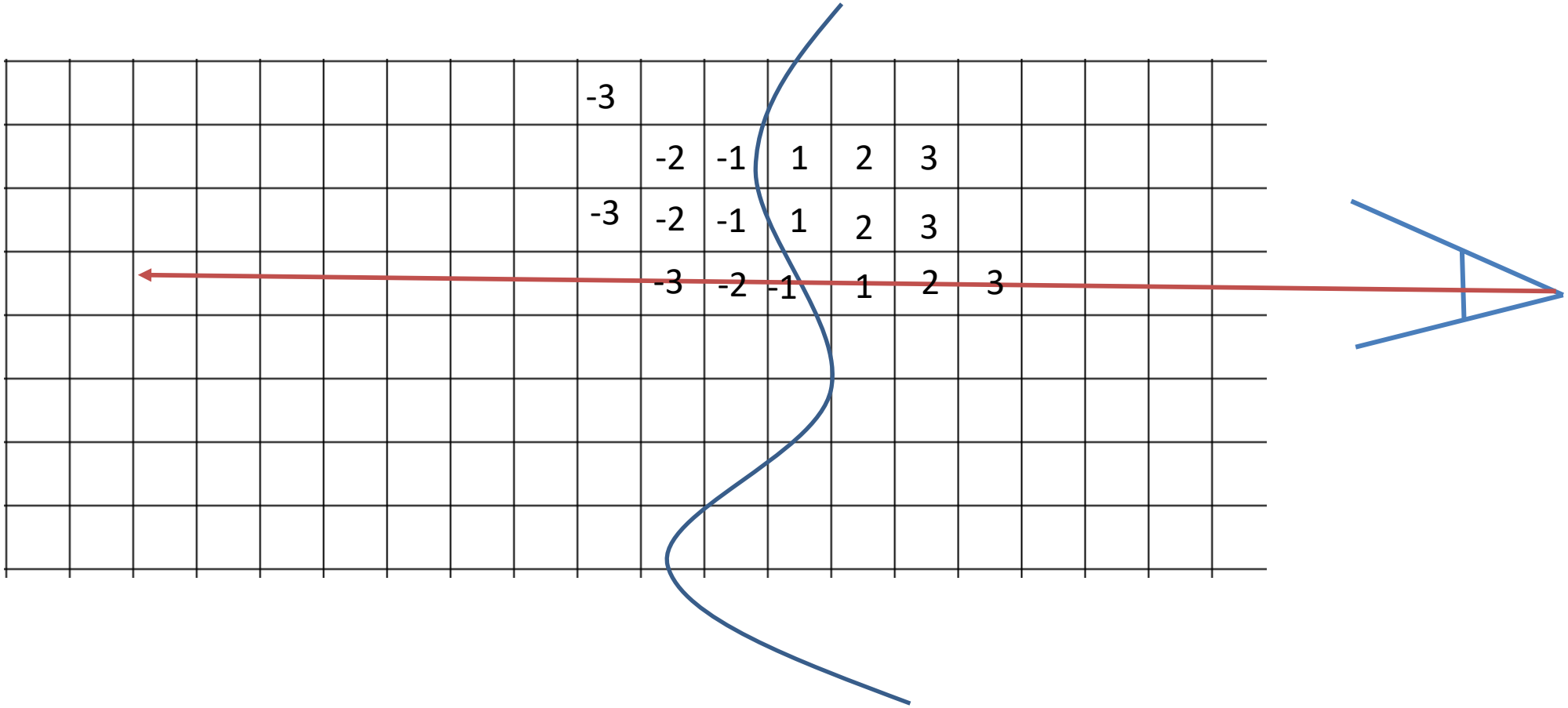


Volumetric Fusion: Surface Integration

Volumetric Fusion: Surface Integration

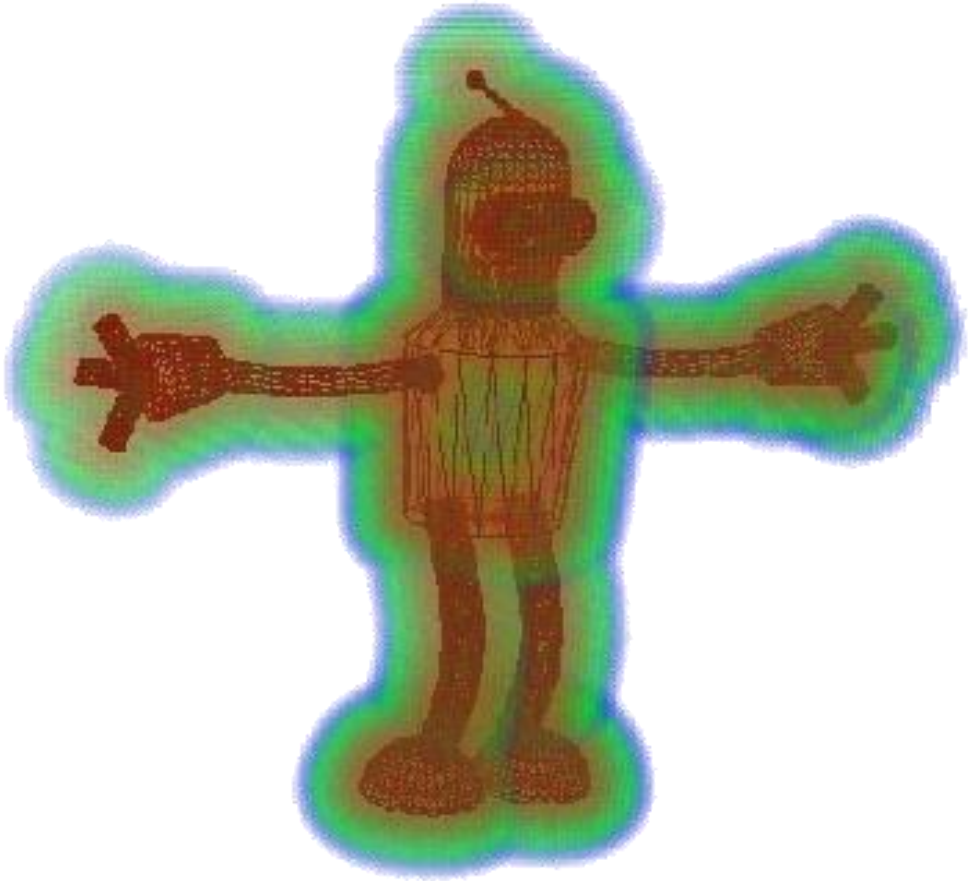


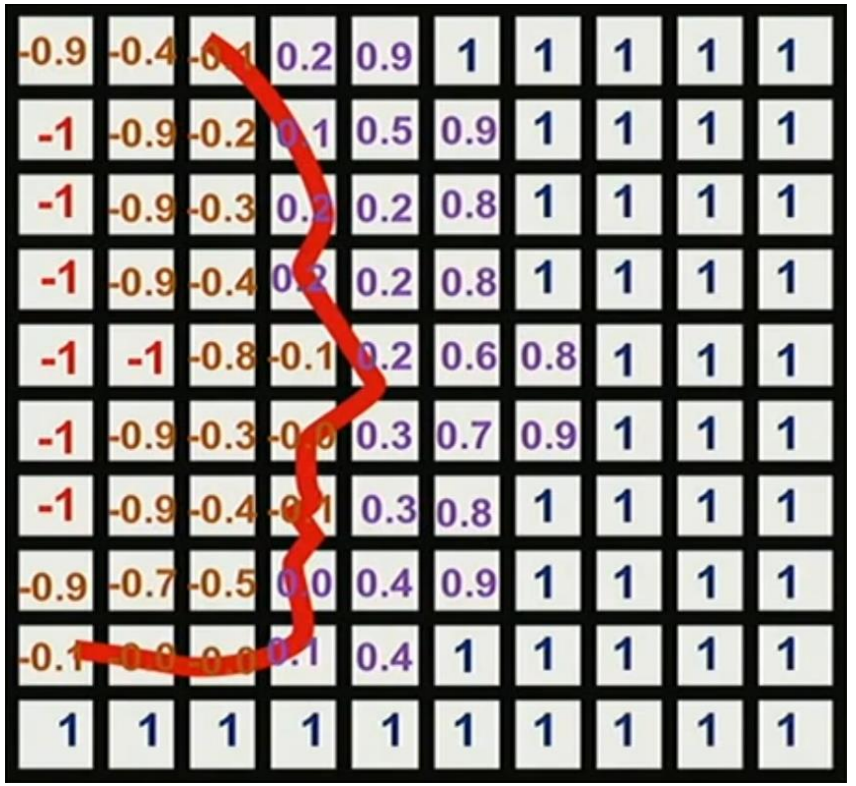
Volumetric Fusion: Surface Integration



Obviously SDF calculation is in floats

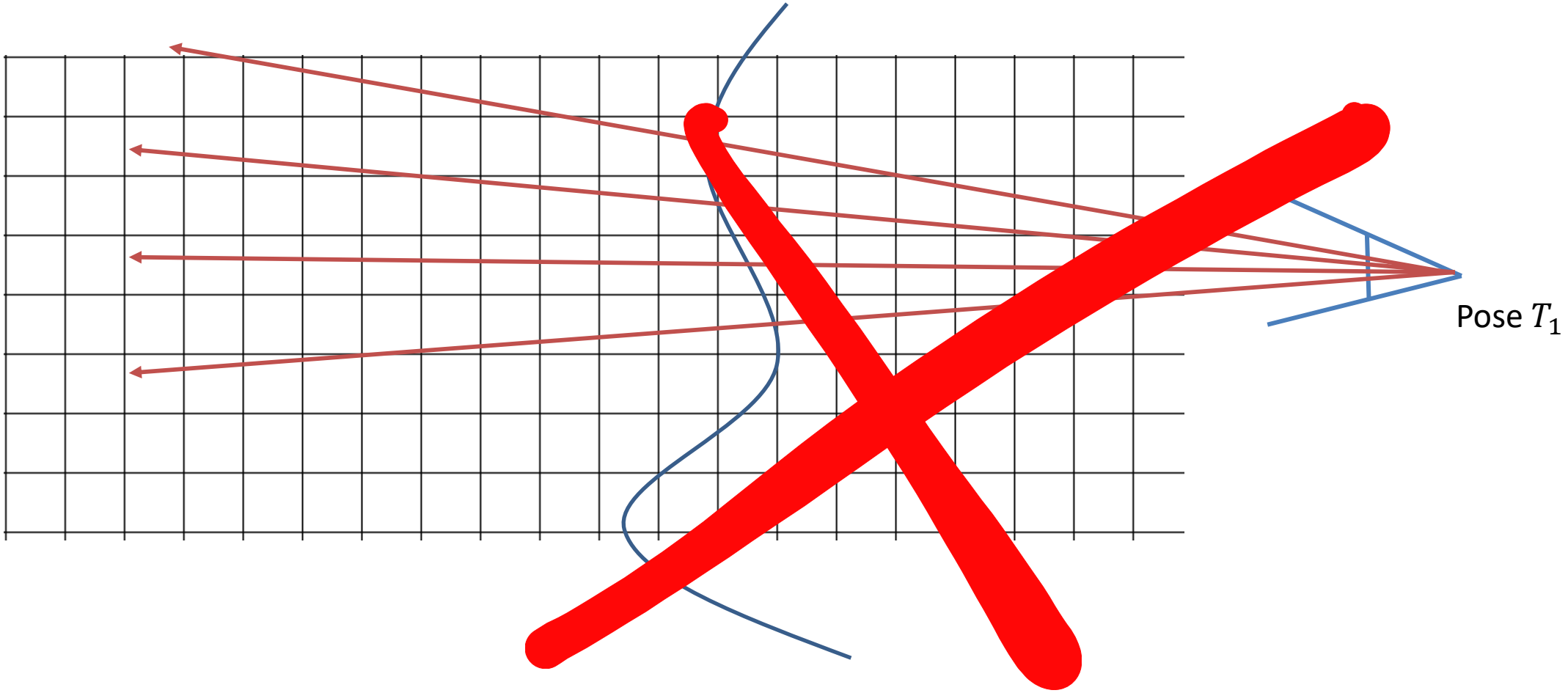
Truncated Signed Distance Field (TSDF)





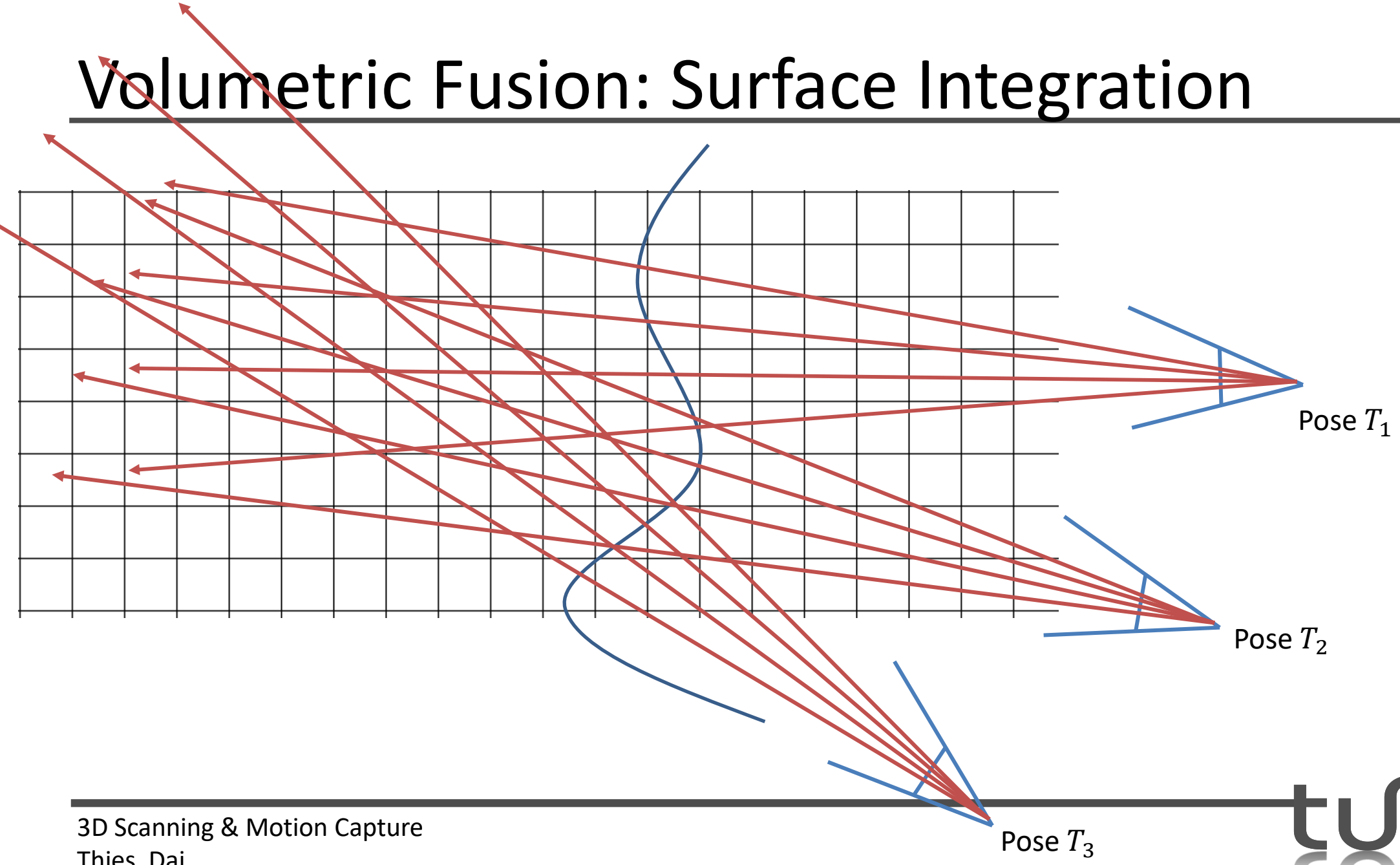
Truncated Signed Distance Field (TSDF)

Volumetric Fusion: Surface Integration



Volumetric Fusion: Surface Integration

Volumetric Fusion: Surface Integration



Volumetric Fusion

- Weighted average of signed distance values
- $D(x) = \frac{\sum w_i(x) d_i(x)}{\sum w_i(x)}$
- $W(x) = \sum w_i(x)$
- Depth value $d_i(x)$ at point x
- Weight functions $w_i(x)$
  - E.g., depth-based -> further away -> lower weight
  - E.g., $w_i(x) = \max(w(1 - depthZeroOne), 1)$
- Isosurface $D(x) = 0$

Volumetric Fusion

- Running average of signed distance values

- $D_{i+1}(x) = \frac{W_i(x)D_i(x)+w_{i+1}(x)d_{i+1}(x)}{W_i(x)+w_{i+1}(X)}$
- $W_{i+1}(x) = W_i(x) + w_{i+1}(x)$

- $W_i(x)$ and $D_i(x)$ are cumulative signed distanced and weight functions after integrating the i-th range image

Volumetric Fusion

- Surface Integration *[Curless and Levoy 96]*

```
Voxel {
    distance;
    color;
    weight;
}
```

$$D(v) = \frac{\sum_i w_i(v) d_i(v)}{\sum_i w_i(v)}$$

$$W(v) = \sum_i w_i(v)$$

Volumetric Fusion

- Surface Integration *[Curless and Levoy 96]*

```
Voxel {
    distance;
    color;
    weight;
}
```

$$D'(v) = \frac{W(v)D(v) + w_k(v)d_k(v)}{W(v) + w_k(v)}$$

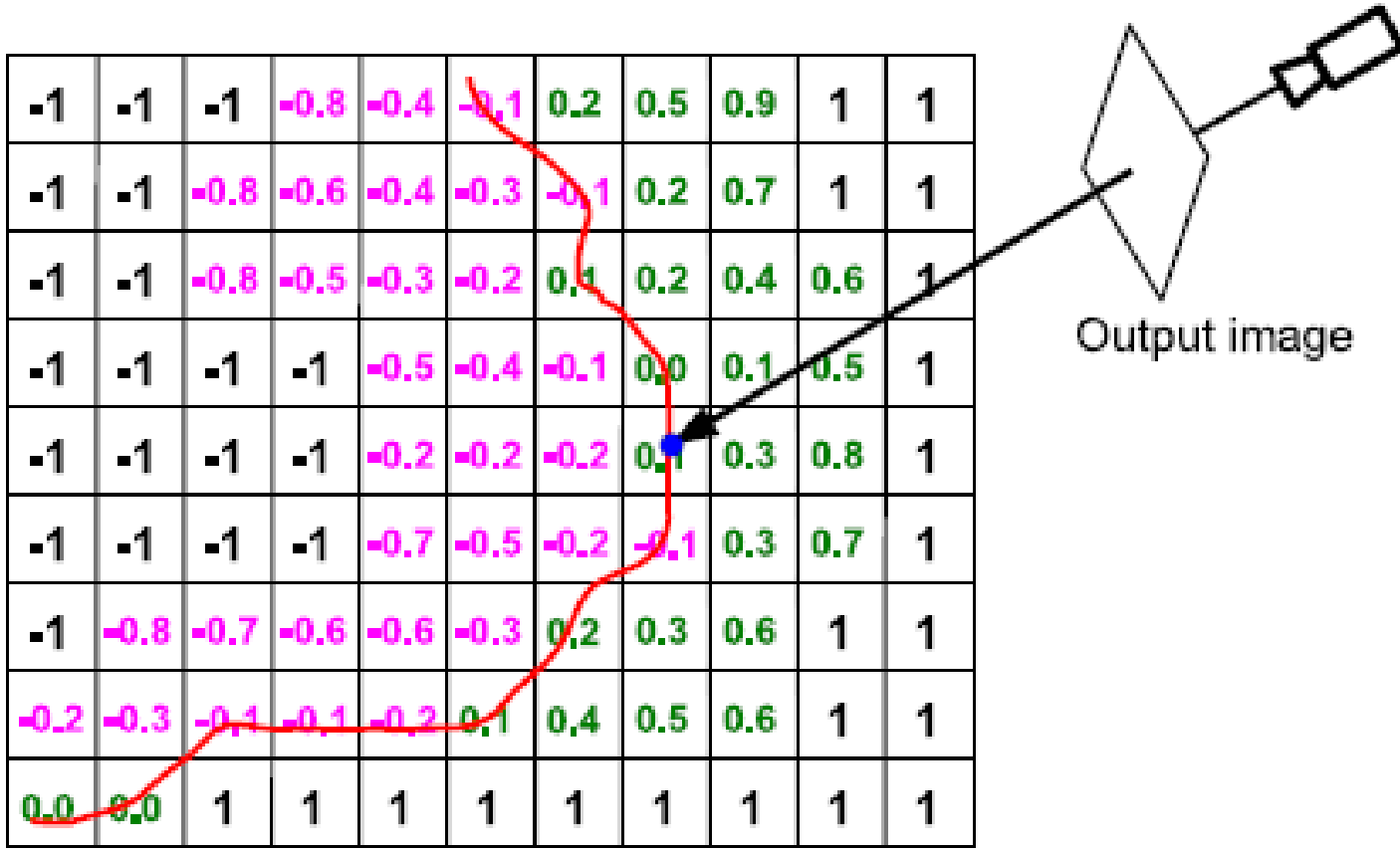
$$W'(v) = W(v) + w_k(v)$$

Volumetric Fusion

```
    void integrate(const mat4f& intrinsic, const mat4f& cameraToWorld, const DepthImage32& depthImage) {

        const mat4f worldToCamera = cameraToWorld.getInverse();
        BoundingBox3<int> voxelBounds = computeFrustumBounds(intrinsic, cameraToWorld, depthImage.getWidth(), depthImage.getHeight());

#pragma omp parallel for
        for (int k = voxelBounds.getMinZ(); k <= voxelBounds.getMaxZ(); k++) {
            for (int j = voxelBounds.getMinY(); j <= voxelBounds.getMaxY(); j++) {
                for (int i = voxelBounds.getMinX(); i <= voxelBounds.getMaxX(); i++) {

                    //transform to current frame
                    vec3f p = worldToCamera * voxelToWorld(vec3i(i, j, k));

                    //project into depth image
                    p = skeletonToDepth(intrinsic, p);

                    vec3i pi = math::round(p);
                    if (pi.x >= 0 && pi.y >= 0 && pi.x < (int)depthImage.getWidth() && pi.y < (int)depthImage.getHeight()) {
                        float d = depthImage(pi.x, pi.y);

                        //check for a valid depth range
                        if (d >= m_depthMin && d <= m_depthMax) {

                            //update free space counter if voxel is in front of observation
                            if (p.z < d) {
                                (*this)(i, j, k).freeCtr++;
                            }

                            //compute signed distance; positive in front of the observation
                            float sdf = d - p.z;
                            float truncation = getTruncation(d);

                            //if (std::abs(sdf) < truncation) {
                            if (sdf > -truncation) {
                                Voxel& v = (*this)(i, j, k);
                                v.sdf = (v.sdf * (float)v.weight + sdf * (float)m_weightUpdate) / (float)(v.weight + m_weightUpdate);
                                v.weight = (uchar)std::min((int)v.weight + (int)m_weightUpdate, (int)std::numeric_limits<unsigned char>::max())
                            }
```

Surface Rendering: Ray Casting

- Cast ray through focal point of each pixel
- Traverse voxels along the ray
  (truncation step size)
- Find surface by checking for sign change
- Compute intersection
  - e.g., bilinear search
  - computer normal by $\nabla TSDF$

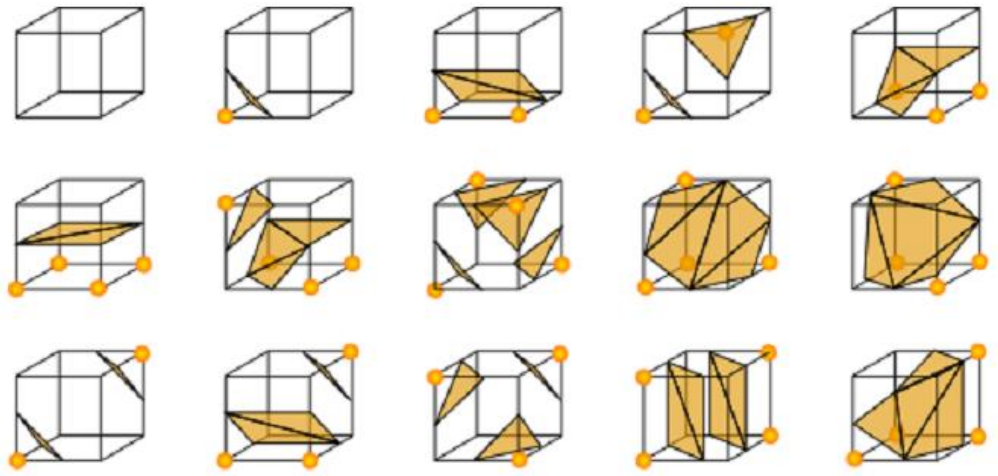


Surface Extraction: Marching Cubes

-1	-1	-1	-0.8	-0.4	-0.1	0.2	0.5	0.9	1	1
-1	-1	-0.8	-0.6	-0.4	-0.3	-0.1	0.2	0.7	1	1
-1	-1	-0.8	-0.5	-0.3	-0.2	0.1	0.2	0.4	0.6	1
-1	-1	-1	-1	-0.5	-0.4	-0.1	0.0	0.1	0.5	1
-1	-1	-1	-1	-0.2	-0.2	-0.2	0.1	0.3	0.8	1
-1	-1	-1	-1	-0.7	-0.5	-0.2	-0.1	0.3	0.7	1
-1	-0.8	-0.7	-0.6	-0.6	-0.3	0.2	0.3	0.6	1	1
-0.2	-0.3	-0.1	-0.1	-0.2	0.1	0.4	0.5	0.6	1	1
0.0	0.0	1	1	1	1	1	1	1	1	1

surface (zero-crossing)

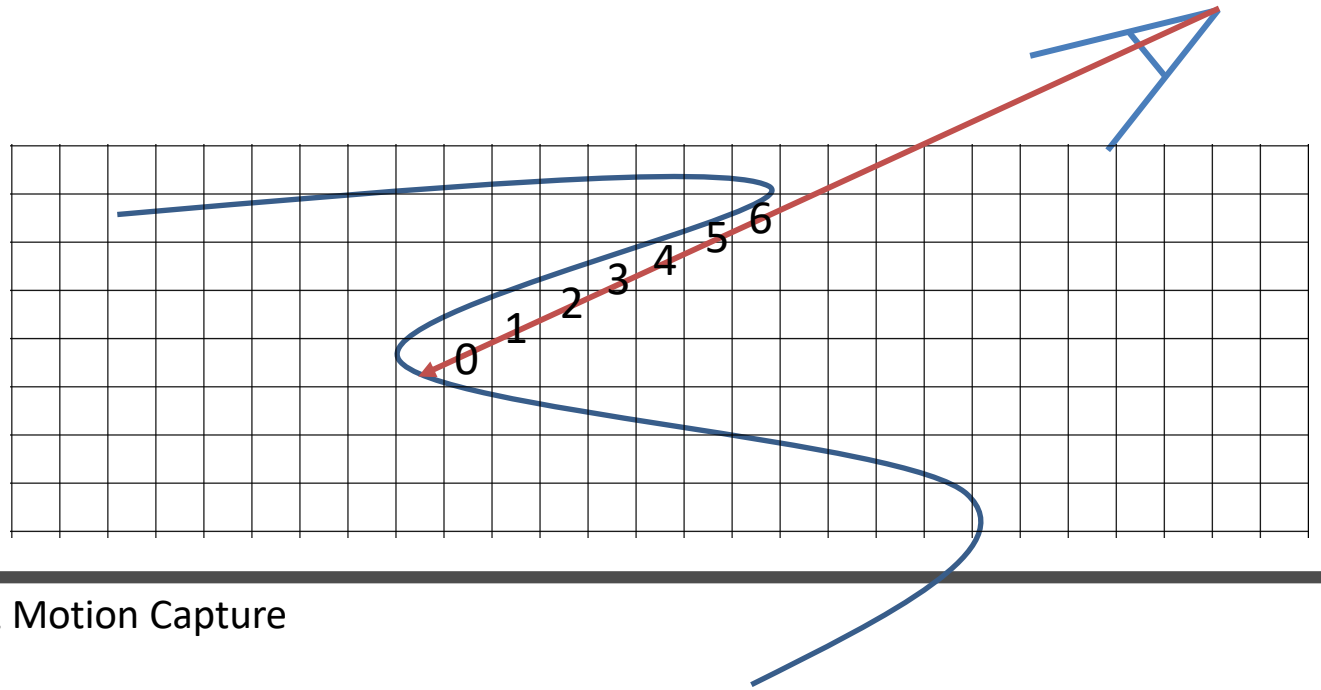
TSDF volume grid



Marching Cubes table

Volumetric Fusion

- Projective integration is not a ‘true’ SDF
  - Less accurate the further away from the surface
  - More accurate with more different views
  - Could apply distance transform but misses data

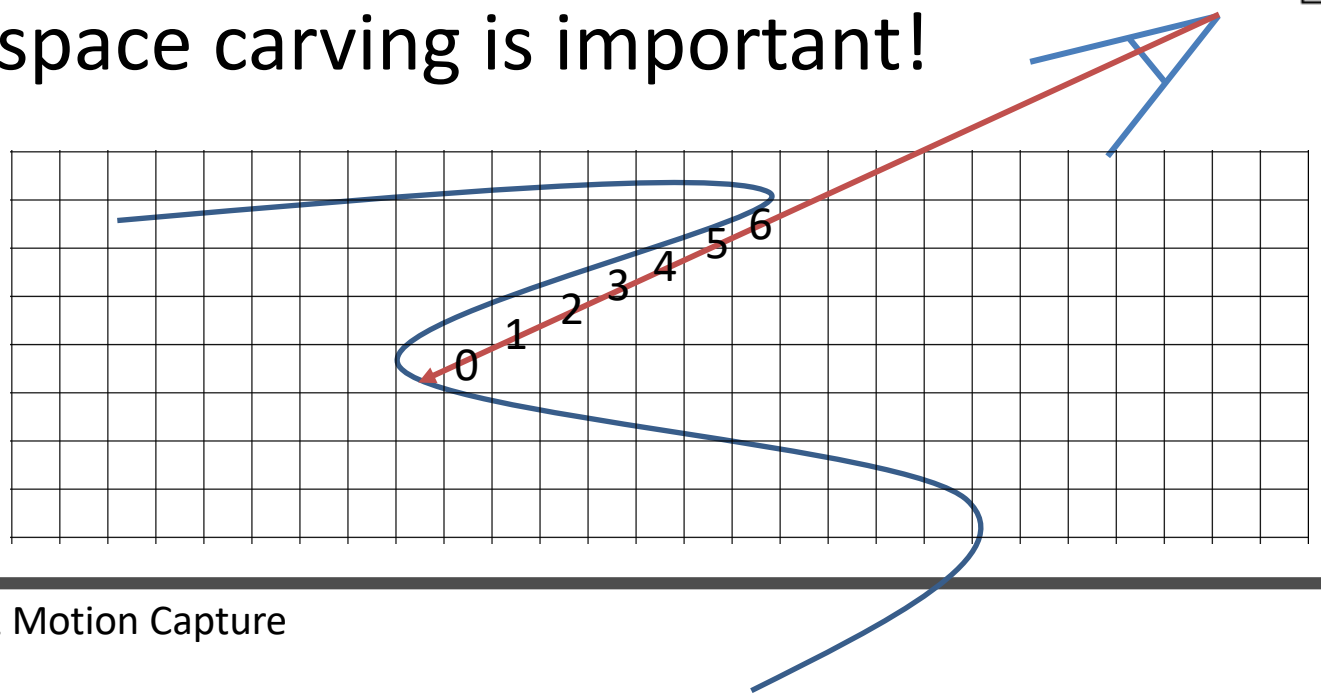


Spot the error!

Volumetric Fusion

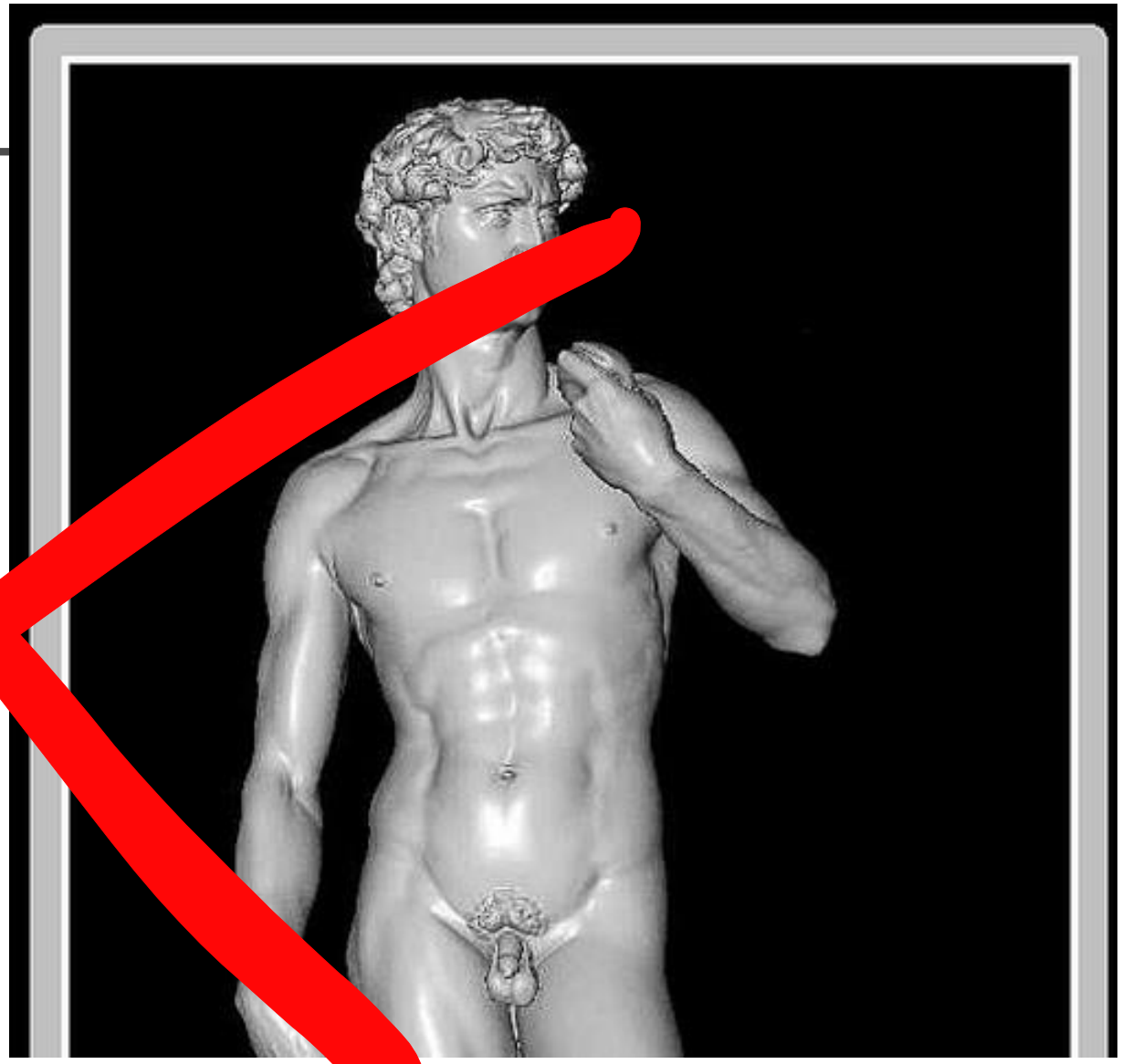
- Interesting property though:
  - ‘0’ -> surface
  - ‘negative’ -> unknown
  - ‘positive’ -> free space
- Free space carving is important!

-1	-1	-1	-0.8	-0.4	-0.1	0.2	0.5	0.9	1	1
-1	-1	-0.8	-0.6	-0.4	-0.3	-0.1	0.2	0.7	1	1
-1	-1	-0.8	-0.5	-0.3	-0.2	0.1	0.2	0.4	0.6	1
-1	-1	-1	-1	-0.5	-0.4	-0.1	0.0	0.1	0.5	1
-1	-1	-1	-1	-0.2	-0.2	-0.2	0.1	0.3	0.8	1
-1	-1	-1	-1	-0.7	-0.5	-0.2	-0.1	0.3	0.7	1
-1	-0.8	-0.7	-0.6	-0.6	-0.3	0.2	0.3	0.6	1	1
-0.2	-0.3	-0.1	-0.1	-0.2	0.1	0.4	0.5	0.6	1	1
0.0	0.0	1	1	1	1	1	1	1	1	1



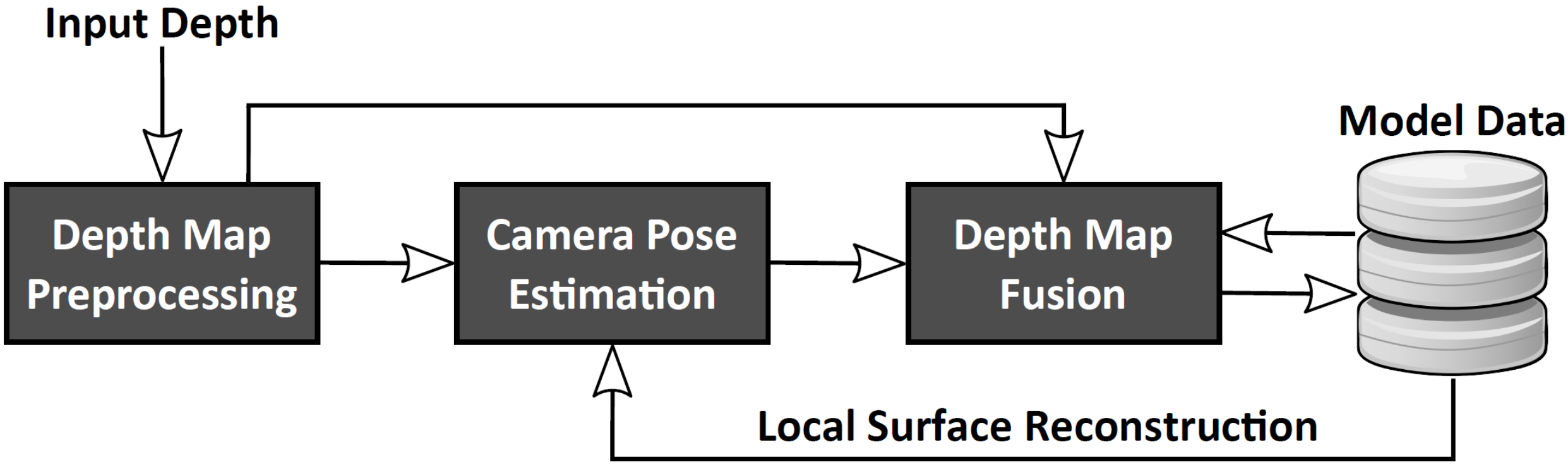
Volumetric Fusion

Sta rd Bunny

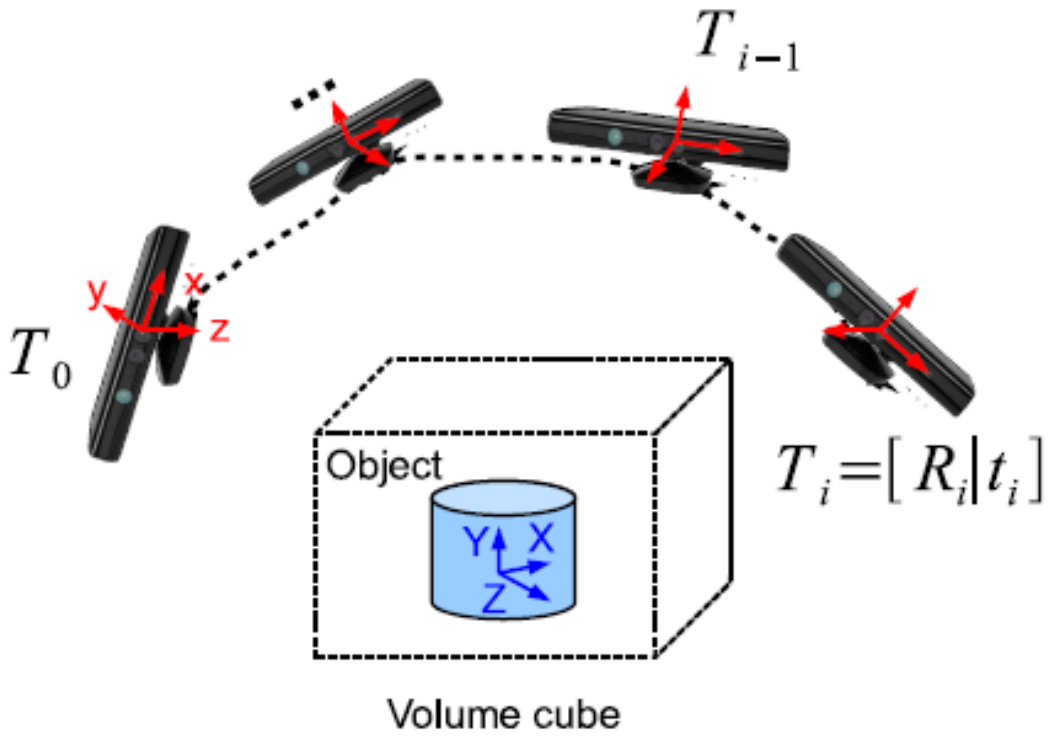


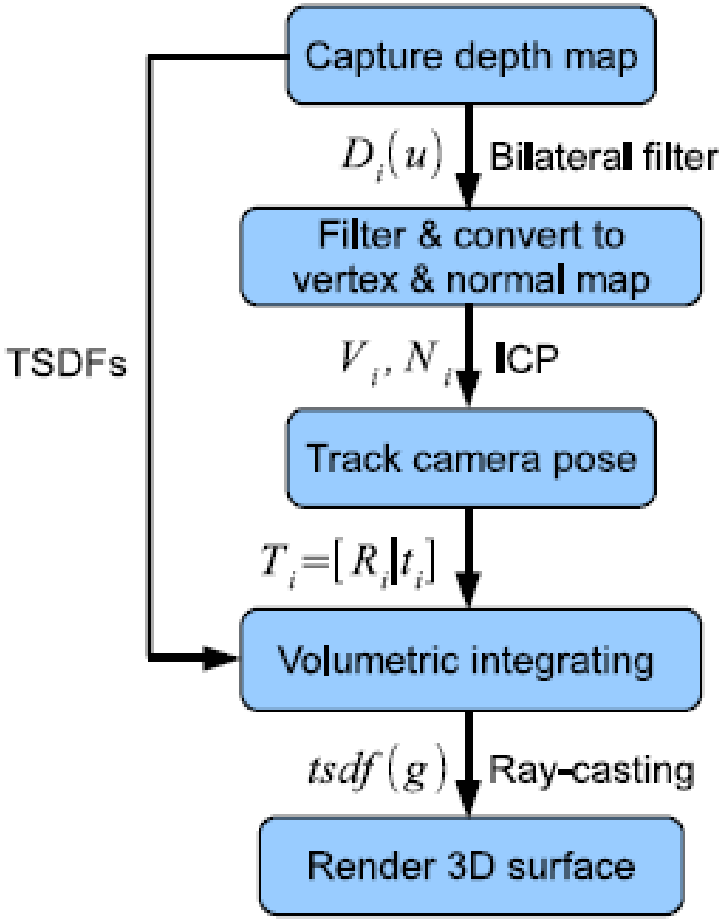
[Curless and levoy 96] [Levoy et al. 2001]

Online 3D Reconstruction



Online 3D Reconstruction





Online 3D Reconstruction

- Often frame-to-model tracking rather than frame-to-frame
  - Use ray-casted normal map -> less noise!

More robust

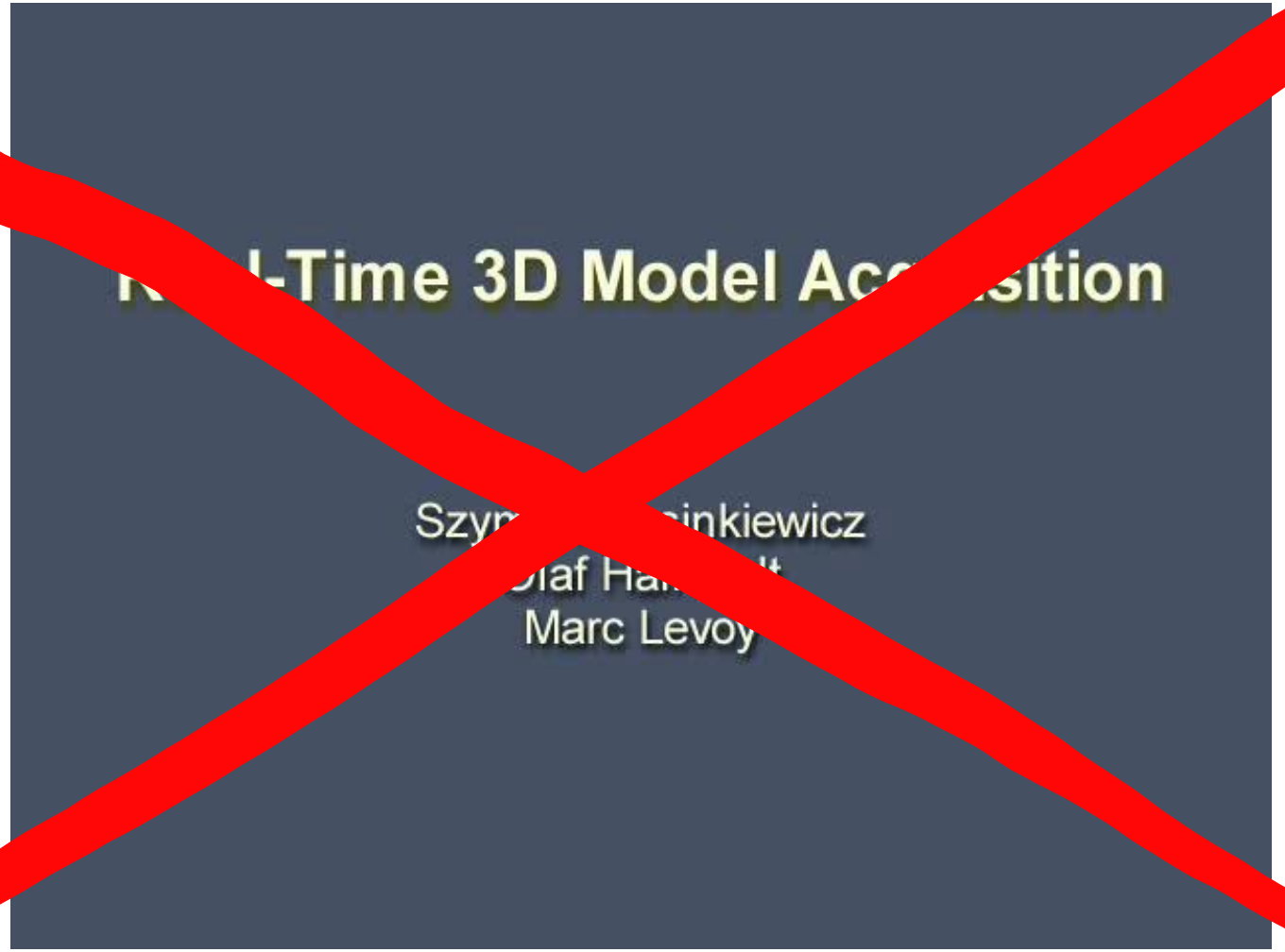
Input normal map

Ray-casted normal map from SDF

Prominent Literature

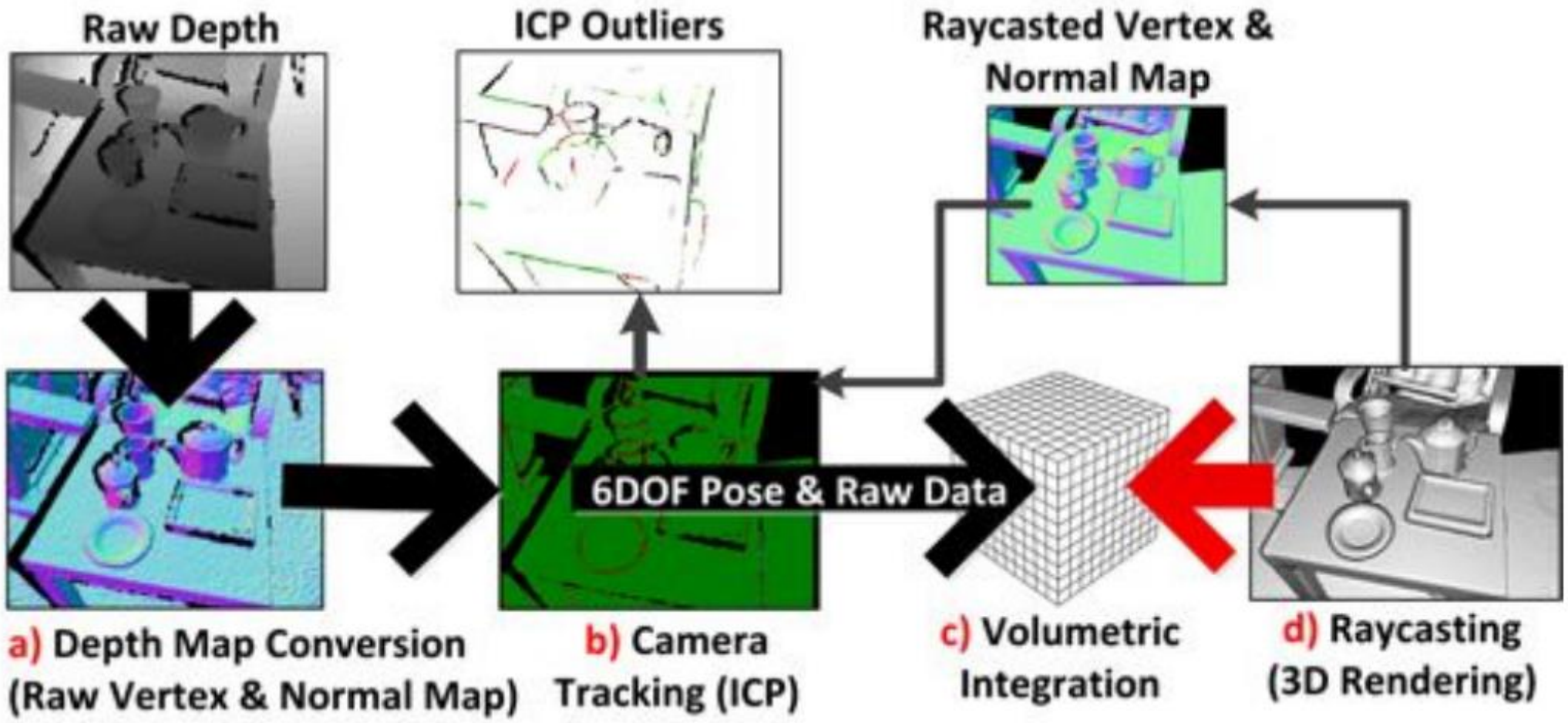
- A volumetric method for building [… Curless and Levoy 96]
  - TSDF integration
- Szymon Rusinkiewicz
  - Efficient Variants of ICP
  - Real-time Model Acquisition <- Siggraph 2002!
- KinectFusion [Newcombe/Izadi et al. 11]

Real-time 3D Model Acquisition



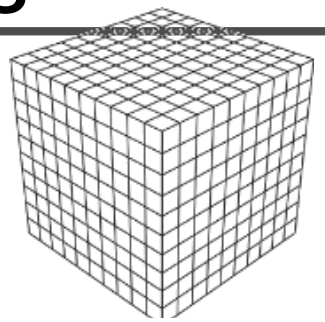
Kinect Fusion

Kinect Fusion

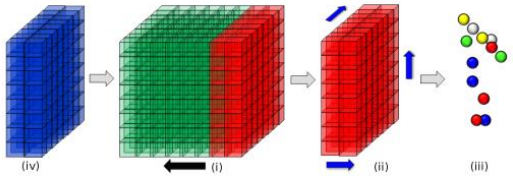


Spatial Representations

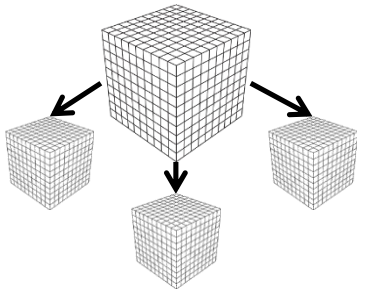
- Kinect Fusion -> Dense Grid
  - [Izadi/Newcombe et al. 11]



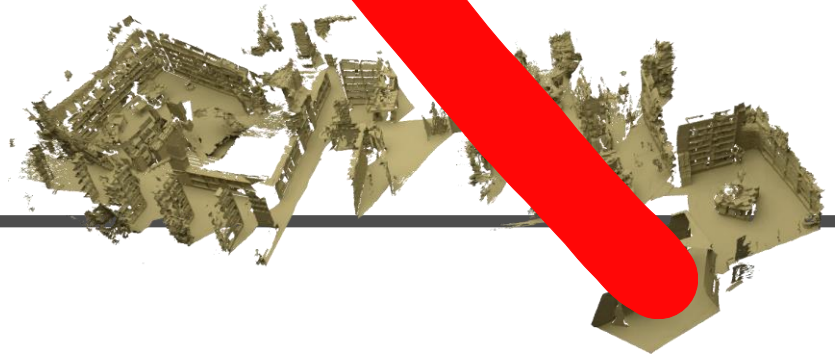
- Extended Kinect Fusion
  - [Roth et al. 12; Whelan et al. 12]



- Hierarchical Fusion
  - [Stueckler et al. 12; Chen et al. 13]

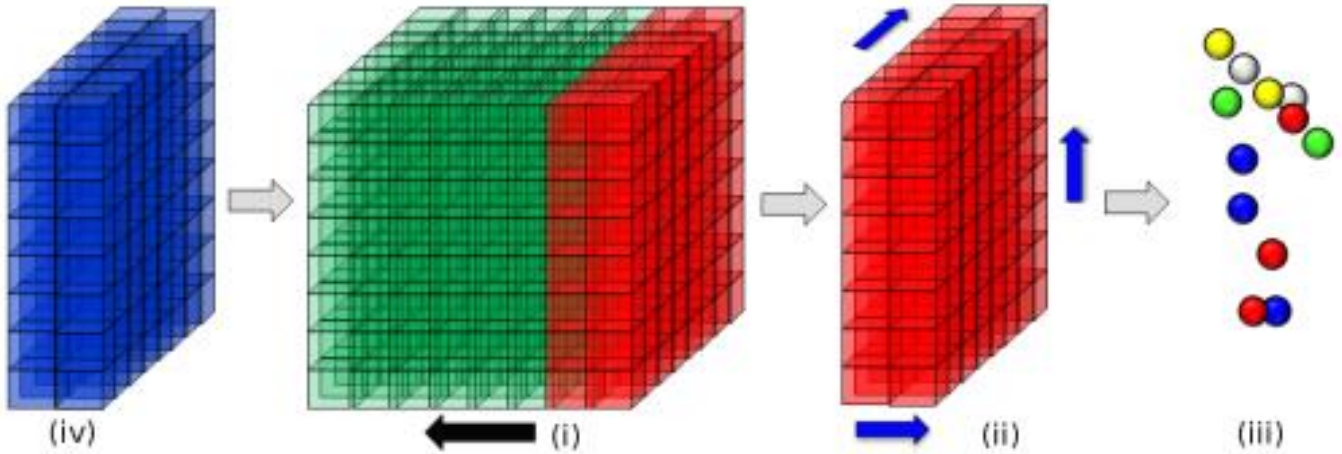


- Voxel Hashing
  - [Niessner et al. 13]



Extended Kinect Fusion

- Moving Volume



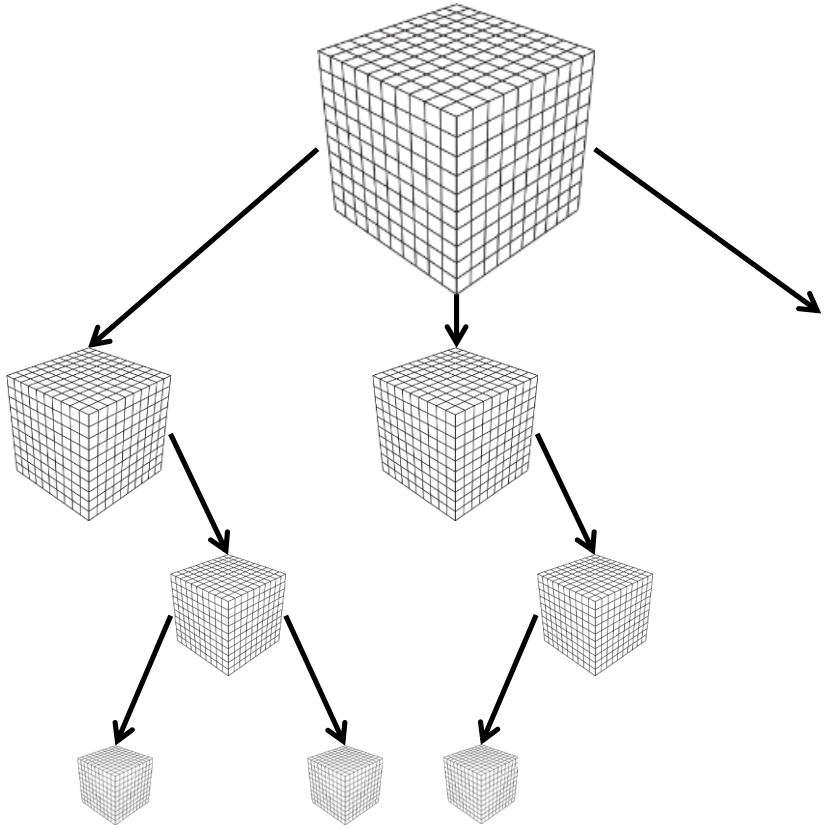
Current Input

Rolling Volume

Marching Cubes on boundary

+ very easy to implement
+ unlimited spatial volumes
- cannot integrate into previously regions

Hierarchical Fusion



Spatial subdivision with tree
SDF values live only on finest level

+ efficient storage: large spatial extent, high local res
+ fast lookups
- costly updates during integration (need tree update)
e.g., insertion, removal, etc.

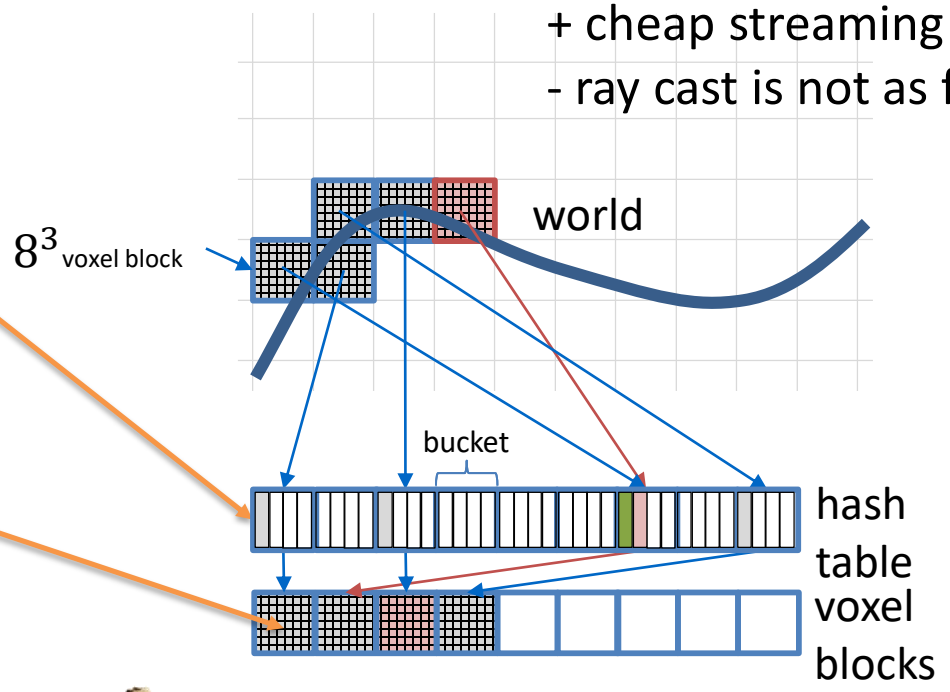
Voxel Hashing

Spatial Hash Function:

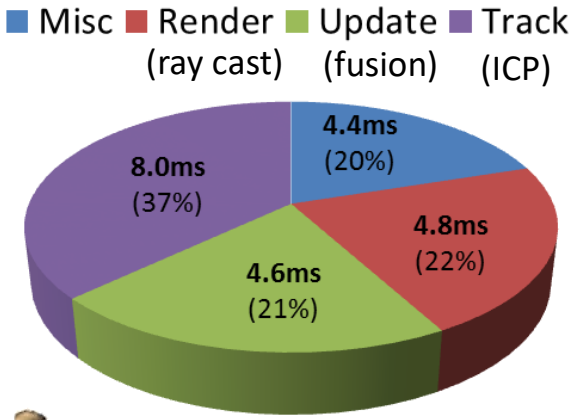
$H(x, y, z) = (x \cdot p_1 \oplus y \cdot p_2 \oplus z \cdot p_3) \bmod n$

```
struct HashEntry {
   short position[3];
   short offset;
   int   pointer;
};
```

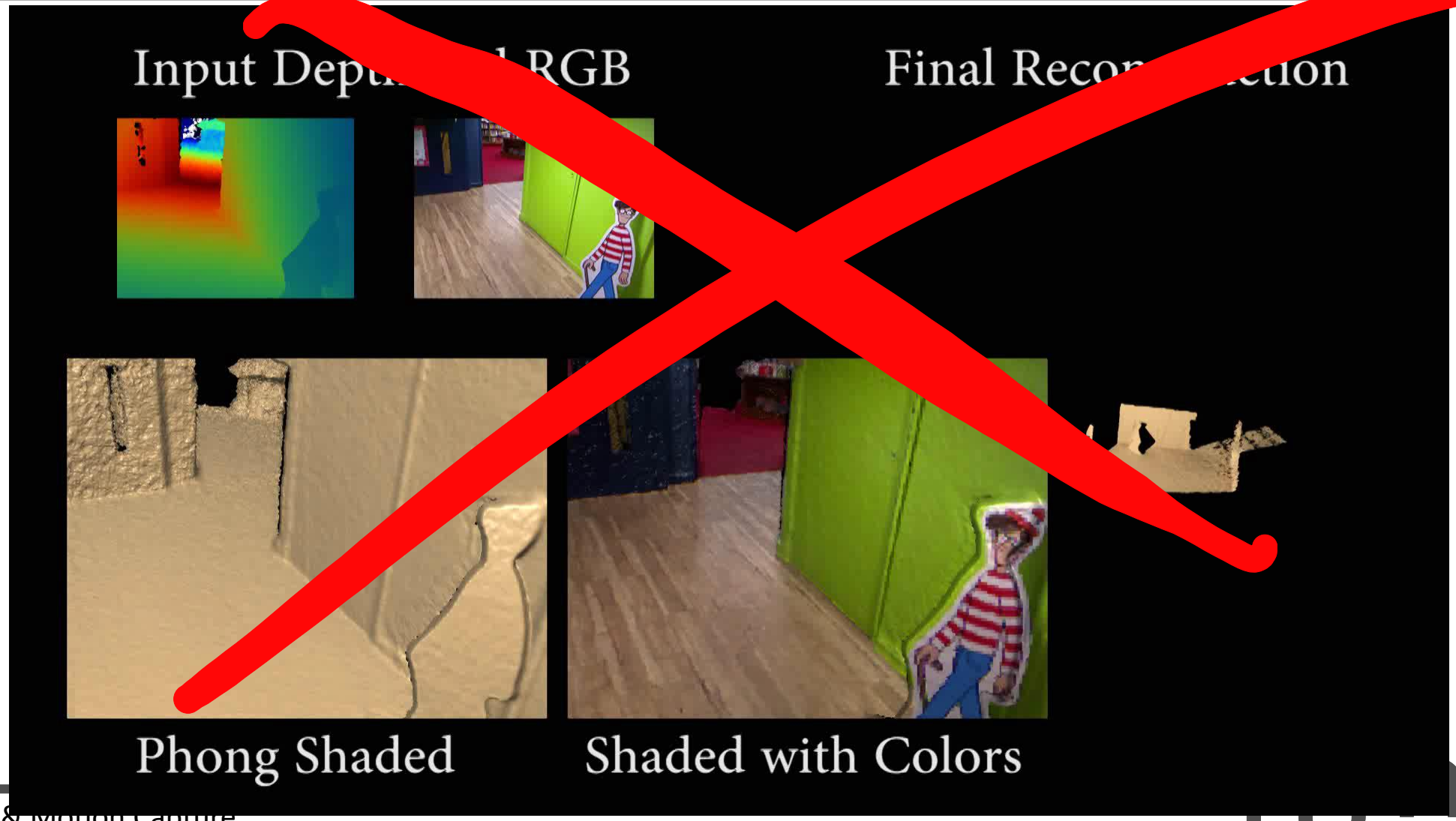
```
struct Voxel {
   float sdf;
   uchar colorRGB[3];
   uchar weight;
};
```



+ efficient storage: large spatial extent, high local res
+ extremely fast updates:
  e.g., insertion, removal, etc. is O(1)
+ cheap streaming
- ray cast is not as fast (need to check voxel by voxel)

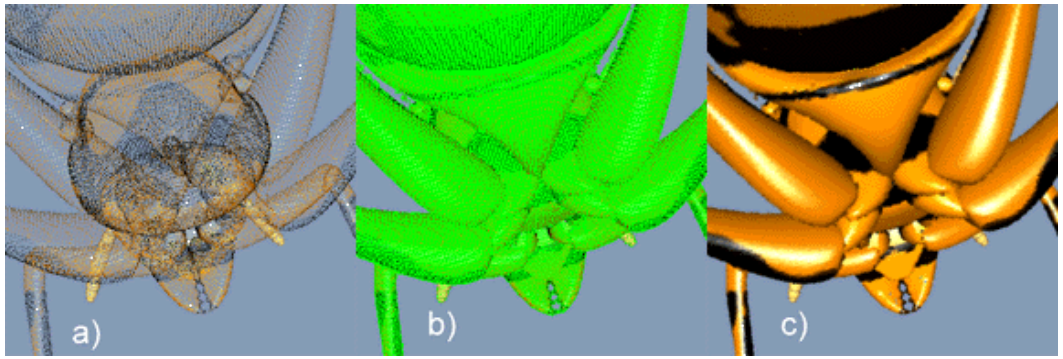


Voxel Hashing



Other Representations: Point-based Fusion

- Surfel representation
  - Point + normal + radius
- Rendering through splatting
  - Use graphics pipeline



+ unstructured storage -> large extent and high-res
+ in a way easier for dynamics (although not quite there)
- surface quality slightly less than volumetric SDF

[Keller et al. 13] Real-time 3D Reconstruction in Dynamic Scenes using Point-based Fusion

Other Representations: Point-based Fusion

Surfel-based Representation

TSDF Implicit Surface

Drift and Loop Closure Problems

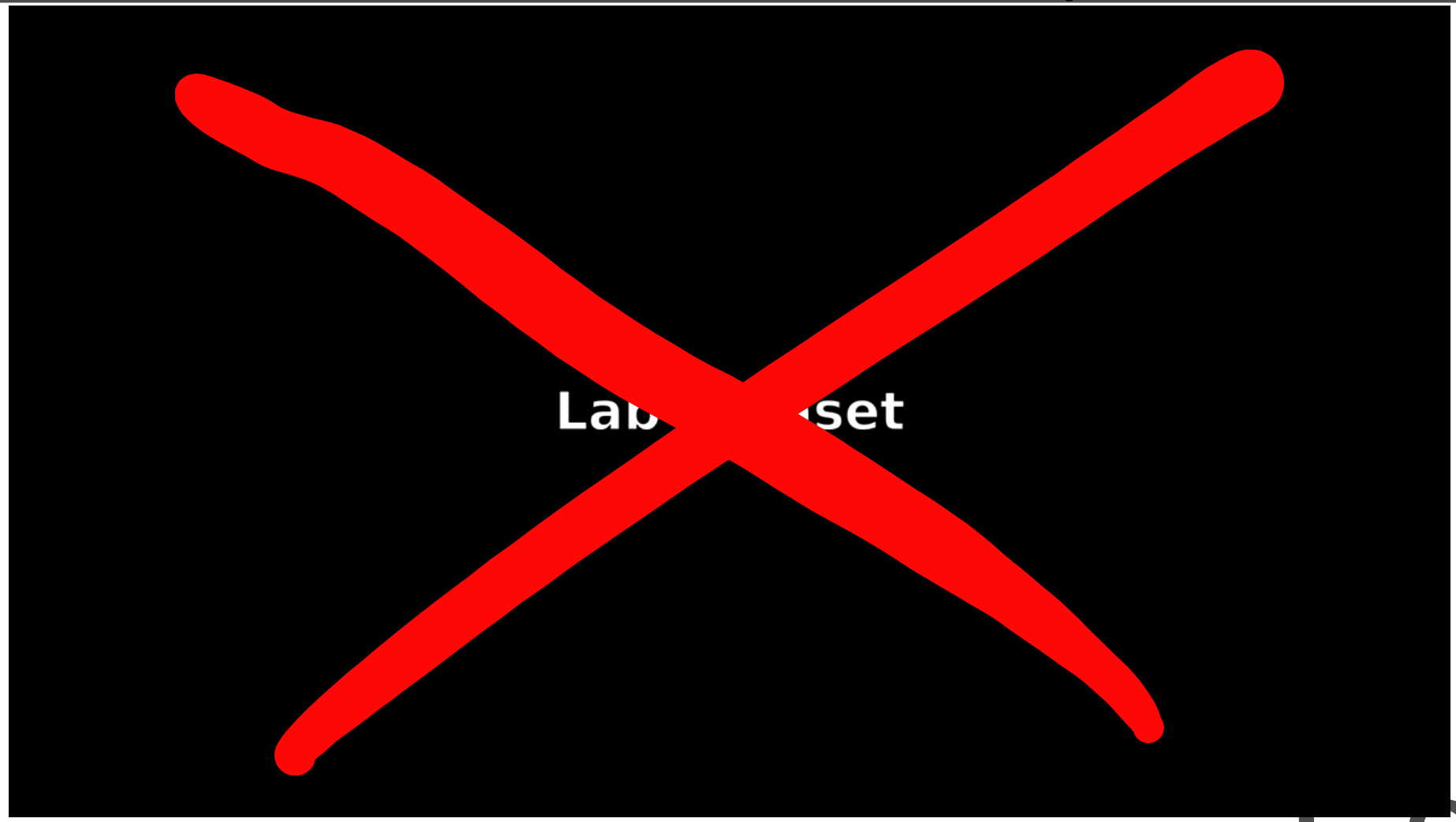
Elastic Fusion: Surfel-based + Loop Closure

- Point-based representation
- Frame-to-model tracking

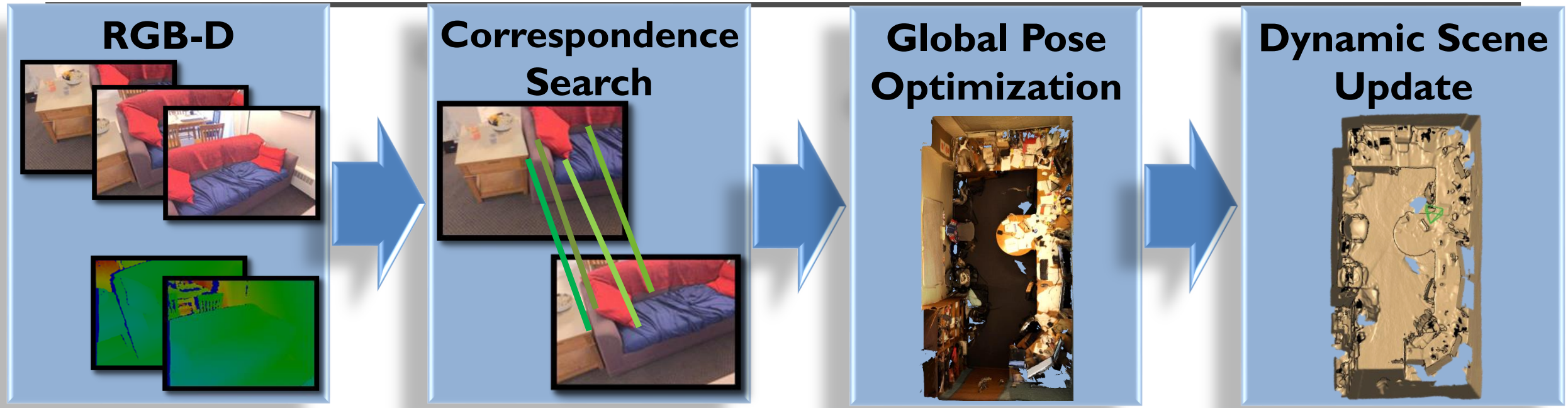
- Explicitly detect loop closures through localization
- Non-rigidly warp representation to close loops

[Whelan et al. 16] ElasticFusion: Dense SLAM Without A Pose Graph

Elastic Fusion: Surfel-based + Loop Closure



Bundle Fusion



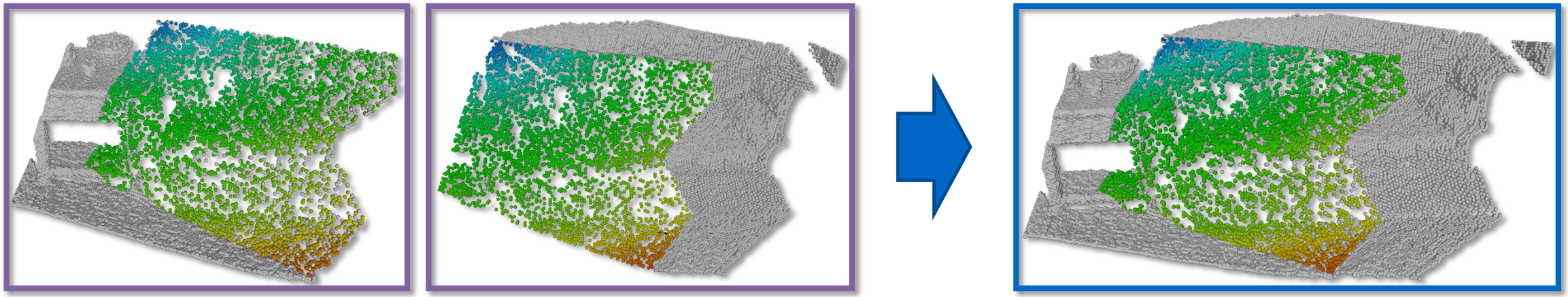
$$E(T) = w_{sparse} E_{sparse}(T) + w_{dense} E_{dense}(T)$$

$$E_{sparse}(T) = \sum_{i,j}^{\#frames} \sum_{k}^{\#corresp.} \left\| T_i p_{ik} - T_j p_{jk} \right\|_2^2$$

Bundle Fusion: Dense Terms

$$E_{dense}(T) = w_{depth} E_{depth}(T) + w_{color} E_{color}(T)$$

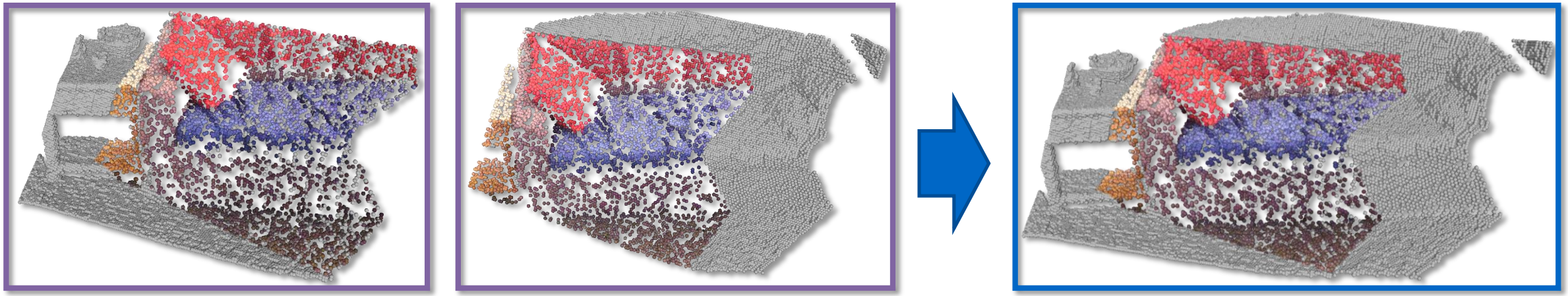
$$E_{depth}(T) = \sum_{i,j}^{\#\text{frames}} \sum_{k}^{\#\text{pixels}} \left\| \left(p_k - T_i^{-1} T_j \pi_d^{-1} \left(D_j \left(\pi_d \left(T_j^{-1} T_i p_k \right) \right) \right) \right) \cdot n_k \right\|_2^2$$



Bundle Fusion: Dense Terms

$$E_{dense}(T) = w_{depth}E_{depth}(T) + w_{color}E_{color}(T)$$

$$E_{color}(T) = \sum_{i,j}^{\#\text{frames}} \sum_{k}^{\#\text{pixels}} \left\| \nabla I(\pi_c(p_k)) - \nabla I(\pi_c(T_j^{-1} T_i p_k)) \right\|_2^2$$



Bundle Fusion: On-the-fly Scene Updates

- Surface Integration *[Curless and Levoy 96]*

```
Voxel {
    distance;
    color;
    weight;
}
```

$$D(v) = \frac{\sum_i w_i(v) d_i(v)}{\sum_i w_i(v)}$$

$$W(v) = \sum_i w_i(v)$$

Bundle Fusion: On-the-fly Scene Updates

- Surface Integration *[Curless and Levoy 96]*

```
Voxel {
    distance;
    color;
    weight;
}
```

$$D'(v) = \frac{W(v)D(v) + w_k(v)d_k(v)}{W(v) + w_k(v)}$$

$$W'(v) = W(v) + w_k(v)$$

Bundle Fusion: On-the-fly Scene Updates

- Surface De-integration: remove d_k from weighted average

```
Voxel {
    distance;
    color;
    weight;
}
```

$$D'(v) = \frac{W(v)D(v) - w_k(v)d_k(v)}{W(v) - w_k(v)}$$

$$W'(v) = W(v) - w_k(v)$$

Bundle Fusion: On-the-fly Scene Updates

Bundle Fusion: On-the-fly Scene Updates

Integrated
(wrong pose)

Bundle Fusion: On-the-fly Scene Updates

De-integrated

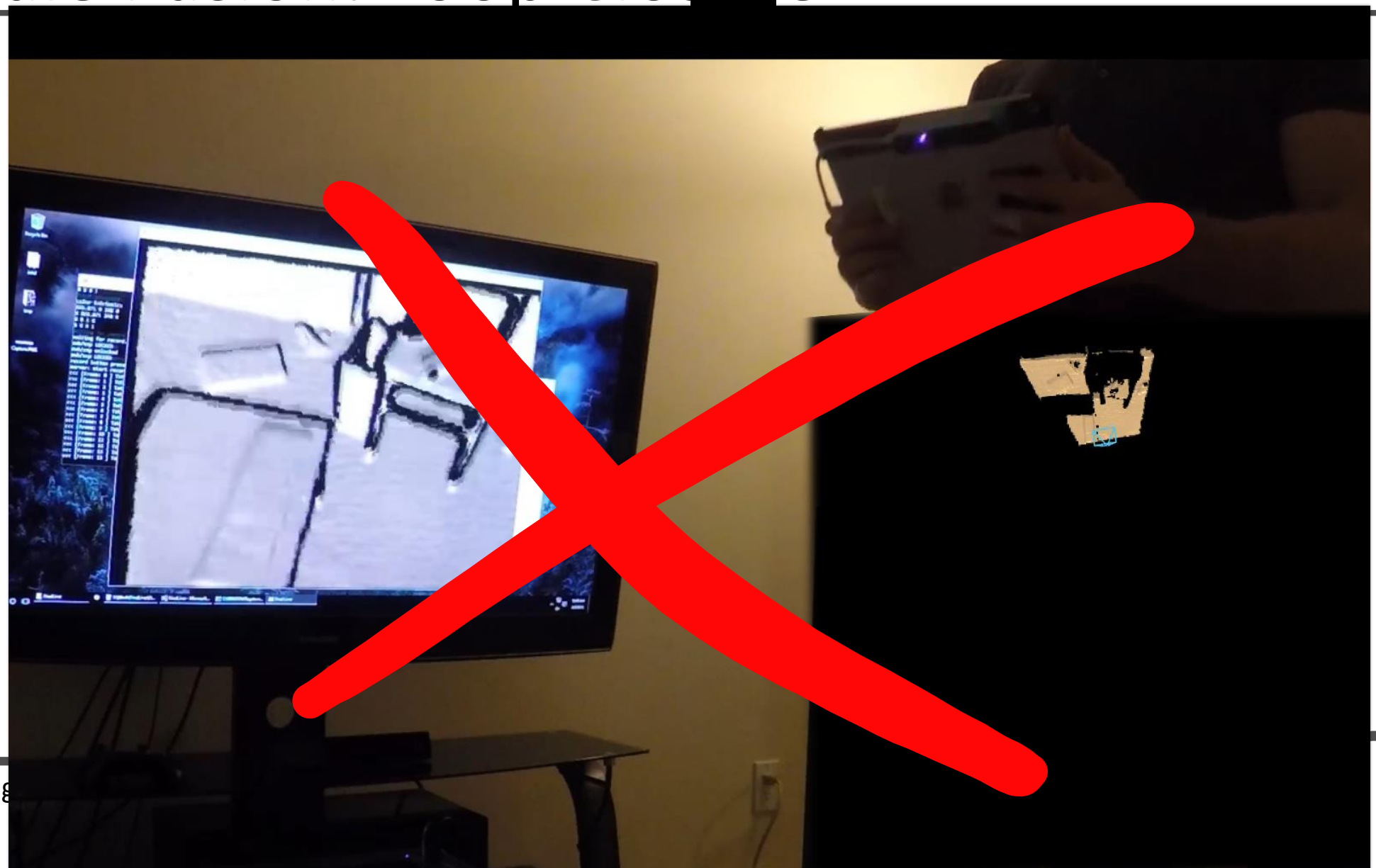
Bundle Fusion: On-the-fly Scene Updates

Re-integrated with updated pose

Bundle Fusion: Live Capture

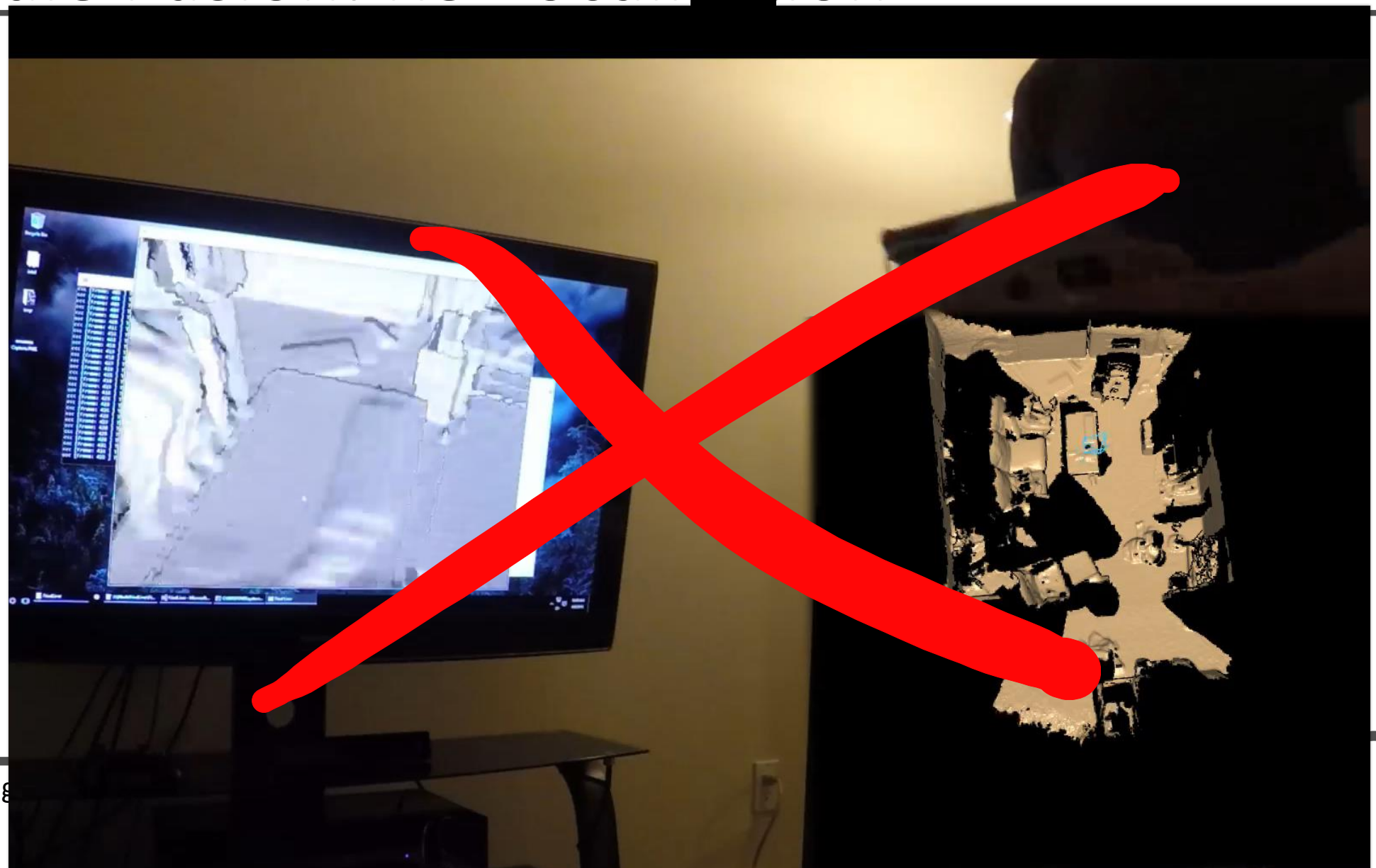
Bundle Fusion: Loop Closure

Bundle Fusion: Loop Closure



Bundle Fusion: Re-Localization

Bundle Fusion: Re-Localization



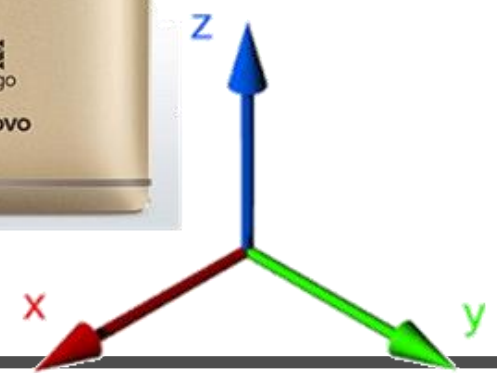
BF: Results

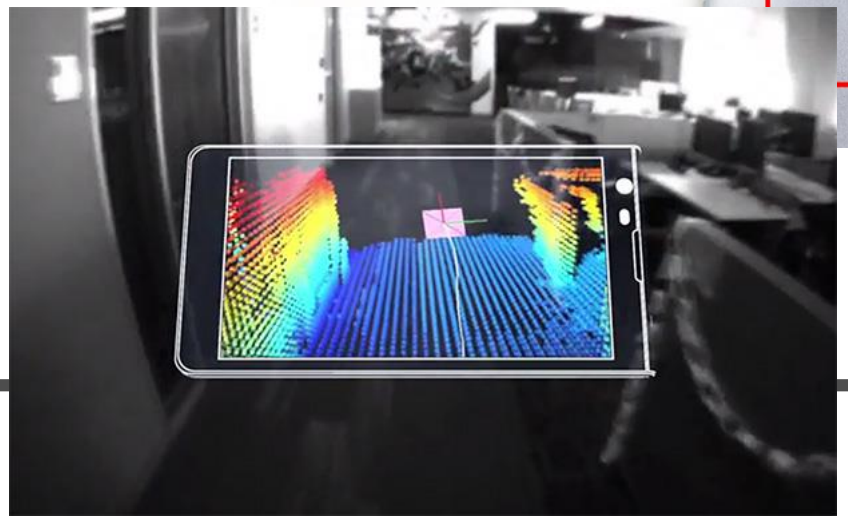
BF: Results

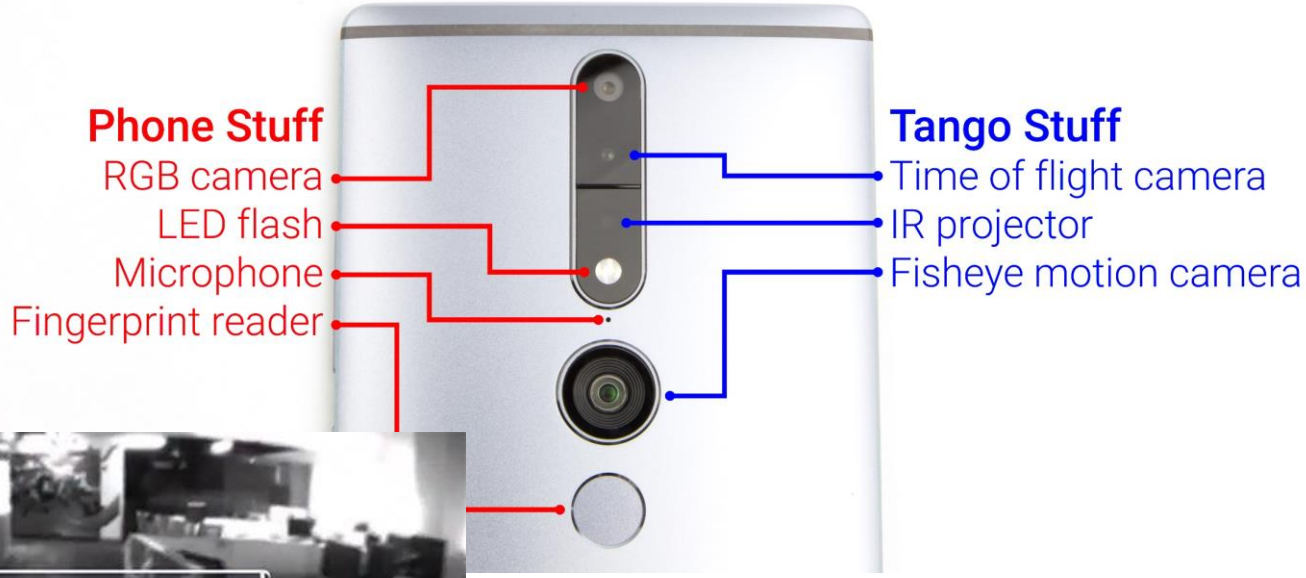
Other Things (e.g., how to scan a white wall?)

- Sensor fusion IMU + sparse feature tracking (+ ICP)
  - E.g., Google Tango or Microsoft HoloLens

device
position: x,y,z
orientation: x,y,z,w







Administrative

- Tutorials
  - Tomorrow 16:00
    - Assignment 4
    - Project Proposals due

- Next Lecture:
  - Non-rigid Surface Tracking & Reconstruction