Camera Tracking Using Dense 3D Reconstruction

Erik Bylow

Motivation

- Doing dense 3D-reconstruction in real-time
- Application in augmented reality
- Useful in robotics
- Online shopping

(Loading Video...)

Outline

- Motivation
- Fusing the depth images into a global model
- Track the depth sensor against this model
- Results and Conclusion

Fusing The Depth Images Into a Global Model With Known Camera Poses

Fusing the Depth Maps

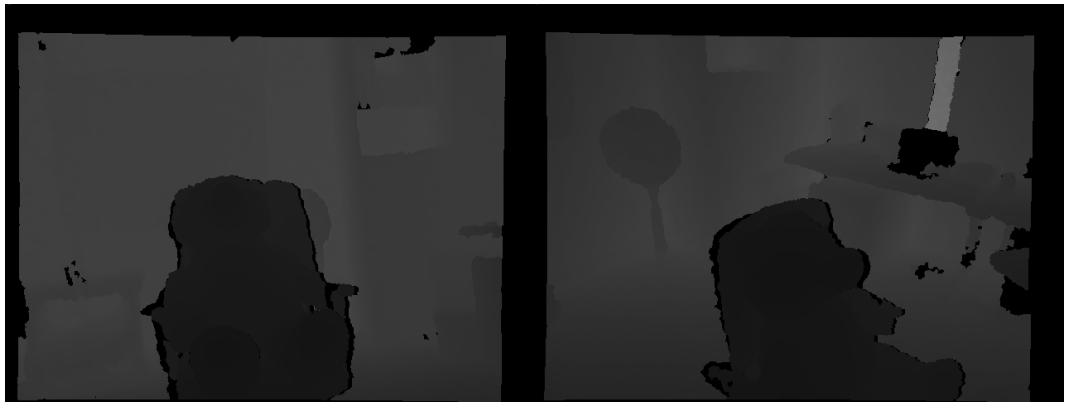


Figure: A couple of depth maps which gives the distance between the camera and the surface

Figure: Example of a reconstructed surface from a depth image

Fusing the Depth Maps

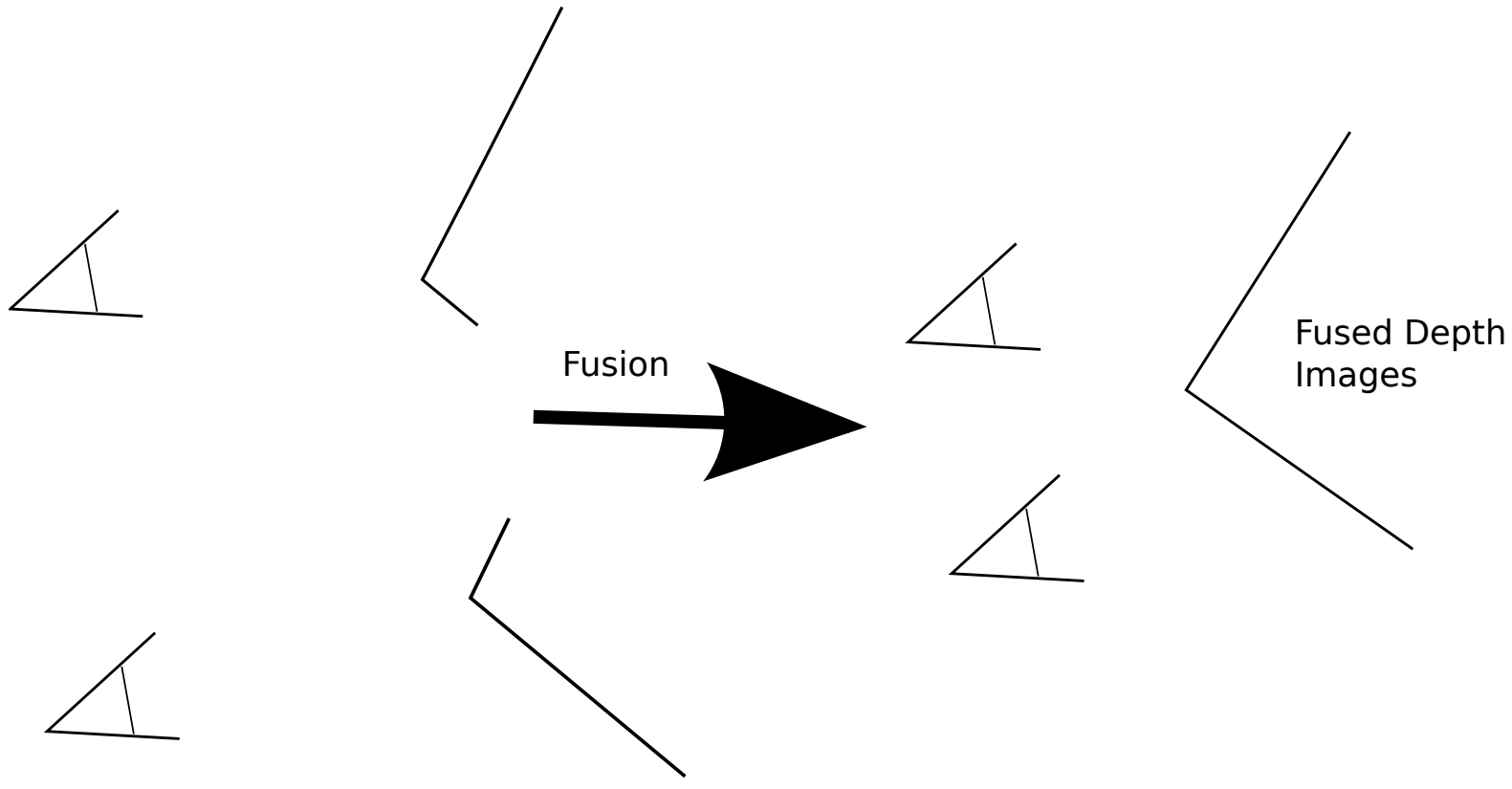


Figure: Two slightlty overlapping depth images are fused into one global model

Fusing the Depth Maps

- A Signed Distance Function can implicitly represent the surface

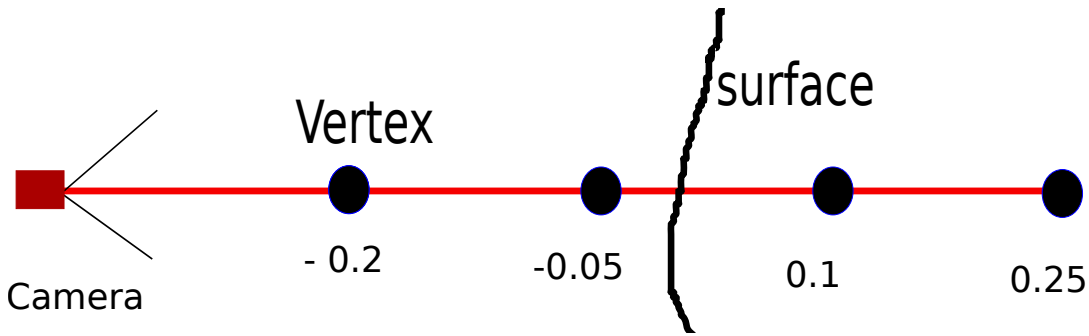


Figure: The principle of a signed distance function, one dimensional

Fusing the Depth Maps

- Measure the distance between a vertex and the surface

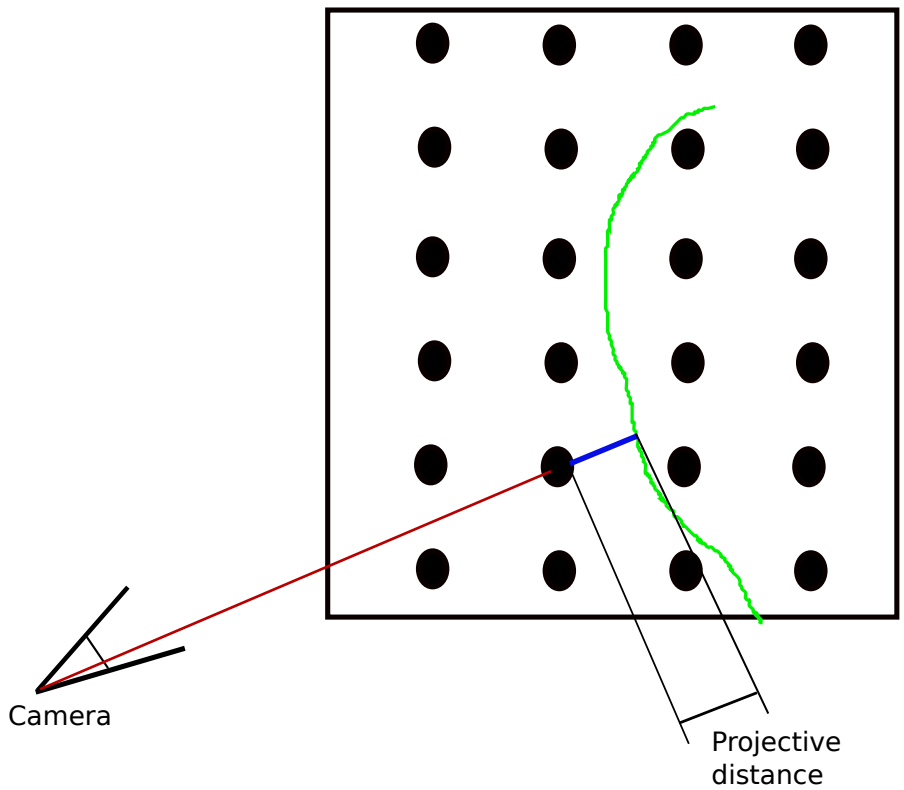


Figure: The projective distance between a vertex and the surface

Fusing the Depth Maps

- Truncated Signed Distance TSDF(x):

$$TSDF(x) = \begin{cases} -\mu & \text{if } \rho(x) < \mu \\ \rho(x) & \text{if } |\rho(x)| \leq \mu \\ \mu & \text{if } \rho(x) > \mu \end{cases}$$

$\rho(x)$ is the projective distance between vertex x and the surface.

Fusing the Depth Maps

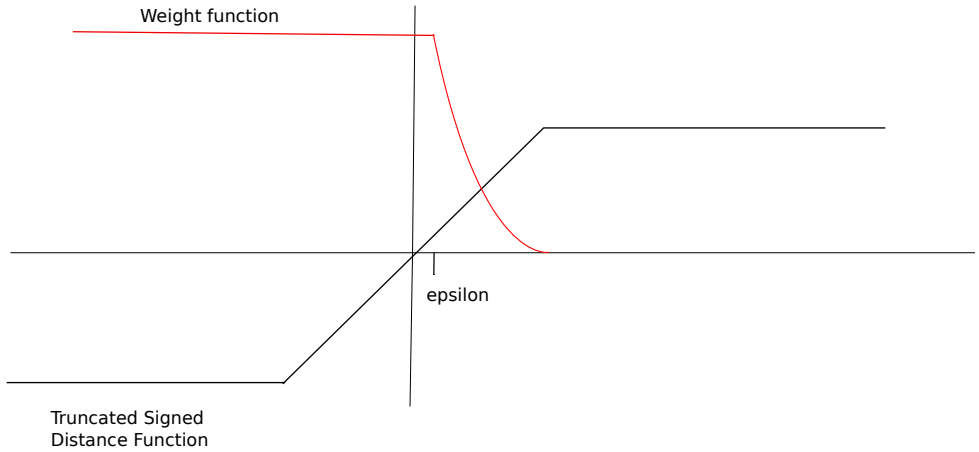


Figure: The truncated distance function and how the weight function looks like

$$W(x) = \begin{cases} 1 & \text{if } \rho(x) < \epsilon \\ \exp(-\sigma(\rho(x) - \epsilon)^2) & \text{if } \rho(x) > \epsilon \text{ and } \rho(x) \leq \mu \\ 0 & \text{if } \rho(x) > \mu \end{cases}$$

Fusing the Depth Maps

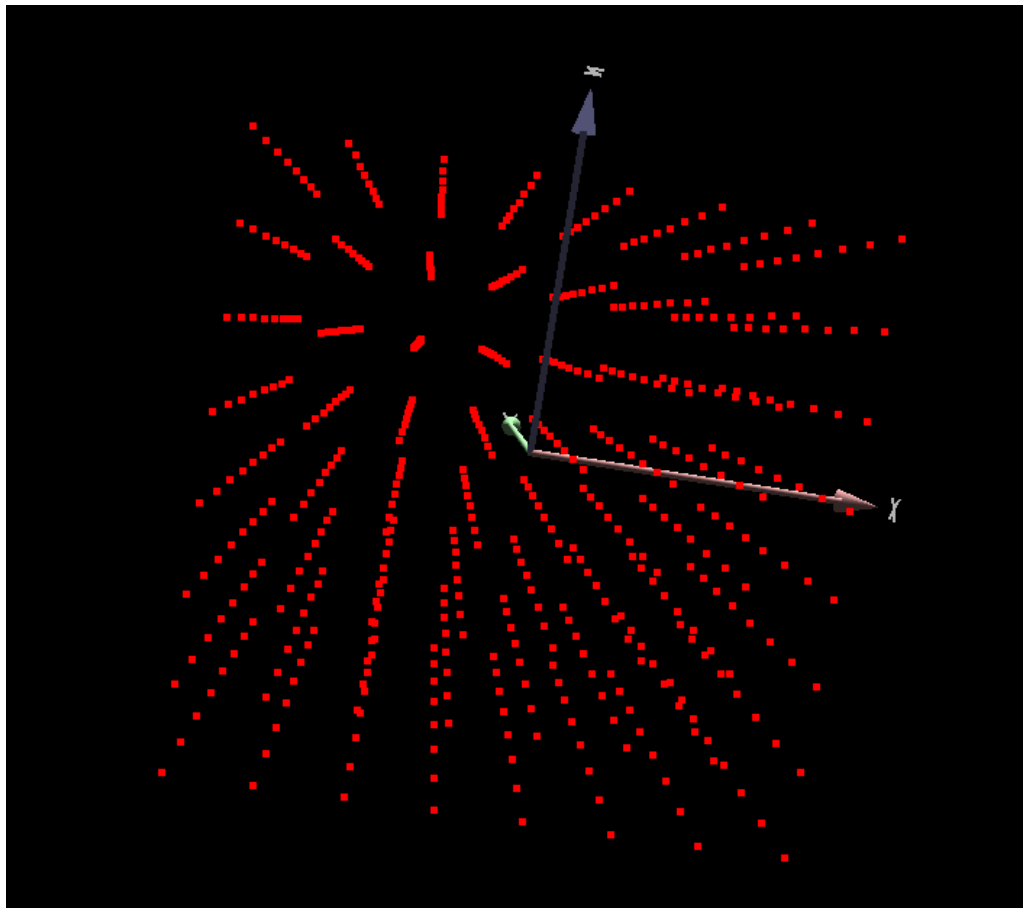


Figure: A 3 dimensional voxel grid where each node contains a truncated signed distance to the surface and the weight

Fusing the Depth Maps

- Measure the projective distance $\rho(x)$ between each vertex and the surface

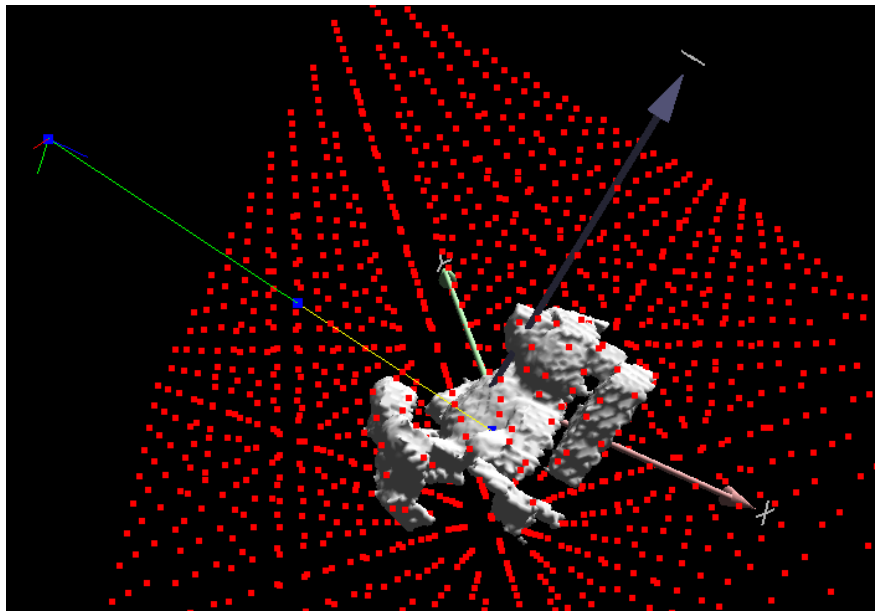


Figure: The projective distance is the distance between the vertex and the surface along the ray. The distance here is the yellow line

Fusing the Depth Maps

- To fuse the new information into the global model, a running weighted average is computed for each vertex:

$$D_{k+1}(x) = \frac{D_k(x)W_k(x) + d(x)_{k+1}w(x)_{k+1}}{W_k(x) + w(x)_{k+1}} \tag{1}$$

$$W_{k+1}(x) = W_{k+1}(x) + w_k(x) \tag{2}$$

where x is the vertex and $w_{k+1}(x)$ and $d_{k+1}(x)$ are the measured weight and distance for frame k+1.

Fusing the Depth Maps

- After that all images are fused into the SDF, a visualization is made
- The zero-crossing is extracted using an algorithm called Marching Cubes

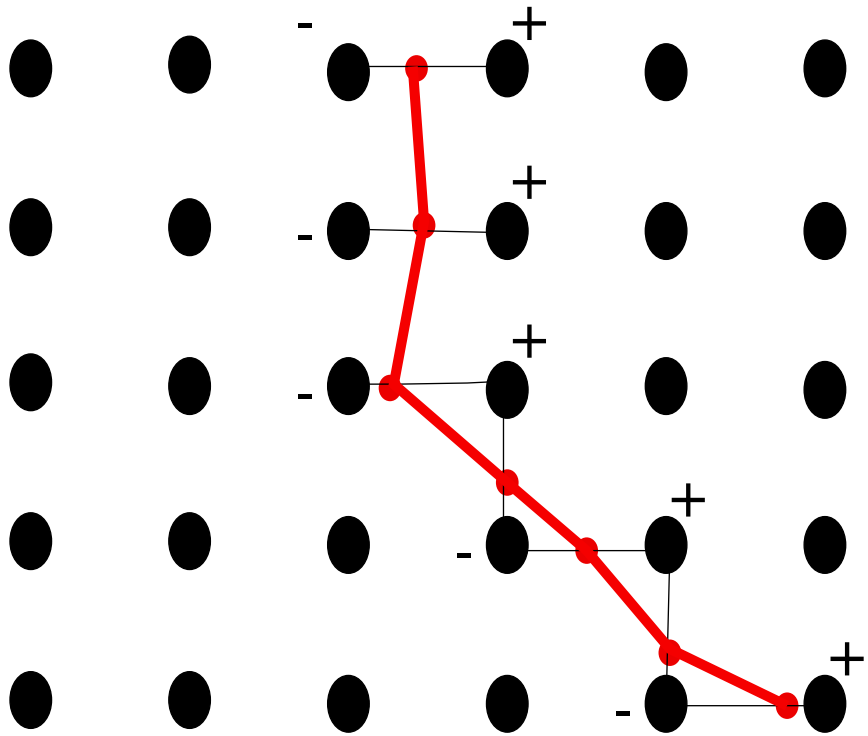


Figure: The principle of marching cube, the surface goes at the zero-crossing

Fusing the Depth Maps

- In 3D, the principle is the same
- Instead of using lines, triangles are drawn

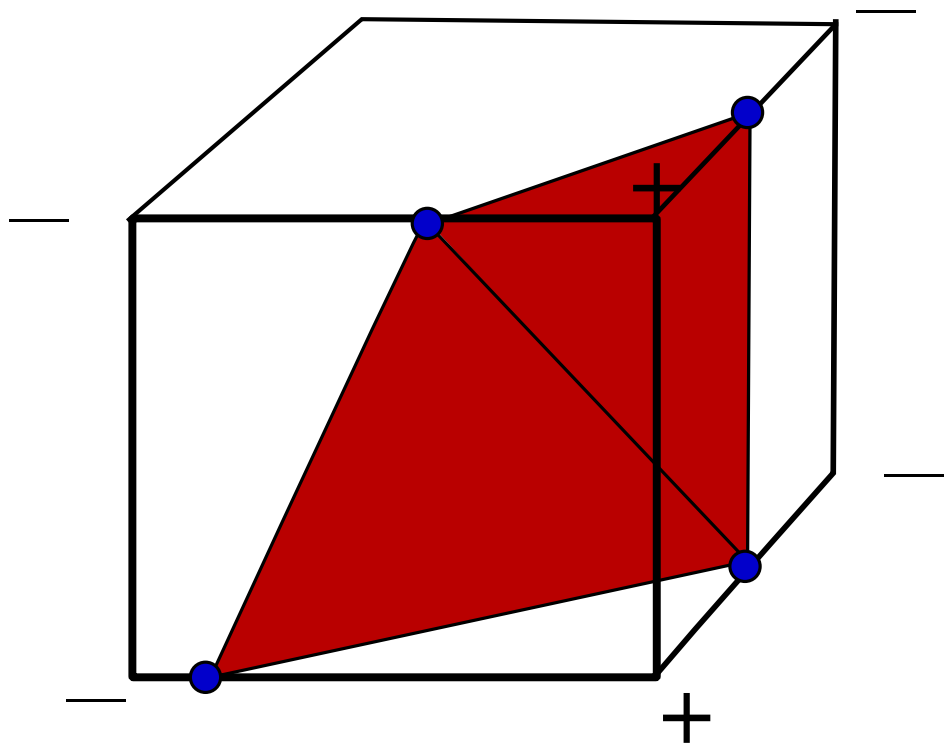


Figure: Example of how the triangles can be drawn

Fusing the Depth Maps

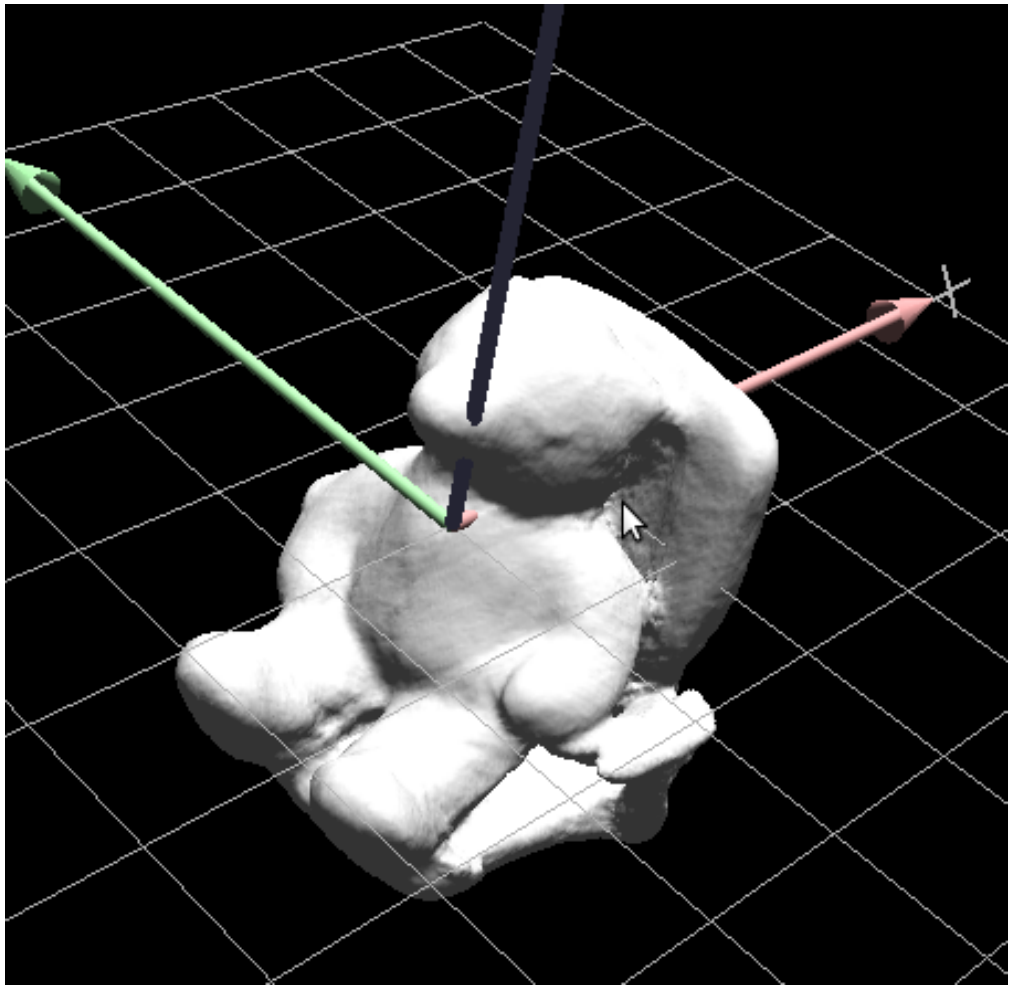


Figure: The result of 1376 depth maps of a teddybear

How To Track the Camera Against the Global Model

Camera Tracking

- Now we have a method for fusing the depth images into a global model when we have the cameras.
- The Signed Distance Function gives a projective point-to-point measure between each vertex and the surface.
- Define an error metric by using the SDF
- A form of projective point-to-point ICP

Camera Tracking

- ICP stands for Iterative Closest Point
- Given two point-clouds, minimize the error between them

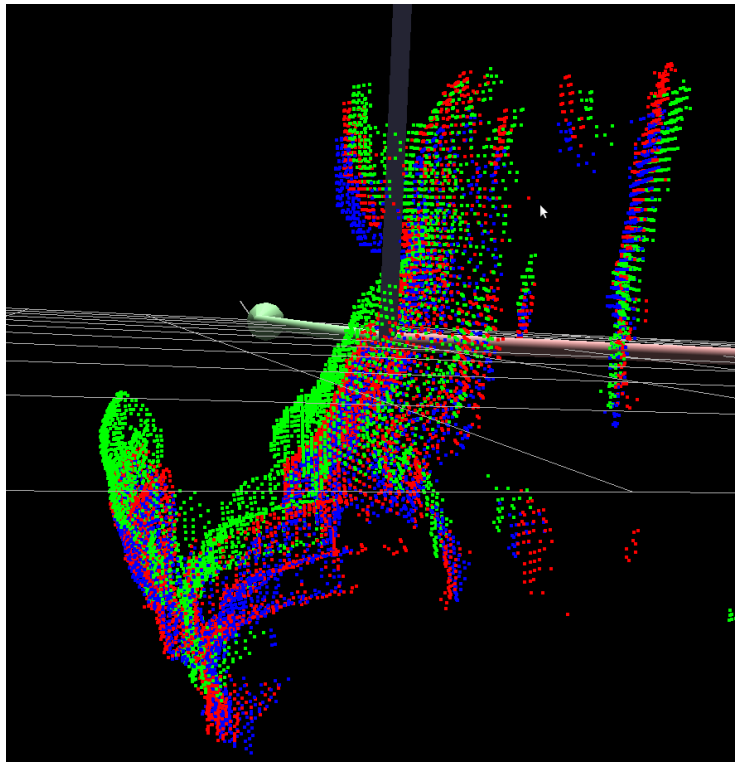


Figure: The red and green initial point-clouds. The blue cloud is the result of minimizing the error between the red and green

Camera Tracking

- From the model obtained from the first k cameras, find camera k+1 by minimizing the error.

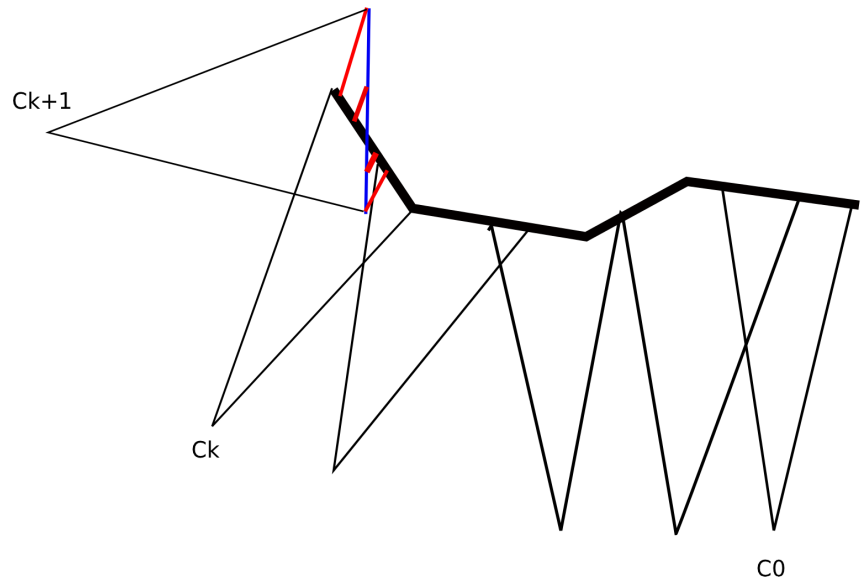
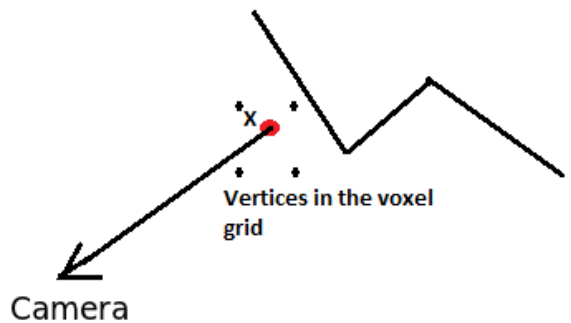


Figure: Aligning camera k+1 agianst the global model

Camera Tracking

$$E(\xi) = \frac{\sum_{i,j}(V(X(\xi, I_d(i,j))))^2}{M} \tag{3}$$

- M is the number of points in the grid
- V is the signed distance function
- ξ is a 6-dimensional vector representing the translation and rotation of the camera.
- $X(\xi, I_d(i,j))$ is a reconstructed point by using the depth value $I_d(i,j)$ and the current camera configuartion ξ



Camera Tracking

- Linearizing around the current camera guess ξ_0

$$V(\xi) \approx V(\xi_0) + \nabla V(\xi_0)(\xi - \xi_0) \tag{4}$$

- Plugging this approximation into the error function gives:

$$E(\xi) \approx \frac{\sum_{i,j}(V(\xi_0) + \nabla V(\xi_0)(\xi - \xi_0))^2}{M} \tag{5}$$

- This function we want to minimize, hence we want to solve $\nabla E(\xi) = 0$.

Camera Tracking

$$\nabla E(\xi) \approx \sum_{i,j} (2V(\xi_0)\nabla V(\xi_0) + 2\nabla V(\xi_0)\nabla V(\xi_0)^T(\xi - \xi_0) = 0 \tag{6}$$

$$(\sum_{i,j} (\nabla V(\xi_0)\nabla V(\xi_0)^T)\xi = -\nabla V(\xi_0) + (\sum_{i,j} (\nabla V(\xi_0)\nabla V(\xi_0)^T)\xi_0 \tag{7}$$

$$A\xi = b \tag{8}$$

$$\xi = A^{-1}b \tag{9}$$

RESULTS

RESULTS

- This has been tested against different benchmarks.

Absolute Trajectory Error (ATE)	Teddy	xyz
rmse [m]	0.091	0.023
mean error [m]	0.086	0.021
median error [m]	0.081	0.0184
error std [m]	0.030	0.011
abs trans error min [m]	0.015	0.0009
abs trans error max [m]	0.211	0.065

- It has also been tested for live-data.

(Loading Video...)

CONCLUSION

- Using live-data, fusion and tracking goes in real-time, as long as the tracking is not lost.
- Tracking works fine as long as camera movement are not to big.

FUTURE WORK

- Next step is to look at different error-metrics such as point-to-plane for instance.
- Looking at different norms, such as L1.
- Compairing to Kinect Fusion
- Speed up the code as much as possible.