Multi-Camera 3D Fusion with BlenDR

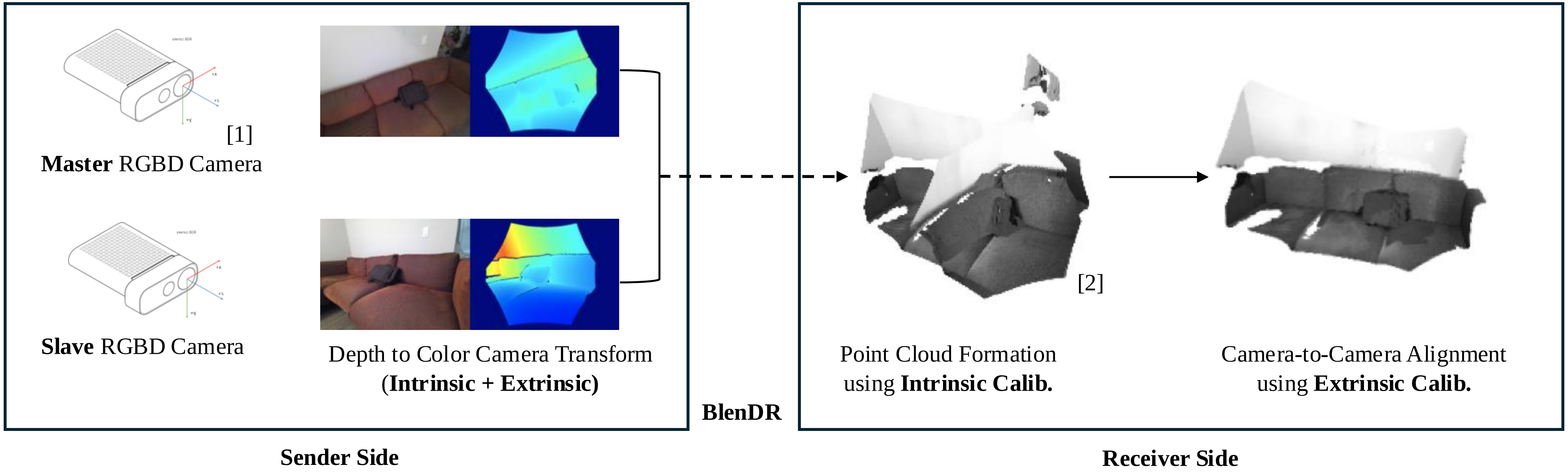
Joon Ha Kim

20180897

Contents

Project Details

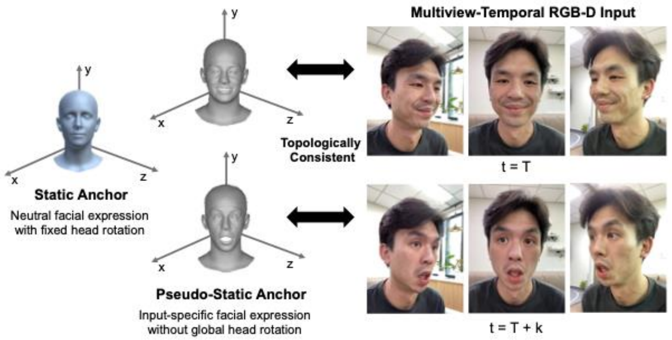
- **Goal:** Fuse multi-view point clouds to transmit (using BlenDR) a dense point cloud for improved spatial and temporal consistency
- **Key Terms:** Master/Slave Camera, Intrinsic/Extrinsic Calibration



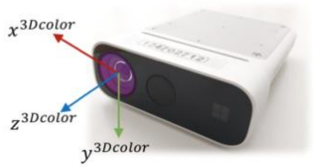
Previous Works on Point Cloud Fusion

- Human Body Tracking
  - Human Limb Anchoring[3] [4]
  - *FarFetch Fusion*: Human Face Anchoring[5]

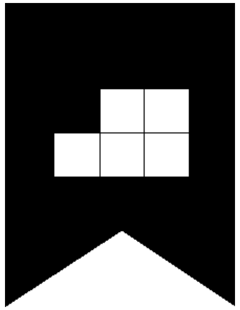
- Calibration Board
  - *LiveScan3D*: Custom Calibration Markers[6]
    - ➔ ***Stereo Calibration*:**
      use of cameras' *intrinsic* and *extrinsic* parameters
      derived from calibration markers
    - ➔ ***ICP Refinement*:** next slide…



[5]



[4]



[6]

General Trend: **Calibration** (either method above) + **Refinement**

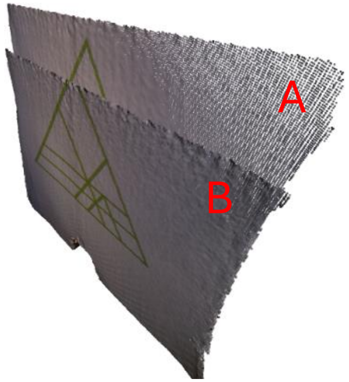
Iterative Closest Point Algorithm (ICP)

- Summary of ICP (Colored) [6]
  1. Start with initial guess transformation, T^0
  2. For each point in point cloud, find correspondence points, *K*, based on both spatial proximity and color similarity.

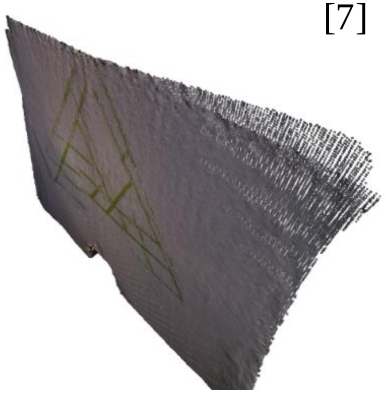
     → Use Euclidean distance for difference
  3. Calculate the transformation that minimizes a cost function (Least-Squares Fitting Function)
  4. Apply this transformation to the source point cloud and repeat until convergence or until maximum number of iterations.

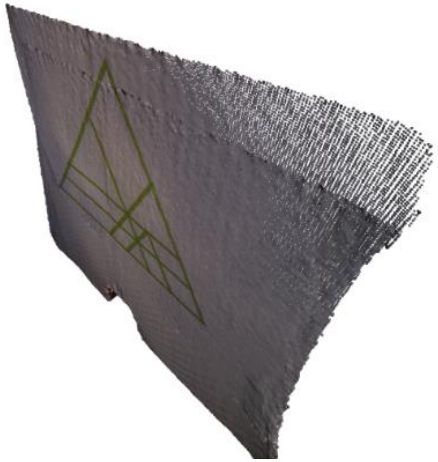
Function used:
open3d::pipelines::registration::RegistrationColoredICP()

[7]

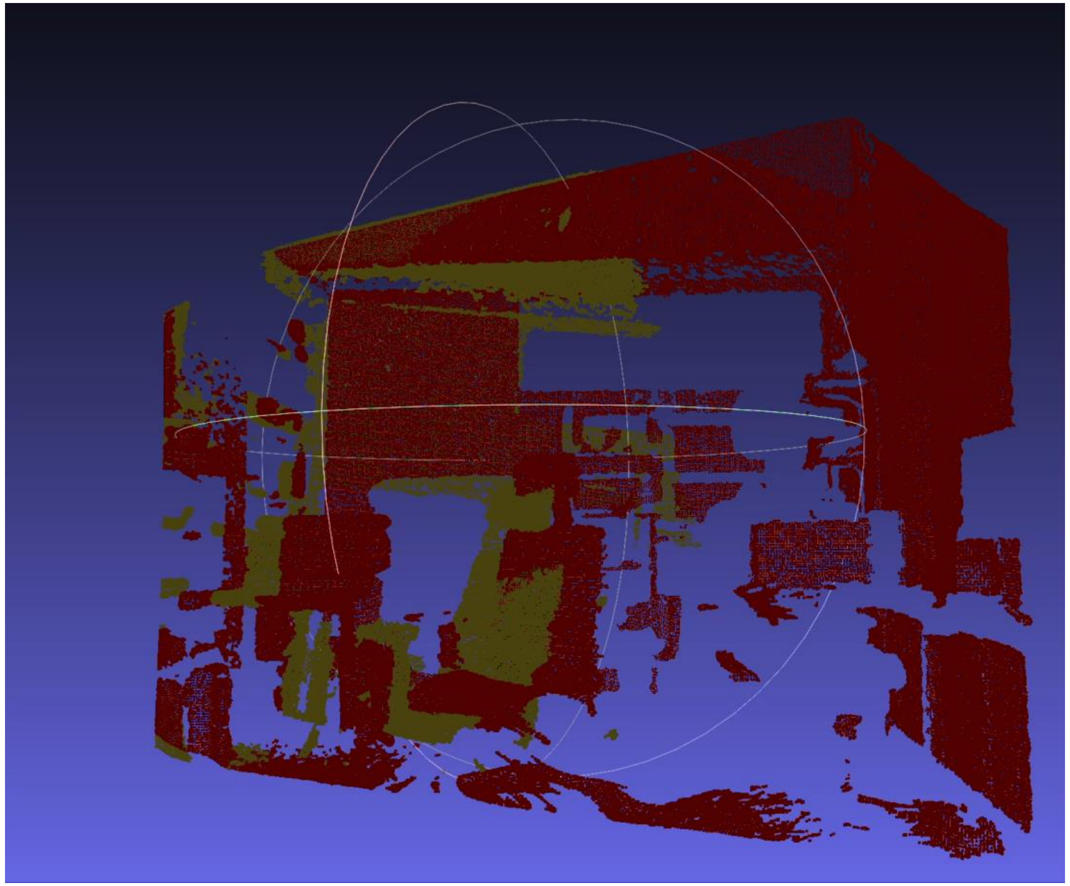


Point-to-Point ICP

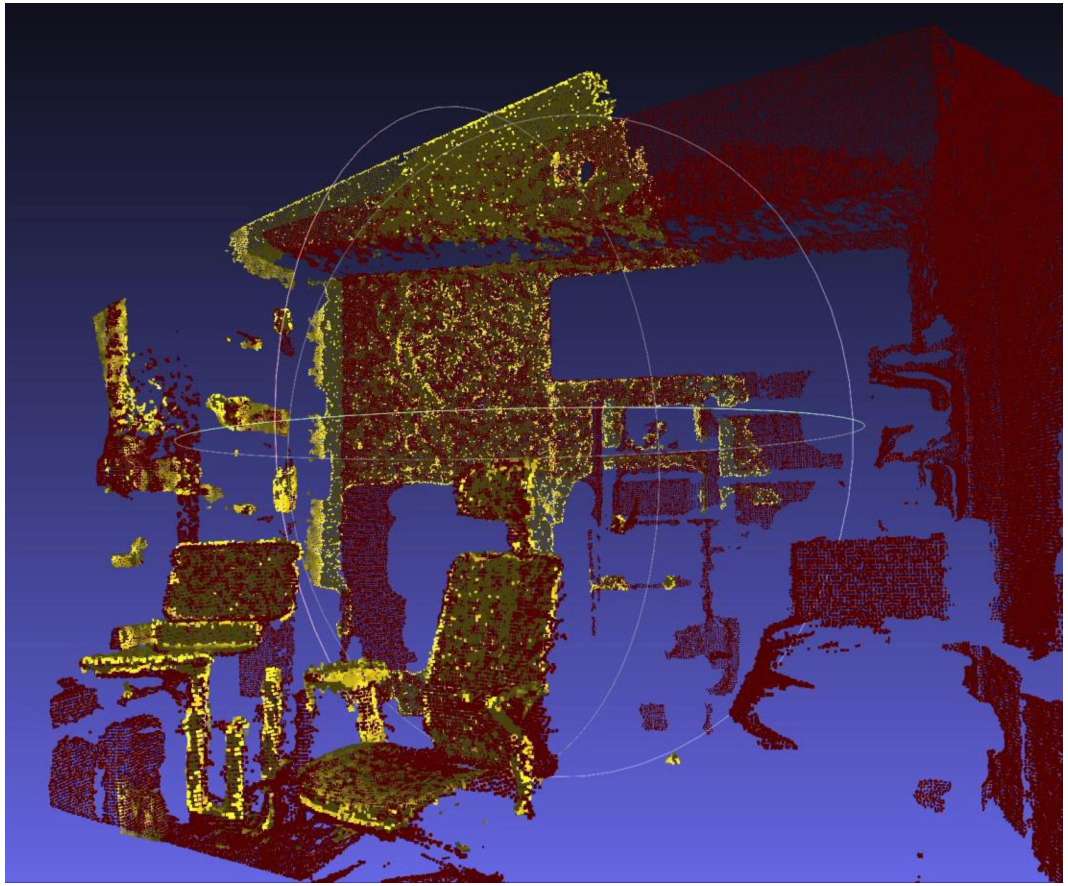

Point-to-Plane ICP


Colored ICP

ICP Ablation Study



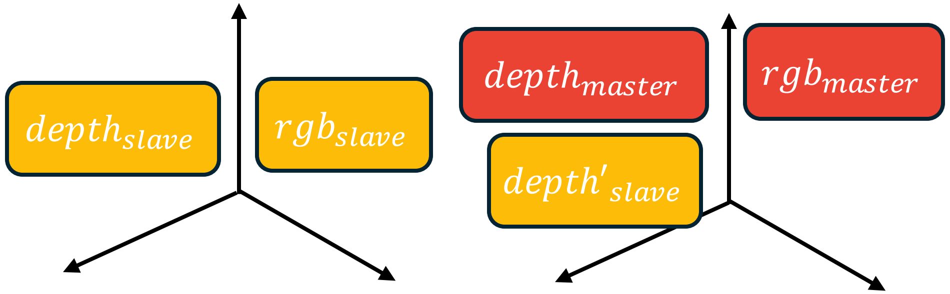
Point Cloud Reconstruction (No ICP)



Point Cloud Reconstruction (With ICP)

Attempt 1: Mapping Slave to Master Camera

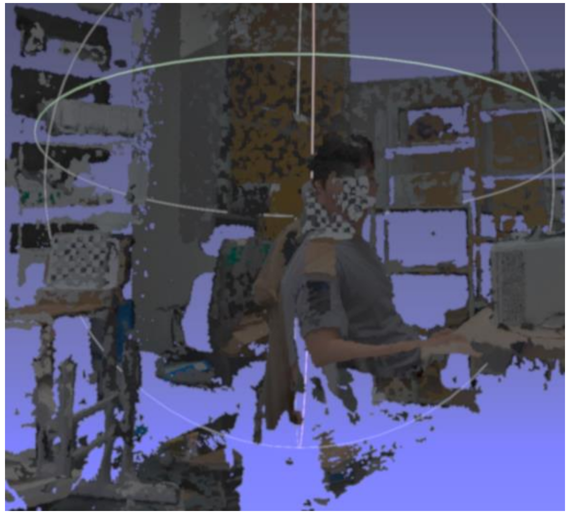
- Summary of Attempt 1:
  1. Retrieve device to device color-to-color calibration using the checkerboard calibration scheme
  2. Create custom transformation from **slave depth** to **master color camera** coordinates
  3. Create point cloud of modified **slave depth** and **master depth**
  4. Apply color to the point cloud + ICP
- **Problem 1**: mapping **slave color** to *modified* slave depth
- **Problem 2:** *final point cloud limited to master color perspective*



Slave Color Coordinates

Master Color Coordinates

Calibration using Checkboard (slave and master)



`k4a::calibration::convert_2d_to_3d`

Attempt 2: Virtual Camera (Pinhole)

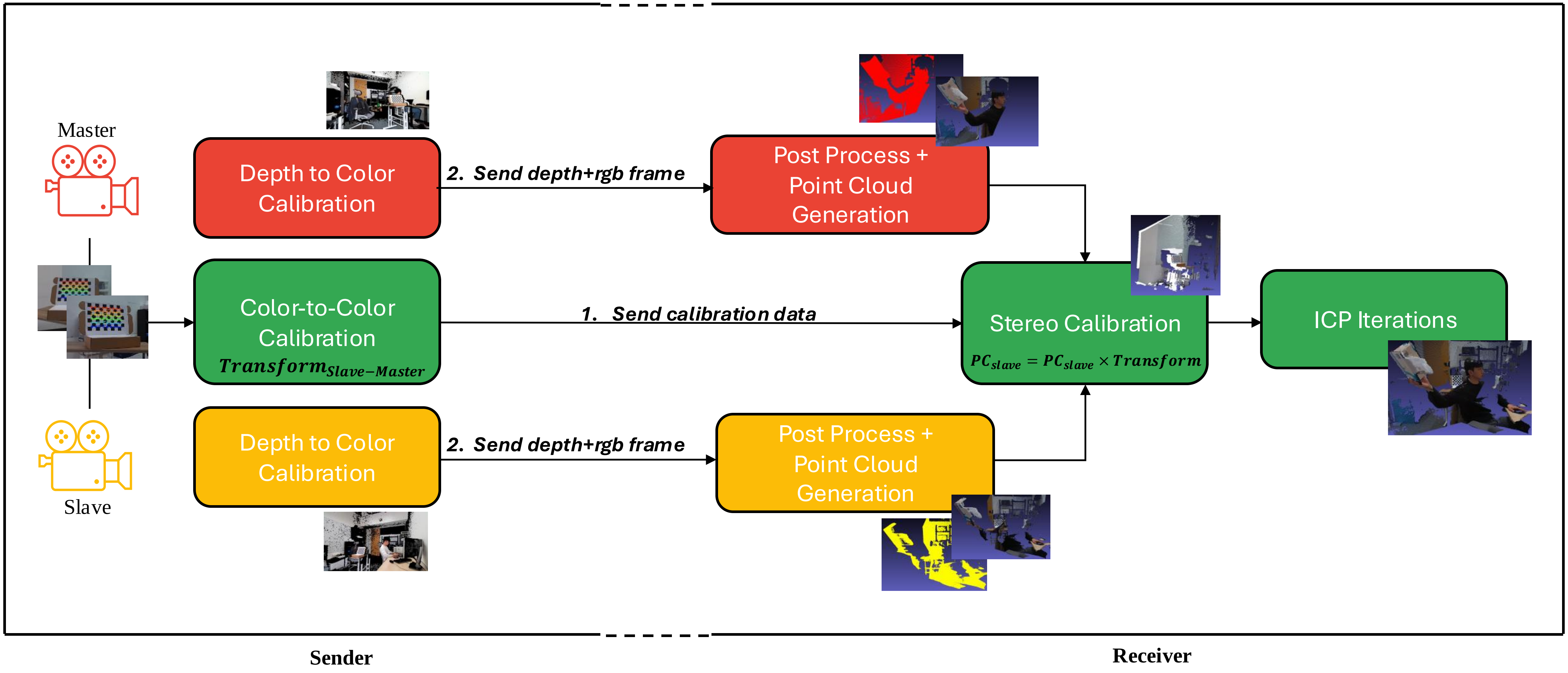
- Summary of Attempt 2:
  1. Retrieve device to device color-to-color calibration using the checkerboard calibration scheme
  2. Create **slave and master point clouds** from respective intrinsic and extrinsic parameters

     ➔ Use ***pinhole*** function in Open3D
  3. Apply stereo transformation matrix to slave point cloud
  4. Refine results with ICP
- **Problem 1**: ICP does not converge even with stereo calibration
- **Problem 2:** Distortion present in point cloud created from pinhole

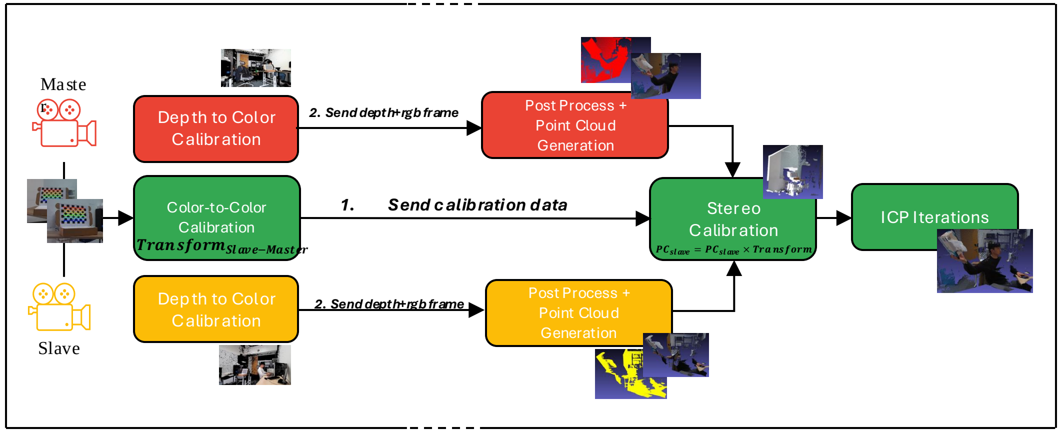
Final ICP result
with same convergence criteria

System Design

Goal 1: Make independent point clouds for master and slave
Goal 2: Avoid using pinhole that adds distortion



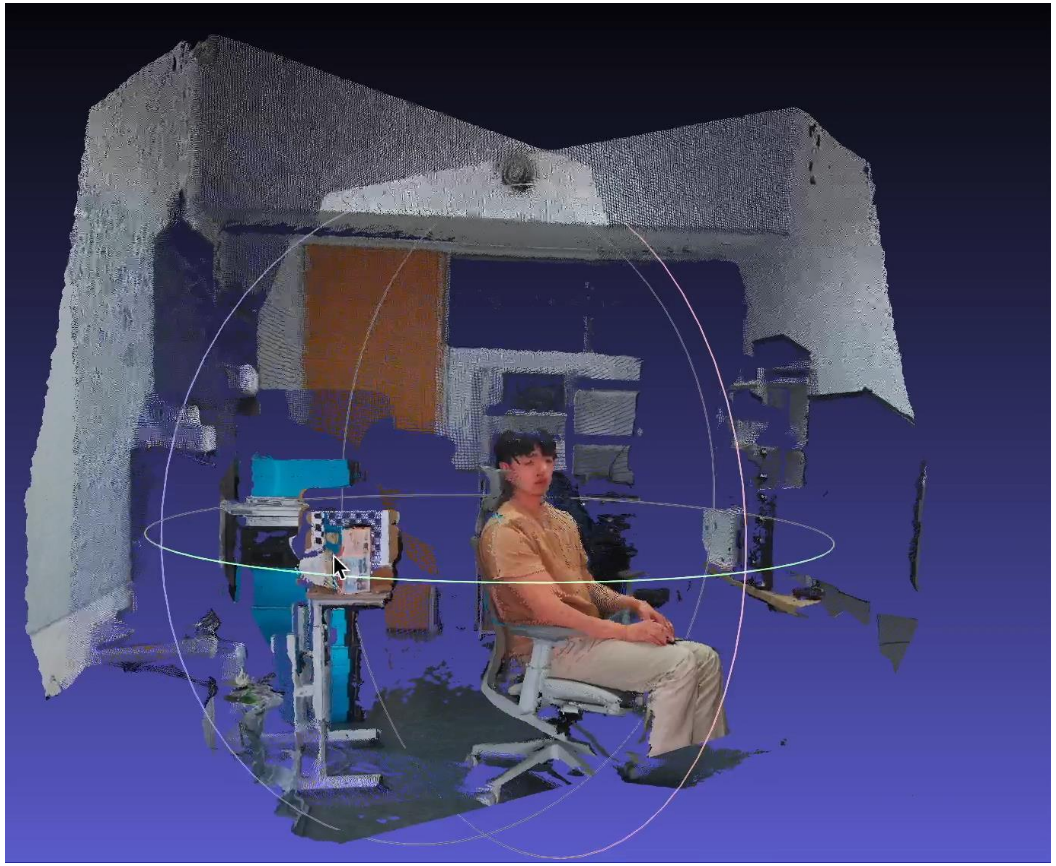
System Design



- Sender:
  1. Calibrate cameras using checkerboard
     - Notify receiver with calibration metrics (socket programming)
  2. Convert depth image to color perspective (main and slave)
     - Send {Main Depth, Main Color}, {Slave Depth, Slave Color} frames
     - 4 main streams - encode using BlenDR
- Receiver:
  1. Receive frames from RTMP server (4 frames)
  2. **Reconstruct point cloud using proposed method**
     - **Stereo Calibration + ICP handled by receiver**
     - **Stereo + ICP latency cost incurred only once (initialization delay)**

Yet to Deploy on BlenDR…

Design Results (Full Result)



Colorized Point Cloud Reconstruction
(Stereo Calib. + ICP)

Future Goal and Plan

Date	Task
APR23 - APR30	• Continue finalizing RTMP pipeline (Receiver end) • Create a full end-to-end demo

MAY01 – …	• Deploy on WebRTC (for bitrate control)
	• Optimize point cloud reconstruction for real-time purposes (30 FPS)
	    • CUDA Implementation of ICP Registration
	    • CUDA Implementation for colorizing Point Cloud

Areas of Improvement	• Fine-tune ICP Registration parameters to minimize discontuinities of planes and edges • Try looking into NeRF (Neural Enhanced Radiance Field) trained with RGBD data for 360° point cloud generation/filling

Thank you.

Appendix

Algorithm 1 Colored point cloud alignment

Input: Colored point cloud $\mathbf{P}$ and $\mathbf{Q}$, initial transformation $\mathbf{T}^0$
Output: Transformation $\mathbf{T}$ that aligns $\mathbf{Q}$ to $\mathbf{P}$

```
Build point cloud pyramids {P^l} and {Q^l}
for p ∈ P^l do
    Precompute d_p by minimizing (10)
    This defines function C_p
T ← T^0, L ← max_pyramid_level
for l ∈ {L, L − 1, ··· , 0} do                ▷ From coarsest to finest
    while not converged do
        r ← 0, J_r ← 0
        Compute the correspondence set K
        for (p, q) ∈ K do
            Compute r_C^(p,q), r_G^(p,q) at T (Eq. 18,19)
            Compute ∇r_C^(p,q), ∇r_G^(p,q) at T (Eq. 29,30)
            Update r and J_r accordingly
        Solve linear system 21 to get ξ
        Update T using Equation 20, then map to SE(3)
Validate if T aligns Q to P
```

Reference

[1] https://scholarworks.calstate.edu/downloads/qr46r322x?locale=it

[2] https://learn.microsoft.com/en-us/azure/kinect-dk/coordinate-systems

[3] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9269787/

[4] https://www.mdpi.com/1424-8220/22/22/8900

[5] https://dl.acm.org/doi/abs/10.1145/3570361.3592525

[6] https://ieeexplore.ieee.org/document/7335499

[7] https://www.open3d.org/docs/release/tutorial/pipelines/colored_pointcloud_registration.html

[8] https://openaccess.thecvf.com/content_ICCV_2017/papers/Park_Colored_Point_Cloud_ICCV_2017_paper.pdf