# FPGA Architectures for Dense SLAM


Quentin Gautier, Alric Althoff, Ryan Kastner

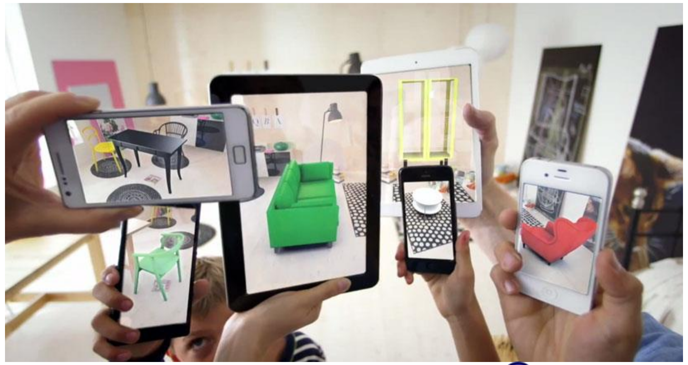
*University of California, San Diego*



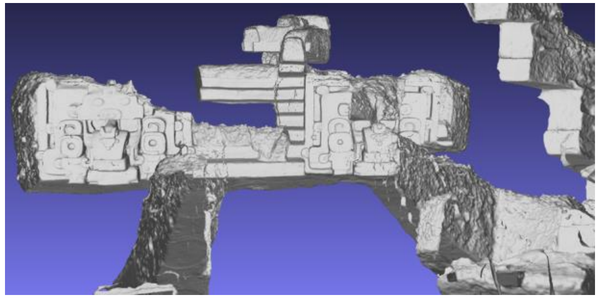
**ASAP 2019**

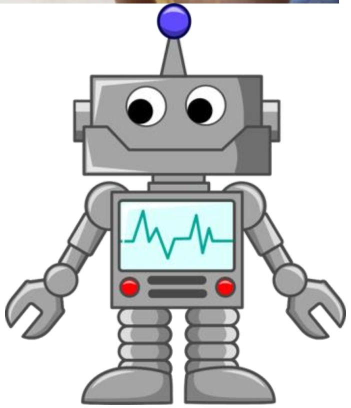
July 15th 2019

# Dense SLAM



- Simultaneous Localization And Mapping
- Robotics, Augmented Reality, Autonomous Driving, Archaeology, etc.
- Sparse SLAM
  - Map with sparse features
  - Mostly for tracking
- Dense SLAM
  - Full 3D model
  - Mostly for scanning





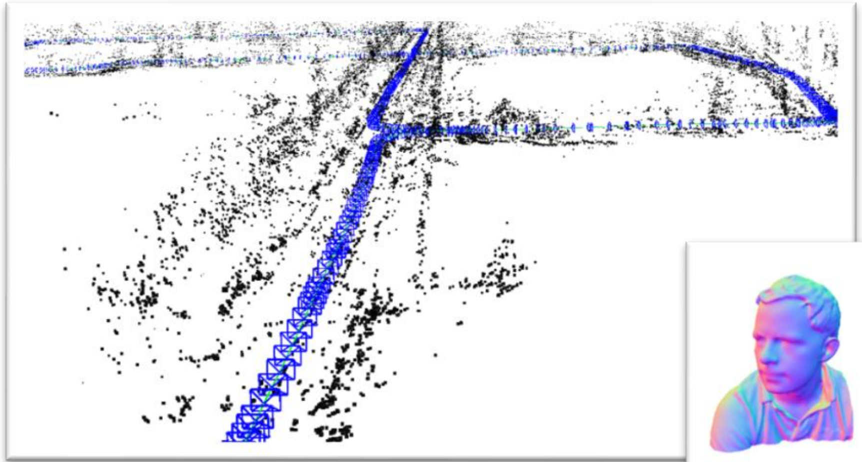


Illustration from: Krombach, Nicola, et al. "Feature-based visual odometry prior for real-time semi-dense stereo SLAM." *Robotics and Autonomous Systems* 109 (2018)

Newcombe, Richard A., et al. "Kinectfusion: Real-time dense surface mapping and tracking." *ISMAR*. 2011.

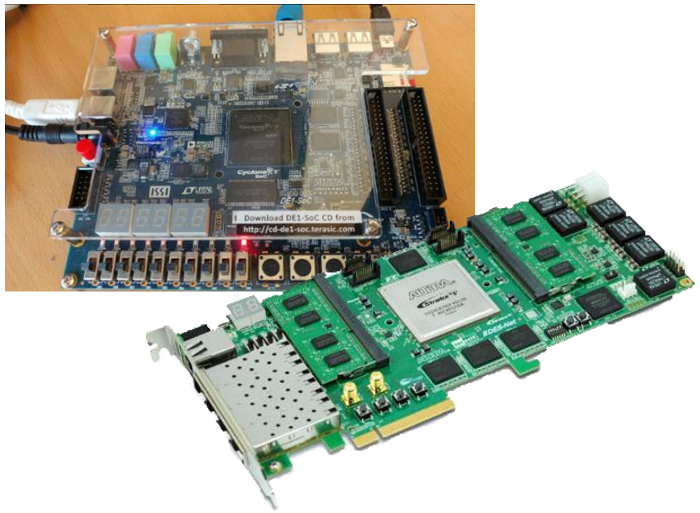
# Accelerating Dense SLAM on hardware

GPU

- KinectFusion [1]
- ElasticFusion [2]
- InfiniTAM [3]
- …

FPGA

- Kalman filter / Feature extraction [4,5]
- Semi-Dense SLAM [6]
- **Our work: Dense SLAM (InfiniTAM) on FPGA**

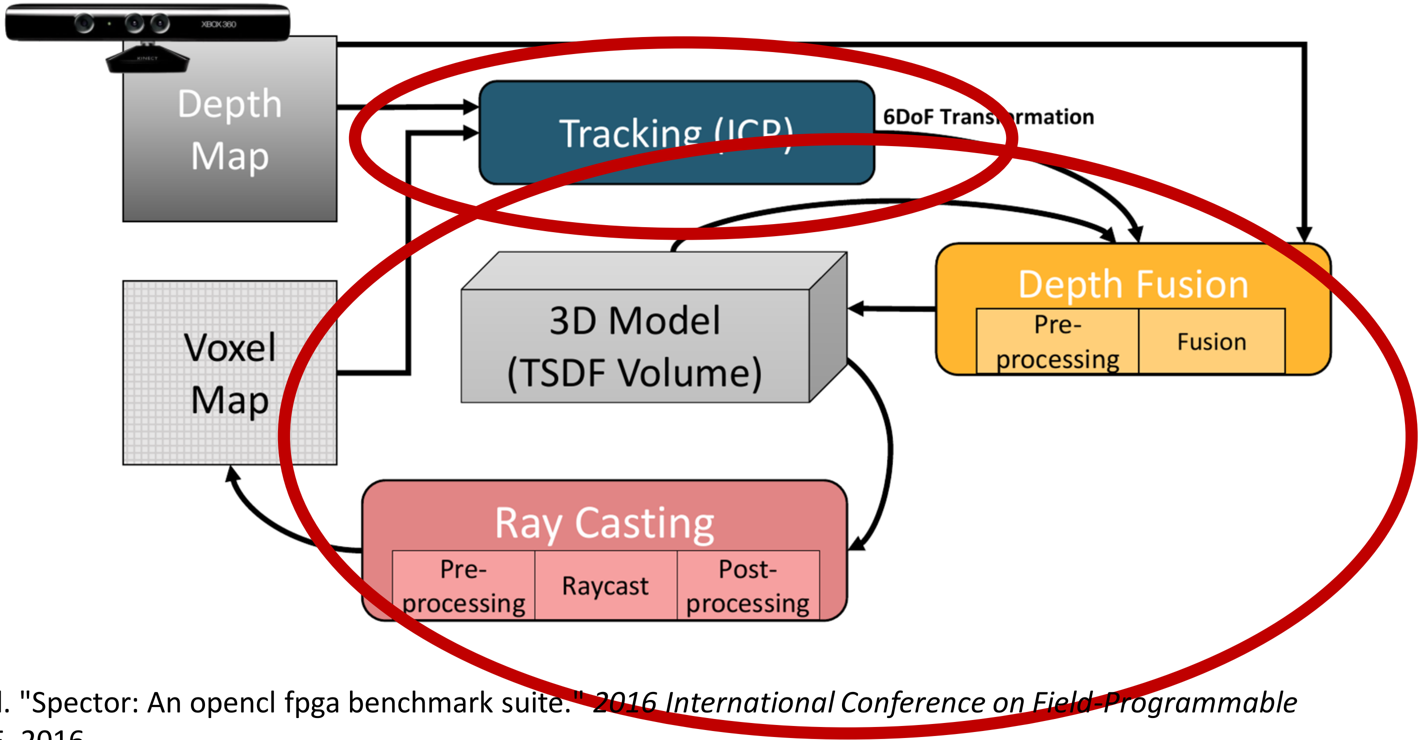


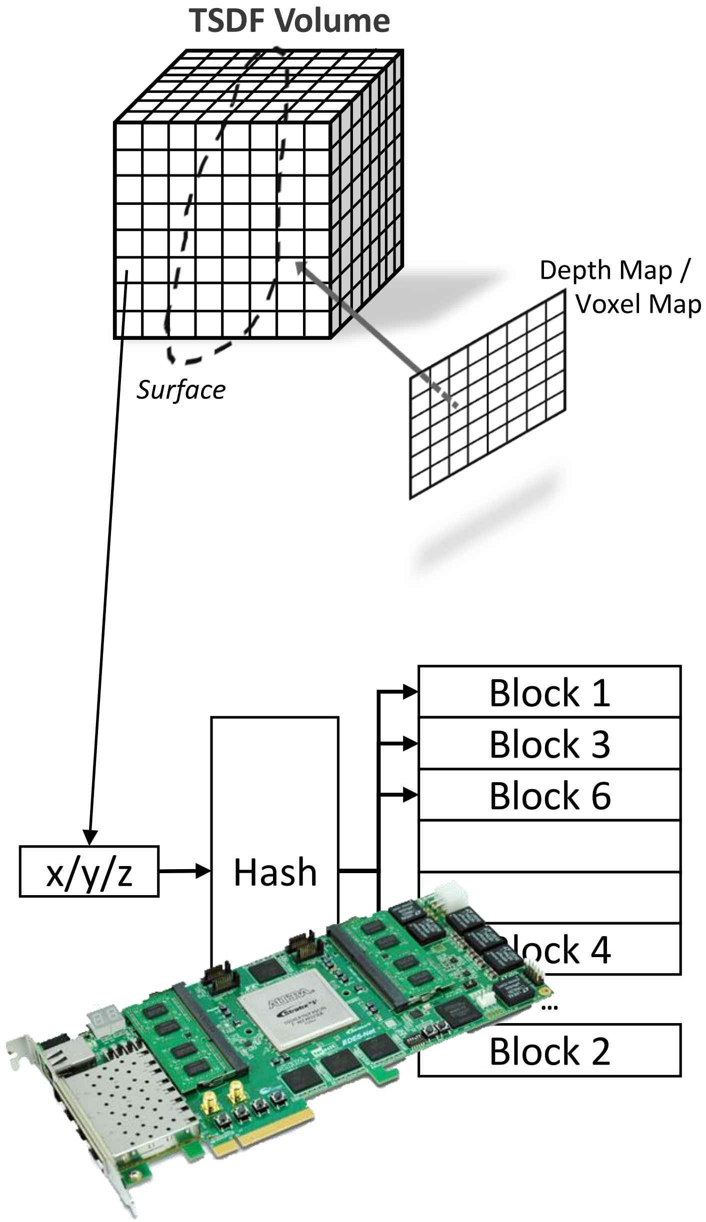

[1] Newcombe, Richard A., et al. "Kinectfusion: Real-time dense surface mapping and tracking." *ISMAR*. Vol. 11. No. 2011. 2011.
[2] Whelan, Thomas, et al. "ElasticFusion: Dense SLAM without a pose graph." Robotics: Science and Systems, 2015.
[3] Kähler, Olaf, et al. "Very high frame rate volumetric integration of depth images on mobile devices." *IEEE transactions on visualization and computer graphics* 21.11 (2015): 1241-1250.
[4] Tertei, Daniel Törtei, Jonathan Piat, and Michel Devy. "FPGA design and implementation of a matrix multiplier based accelerator for 3D EKF SLAM." *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14)*. IEEE, 2014.
[5] Fang, Weikang, et al. "FPGA-based ORB feature extraction for real-time visual SLAM." *2017 International Conference on Field Programmable Technology (ICFPT)*. IEEE, 2017.
[6] Boikos, Konstantinos, and Christos-Savvas Bouganis. "Semi-dense SLAM on an FPGA SoC." *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2016.

# InfiniTAM

## Real-time Dense SLAM

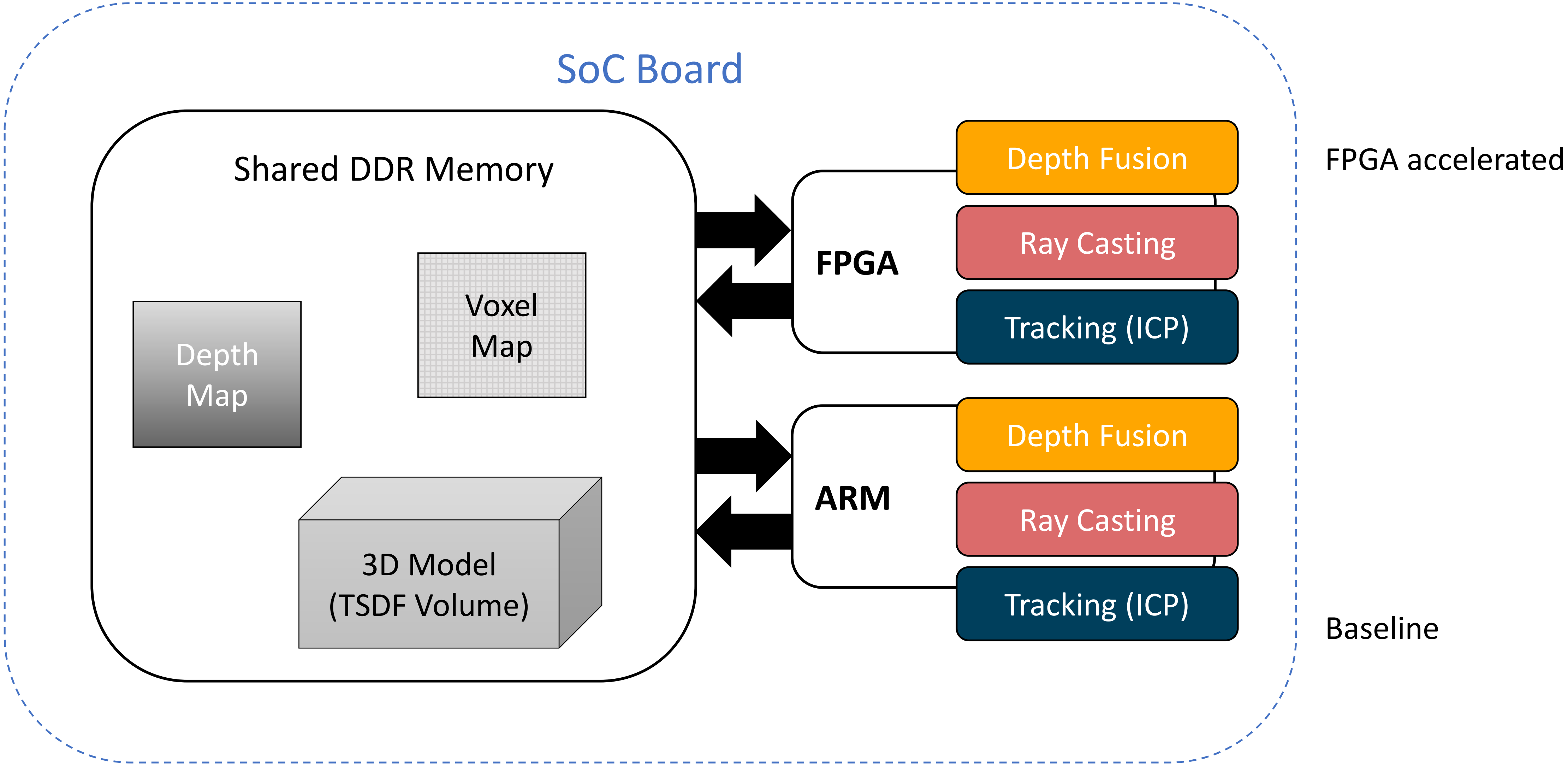
*with hash table*



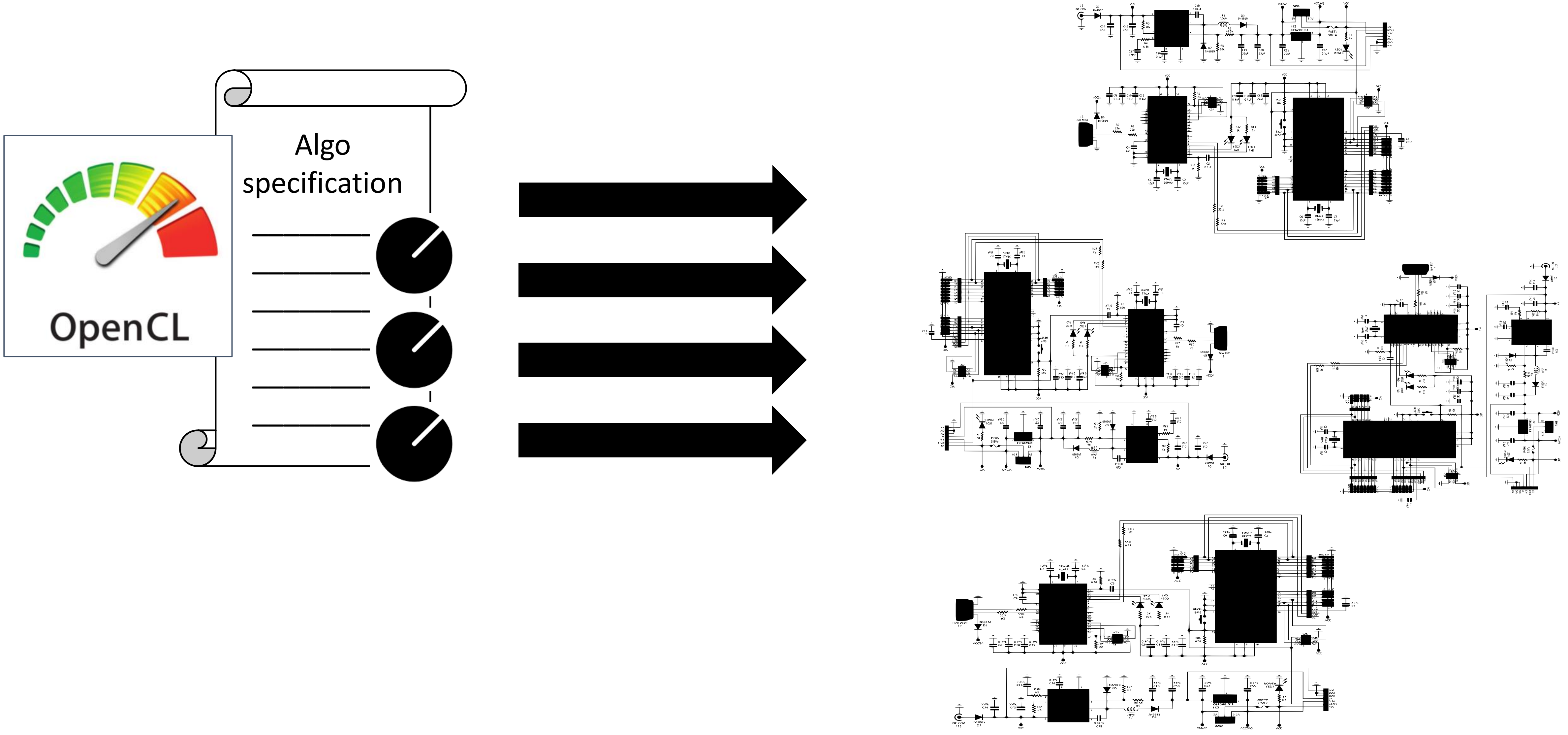




Gautier, Quentin, et al. "Spector: An opencl fpga benchmark suite." *2016 International Conference on Field-Programmable Technology (FPT)*. IEEE, 2016.

# InfiniTAM on FPGA



# FPGA architecture specification



# Depth Fusion

## Tunable Parameters (knobs)

- Loop
  - Work-Items vs. *for* loop (pipeline)
  - # of compute unit duplications
- Memory caching: automatic / manual
- Unrolling factors
- Placement of conditional
- Interpolation
- Etc.

*Algorithm specification in OpenCL*

**Input** : Depth Map; Visible blocks IDs; Hash table; Voxel Buffer; Truncation threshold $\mu$

**for** *each visible block ID* **do**

  Fetch block $B$ from hash table

  From $B$, fetch pointer to voxel block in voxel buffer

  **for** *each voxel* $V = (x, y, z)$ *in the voxel block* **do**

    $(w, d) \leftarrow$ Fetch weight and distance from $V$

    **if** $w > w_{max}$ **then** continue ;

    $(g_x, g_y, g_z) \leftarrow$ Calculate global coordinates of $V$

    $(c_x, c_y, c_z) \leftarrow (g_x, g_y, g_z)$ to camera view

    $(i, j) \leftarrow$ Project $(c_x, c_y, c_z)$ into 2D image

    $D \leftarrow$ Get depth at $(i, j)$

    **if** $|c_z - D| > \mu$ **then** continue ;

    $(w', d') \leftarrow$ Calculate new weight and distance

    Store new voxel $V \leftarrow (w', d')$

  **end**

**end**

# ICP

**Input :** Depth map; Voxel Map; Normal Map
Initialize $H_g$ and $\nabla_g$ to 0
**for** *each pixel (x,y)* **do**
  Fetch depth $D$ at $(x, y)$
  **if** $D = 0$ **then** continue ;
  $p_{cur} \leftarrow$ Transform $D$ with current estimated pose
  $(i, j) \leftarrow$ Project $p_{cur}$ into 2D view
  **if** $(i, j)$ *not valid* **then** continue ;
  $p_{prev} \leftarrow$ Fetch 3D point from Voxel Map at $(i, j)$
  **if** $p_{prev}$ *not valid* **then** continue ;
  **if** $distance(p_{prev}, p_{cur}) > threshold$ **then** continue ;
  $n_{prev} \leftarrow$ Fetch 3D normal at $(i, j)$
  Calculate $H_l$ and $\nabla_l$
  Accumulate $H_l$ and $\nabla_l$ into $H_g$ and $\nabla_g$
**end**
Save $H_g$ and $\nabla_g$ into global memory
**Output:** $H_g$ and $\nabla_g$

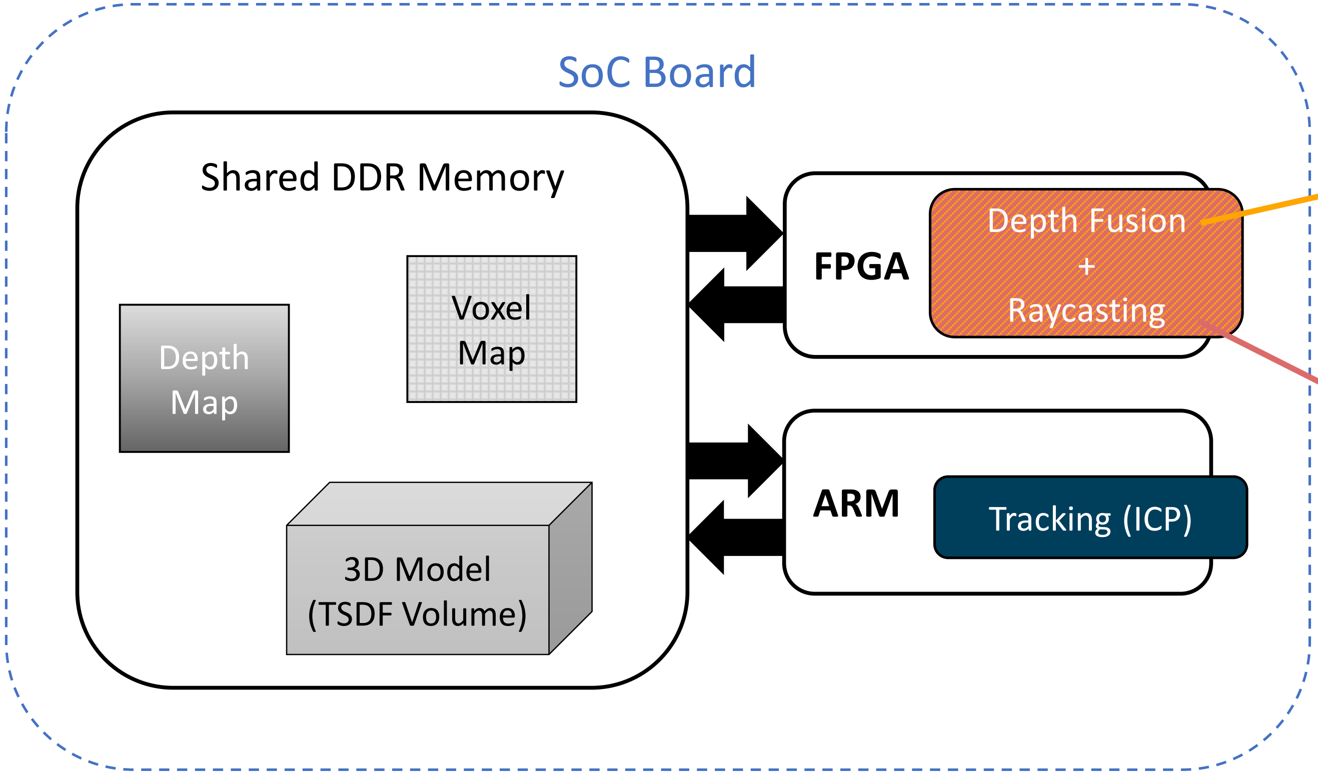
Tunable Knobs

# Raycasting

**Input :** Min/Max map; Hash table; Voxel Buffer
**for** *each pixel (x,y)* **do**
  Fetch start/end depth $(D_S, D_E)$ from Min/Max map
  Convert $(D_S, D_E)$ to global coordinates $(S_G, E_G)$
  Calculate ray direction $(E_G - S_G)$ and norm
  **while** *Starting from $S_G$, until $E_G$* **do**
    At current point $(x, y, z)$:
    $d \leftarrow$ ReadDistanceUninterpolated()
    **if** *no voxel or no distance* **then**
      $(x, y, z)$ += one block size along ray
      Continue
    **else if** $d > (0 + \epsilon)$ **then**
      $(x, y, z)$ += $d$ along ray; Continue
    **else if** $d < (0 + \epsilon)$ **then**
      $d \leftarrow$ ReadDistanceInterpolated()
    **end**
    **if** $d < 0$ **then** Break ;
  **end**
  **if** $d < 0$ **then**
    $(x, y, z)$ += $d$ along ray
    $d \leftarrow$ ReadDistanceInterpolated()
    $(x, y, z)$ += $d$ along ray
    Save $(x, y, z)$ position into Voxel Map
  **end**
**end**
**Output:** Voxel Map

# Depth Fusion + Raycasting

- Combined algorithm with custom data structure




**Input** : Depth Map; Visible blocks IDs; Hash table; Voxel Buffer; Truncation threshold $\mu$

**for** *each visible block ID* **do**
  Fetch block $B$ from hash table
  From $B$, fetch pointer to voxel block in voxel buffer
  **for** *each voxel* $V = (x, y, z)$ *in the voxel block* **do**
    $(g_x, g_y, g_z) \leftarrow$ Calculate global coordinates of $V$
    $(c_x, c_y, c_z) \leftarrow (g_x, g_y, g_z)$ to camera view
    $(i, j) \leftarrow$ Project $(c_x, c_y, c_z)$ into 2D image
    $D \leftarrow$ Get depth at $(i, j)$
    $(w, d) \leftarrow$ Fetch weight and distance from $V$
    **if** $w \leqslant w_{max}$ *or* $|c_z - D| \leqslant \mu$ **then**
      $(w', d') \leftarrow$ New weight and distance
      Store new voxel $V \leftarrow (w', d')$
    **end**
    $d \leftarrow$ Fetch distance from $V$
    **if** $|d| <$ *ProjectionMap[i,j]* **then**
      $c_z \leftarrow c_z + d$
      $(x, y, z) \leftarrow (c_x, c_y, c_z)$ to global coord.
      Save $(x, y, x)$ position into Voxel Map
      ProjectionMap[i,j] $\leftarrow |d|$
    **end**
  **end**
**end**

# Implementation

## Platforms

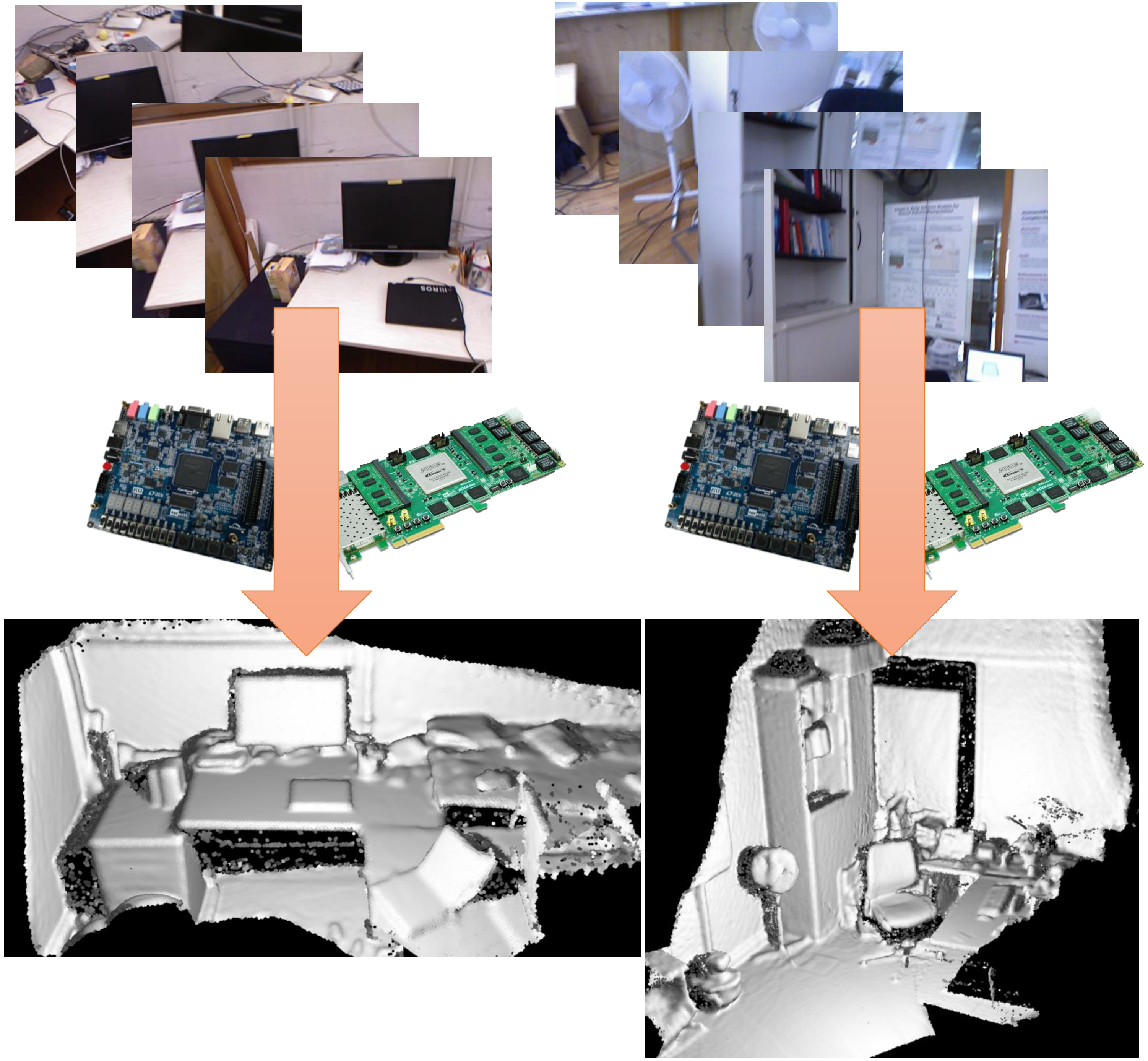
- DE1 FPGA SoC board
- DE5 FPGA PCI-e board

## Datasets

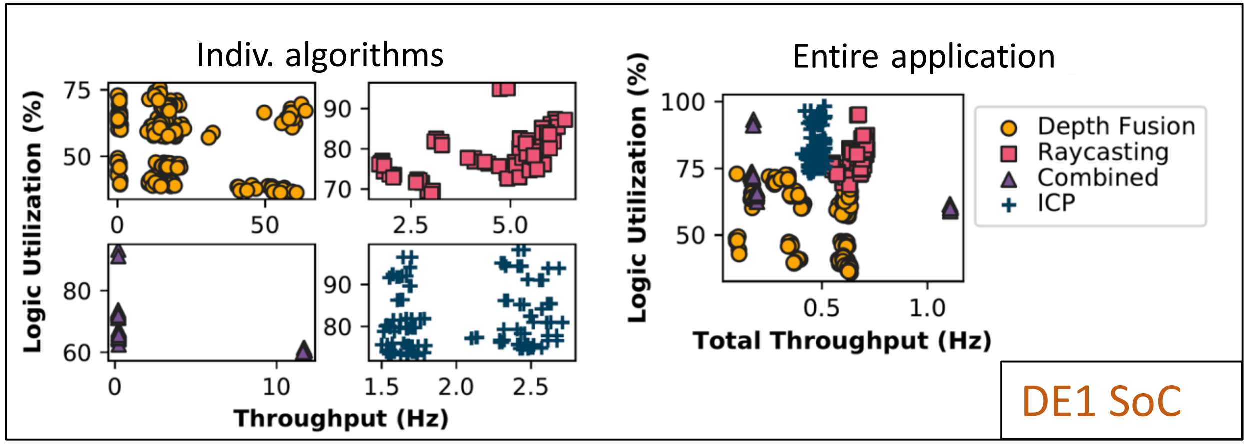
- TUM (320 x 240)
- Cave (320 x 180)

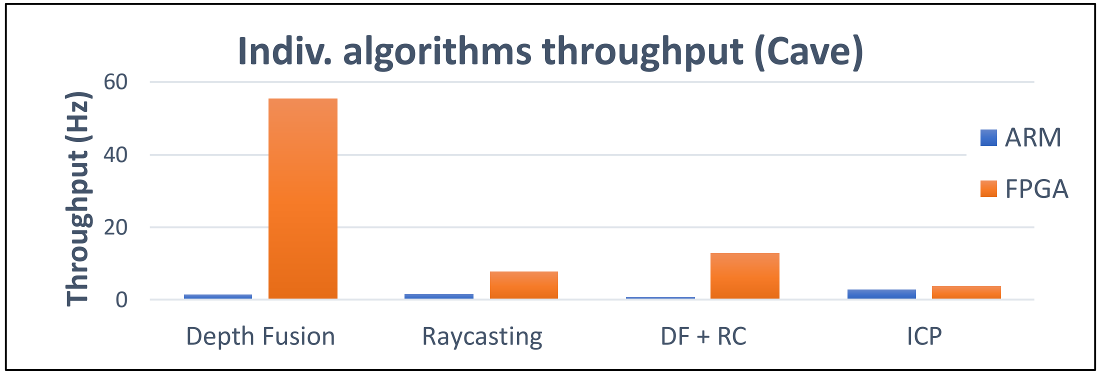
*End-to-end runtime*

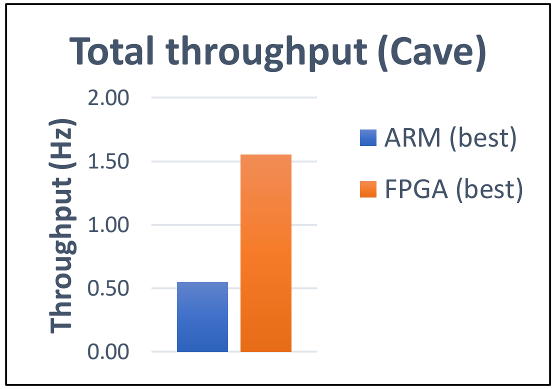
*+ Logic Utilization*

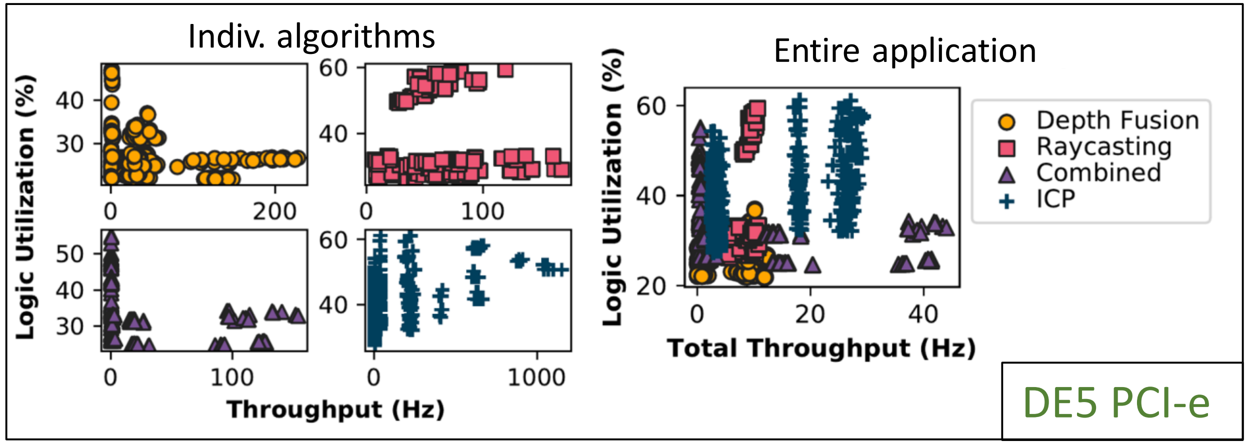


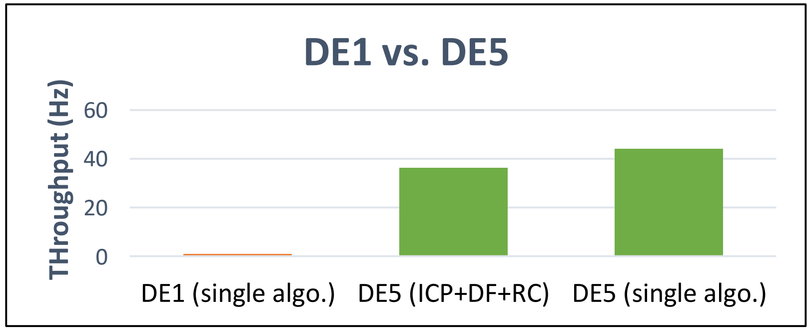
# Results


Indiv. algorithms
Entire application
Logic Utilization (%)
Throughput (Hz)
Total Throughput (Hz)
Depth Fusion
Raycasting
Combined
ICP
DE1 SoC


Indiv. algorithms throughput (Cave)
Throughput (Hz)
ARM
FPGA
Depth Fusion
Raycasting
DF + RC
ICP

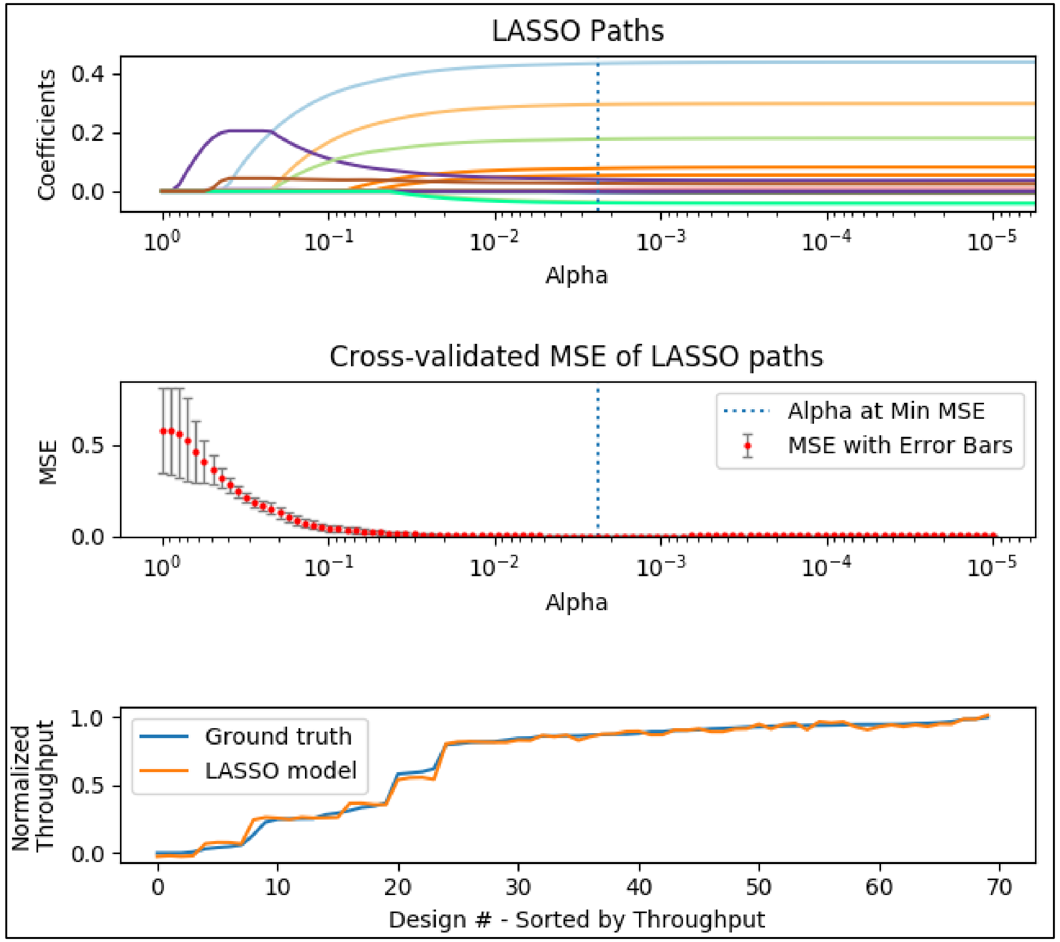

Total throughput (Cave)
Throughput (Hz)
ARM (best)
FPGA (best)


Indiv. algorithms
Entire application
Logic Utilization (%)
Throughput (Hz)
Total Throughput (Hz)
Depth Fusion
Raycasting
Combined
ICP
DE5 PCI-e


DE1 vs. DE5
THroughput (Hz)
DE1 (single algo.)
DE5 (ICP+DF+RC)
DE5 (single algo.)

# Design Spaces Analysis

## *LASSO Analysis*




| **Depth Fusion** (Min MSE = 0.0075) | | **Raycasting** (Min MSE = 0.0010) | |
|---|---|---|---|
| Knob | Coef | Knob | Coef |
| $\text{XyzLoop}^2$ | 0.130 | UseWI | 0.261 |
| $\text{CacheVoxels}^2$ | 0.067 | $\text{VoxelClcache}^2$ | 0.152 |
| CacheVoxels | 0.031 | Minmax | 0.075 |
| $\text{XyzUnroll}^2$ | 0.031 | IndexCache,UseWi | 0.071 |
| CacheVoxels,XyzLoopFlat | -0.023 | $\text{HashClcache}^2$ | 0.055 |
| **ICP** (Min MSE = 0.0054) | | **Combined** (Min MSE = $10^{-6}$) | |
| HessianUnroll | [illegible] | [illegible] | [illegible] |
| NablaUnroll | [illegible] | [illegible] | [illegible] |
| Branch NablaSumty | [illegible] | [illegible] | [illegible] |
| NablaSumtype | [illegible] | [illegible] | [illegible] |
| HessianUnroll,Nabl | [illegible] | [illegible] | [illegible] |

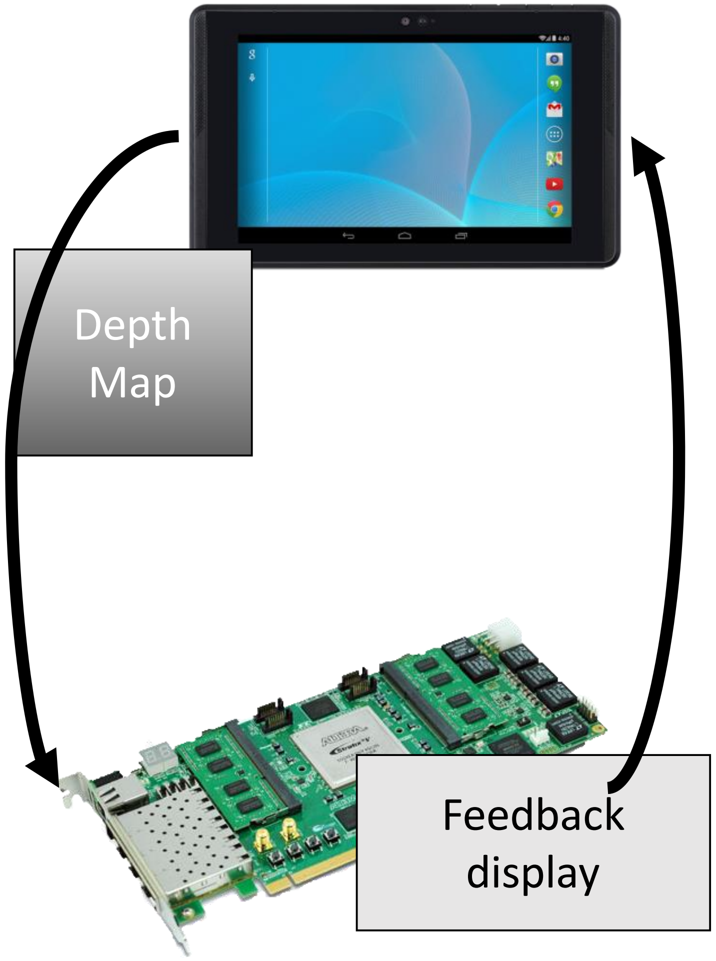
| **Depth Fusion** (Min MSE = 0.010) | | **Raycast** (Min MSE = 0.003) | |
|---|---|---|---|
| Knob | Coef | Knob | Coef |
| $\text{XyzLoop}^2$ | 0.047 | $\text{HashCLCache}^2$ | 0.152 |
| XyzLoop,DepthLocal | -0.039 | UseWI | 0.116 |
| ComputeUnits,DepthLocal | -0.039 | Minmax | 0.088 |
| EntryidLoop,DepthLocal | -0.034 | $\text{IndexCache}^2$ | 0.085 |
| CacheVoxels | 0.029 | $\text{Minmax}^2$ | 0.055 |
| **ICP** (Min MSE = 0.007) | | **Combined** (Min MSE = 0.0016) | |
| HessianSumtype | 0.056 | $\text{XyzLoop}^2$ | 0.061 |
| NablaSumtype | 0.054 | SdfLocal,XyzLoop | -0.024 |
| NablaSumtype,ShiftRegister | 0.047 | SdfLocal,XyzUnroll | -0.012 |
| HessianUnroll | 0.038 | SdfLocal,XyzFlat | 0.012 |
| HessianUnroll,HessianSumtype | 0.033 | XyzUnroll | -0.010 |

Loop implementation / pipelining

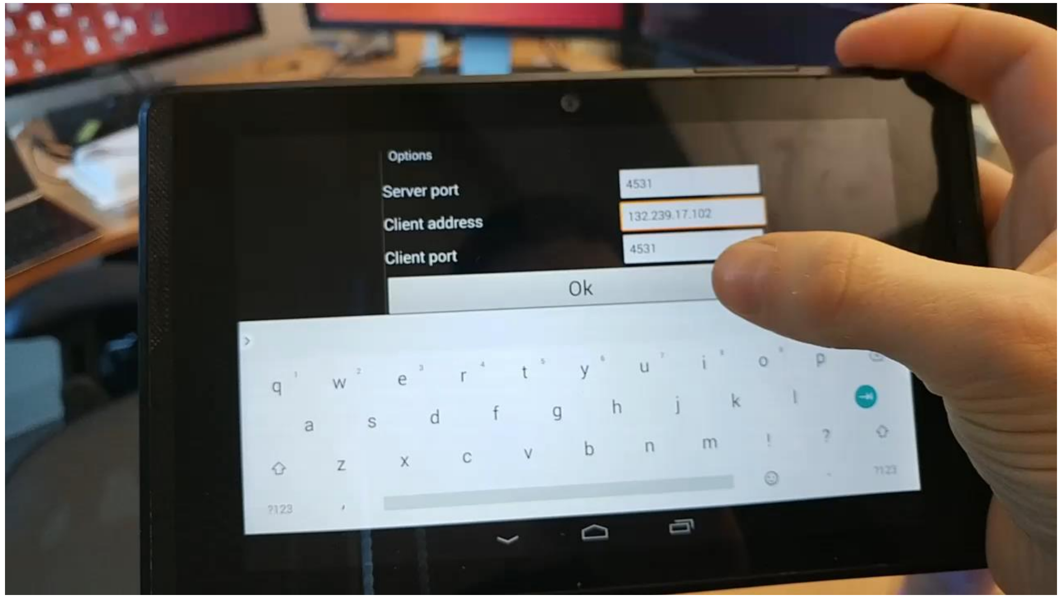
Memory caching

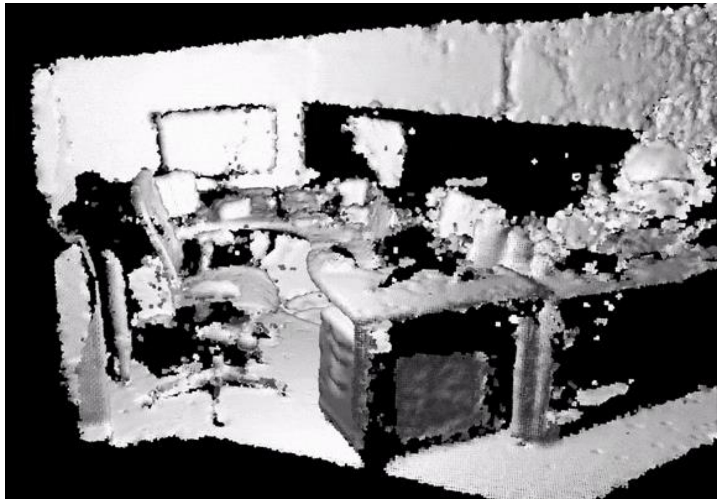
Other computation knobs

# Google Tango Demo

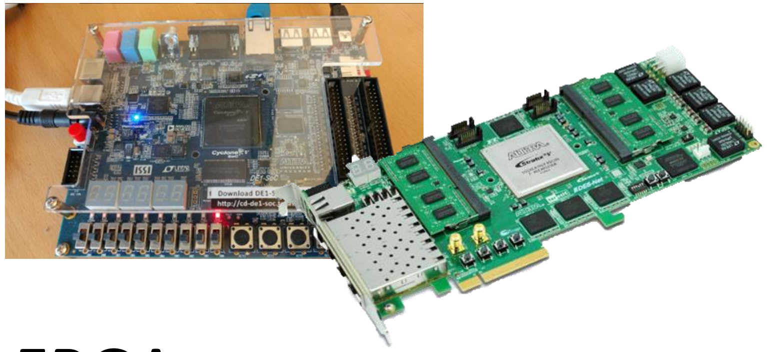



*Depth Fusion + Raycasting + ICP*
*on PCI-e FPGA*



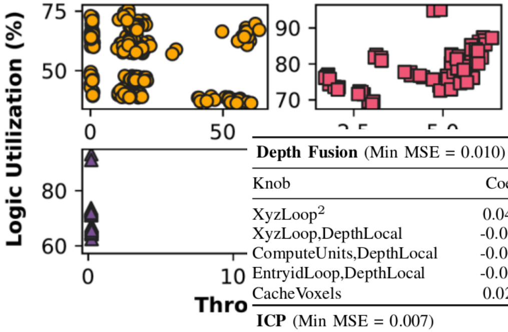



# Conclusion



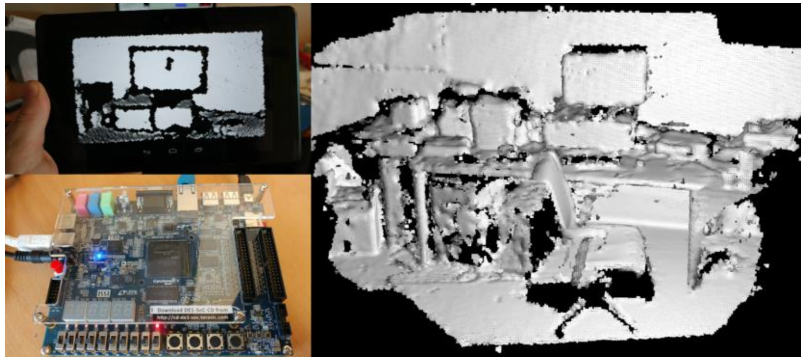
- Explored different dense SLAM architectures on FPGA
  - FPGA SoC and PCI-e FPGA

- Parameterized algorithms for knob analysis




| **Depth Fusion** (Min MSE = 0.010) | | **Raycast** (Min MSE = 0.003) | |
|---|---|---|---|
| Knob | Coef | Knob | Coef |
| $XyzLoop^2$ | 0.047 | $HashCLCache^2$ | 0.152 |
| XyzLoop,DepthLocal | -0.039 | UseWI | 0.116 |
| ComputeUnits,DepthLocal | -0.039 | Minmax | 0.088 |
| EntryidLoop,DepthLocal | -0.034 | $IndexCache^2$ | 0.085 |
| CacheVoxels | 0.029 | $Minmax^2$ | 0.055 |
| **ICP** (Min MSE = 0.007) | | **Combined** (Min MSE = 0.0016) | |
| HessianSumtype | 0.056 | $XyzLoop^2$ | 0.061 |
| NablaSumtype | 0.054 | SdfLocal,XyzLoop | -0.024 |
| NablaSumtype,ShiftRegister | 0.047 | SdfLocal,XyzUnroll | -0.012 |
| HessianUnroll | 0.038 | SdfLocal,XyzFlat | 0.012 |
| HessianUnroll,HessianSumtype | 0.033 | XyzUnroll | -0.010 |

- Dense SLAM runs on FPGA
  → Needs medium-sized FPGA SoC

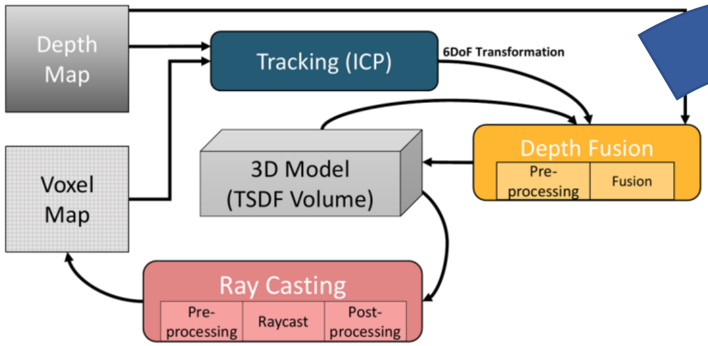


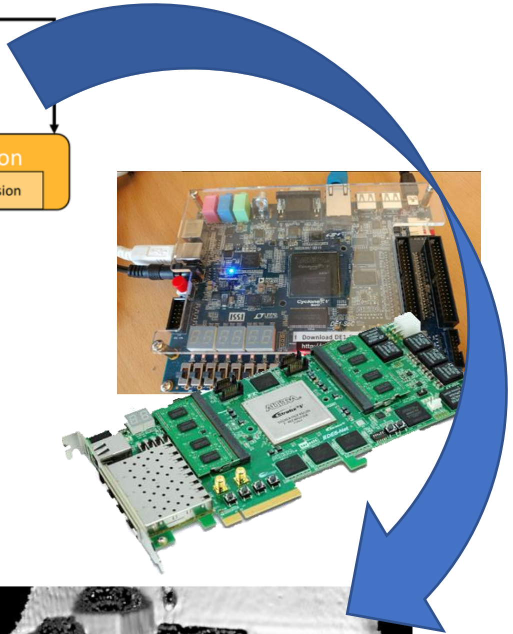
**More on GitHub:**
https://github.com/KastnerRG/infinitam_fpga

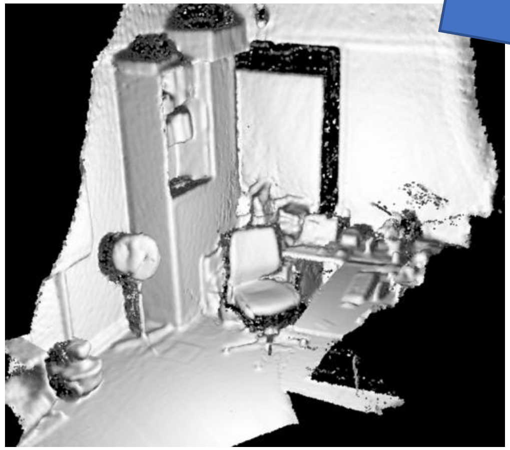
# FPGA Architectures for Dense SLAM

## Acknowledgments

Kastner Research Group





## Questions?

https://github.com/KastnerRG/infinitam_fpga