

BlendPCR

Seamless and Efficient Rendering of Dynamic Point Clouds captured by Multiple RGB-D Cameras



Andre Mühlenbrock*, Rene Weller and Gabriel Zachmann

Computer Graphics and Virtual Reality Research Lab (CGVR), University of Bremen

ICAT-EGVE 2024, 1-3 December 2024, Tsukuba, Japan



Motivation



- Rendering dynamic point cloud is crucial for many VR and XR applications, e.g.:
 - Telepresence
 - Point cloud avatars
 - XR telemedicine
 - Performance capture and live performances
 - In general: Visualization of dynamic RGB-D data



Yu et al. (2021)



Gasques et al. (2021)



Motivation



- Multiple RGB-D cameras are often used to capture a scene more completely



Camera 1



Camera 2





Problem



- However, rendering point clouds by multiple RGB-D cameras currently leads to visible artifacts which we call **seam-flickering**



Uniform Splatting

Separate Meshes



Problem



- Seam-flickering also occurs in leading-edge rendering techniques
 - Due to (a) different specular reflections, (b) white balance and (c) slightly different color gamuts of each camera



Splats



Separate Meshes



Pointersect (2023)
Apple Machine Learning Research



P2ENet (2024)
based on Gaussian Splatting



Our Method



- **BlendPCR**

- A two-step approach to render dynamic point clouds
- **Step 1:** Create separate surfaces for each camera and render them to individual framebuffers
- **Step 2:** Selectively blend these individual framebuffers

➤ This avoids seams and seam-flickering



Step 1: Render Separate Continuous Surfaces



Surface 1

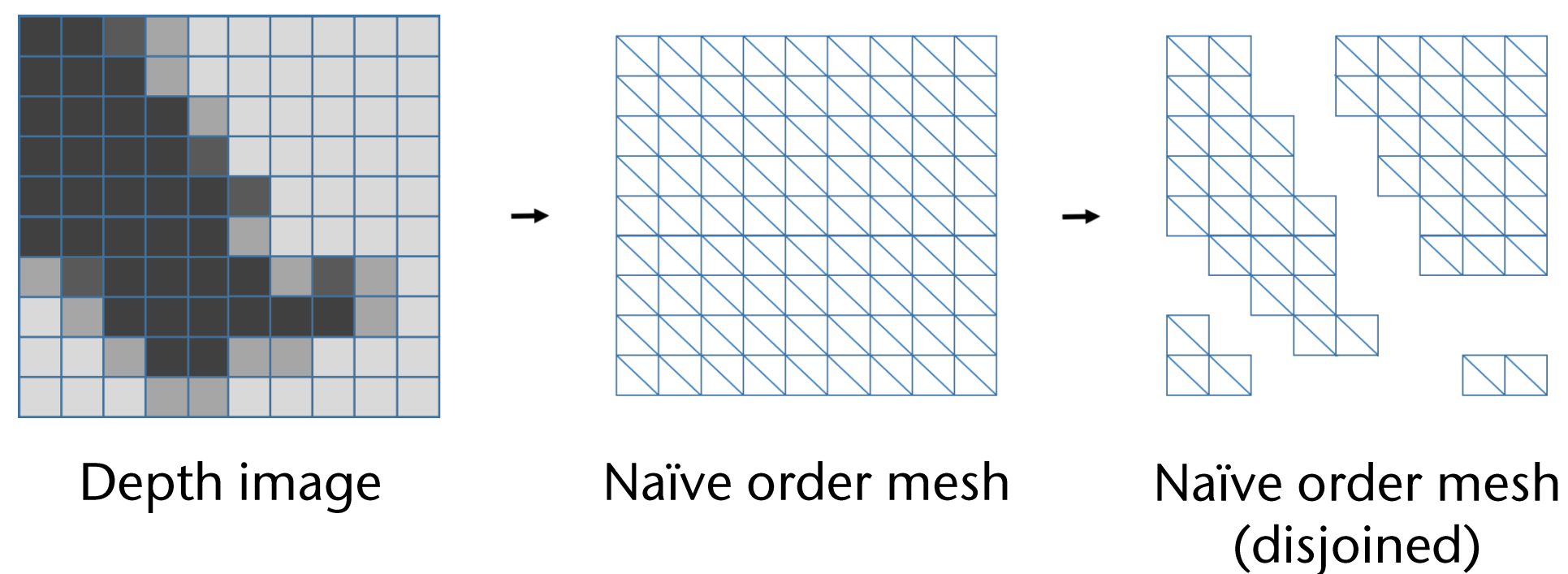


Surface 2



Step 1: Create Separate Continuous Surfaces

- a) Create a mesh in naïve order of the depth image
- High precision & very performant





Step 1: Create Separate Continuous Surfaces

- a) Create a mesh in naïve order of the depth image
 - High precision & very performant
- b) Smooth edges via Moving Least Squares filter
 - To reduce sharp edges and noise



Step 1: Create Separate Continuous Surfaces

- a) Create a mesh in naïve order of the depth image
 - High precision & very performant
- b) Smooth edges via Moving Least Squares filter
 - To reduce sharp edges and noise
- c) Estimate surface normal
 - Using Cholesky decomposition [1]

[1] J. Klein, G. Zachmann, "Proximity Graphs for Defining Surfaces over Point Clouds," SPBG 2004.

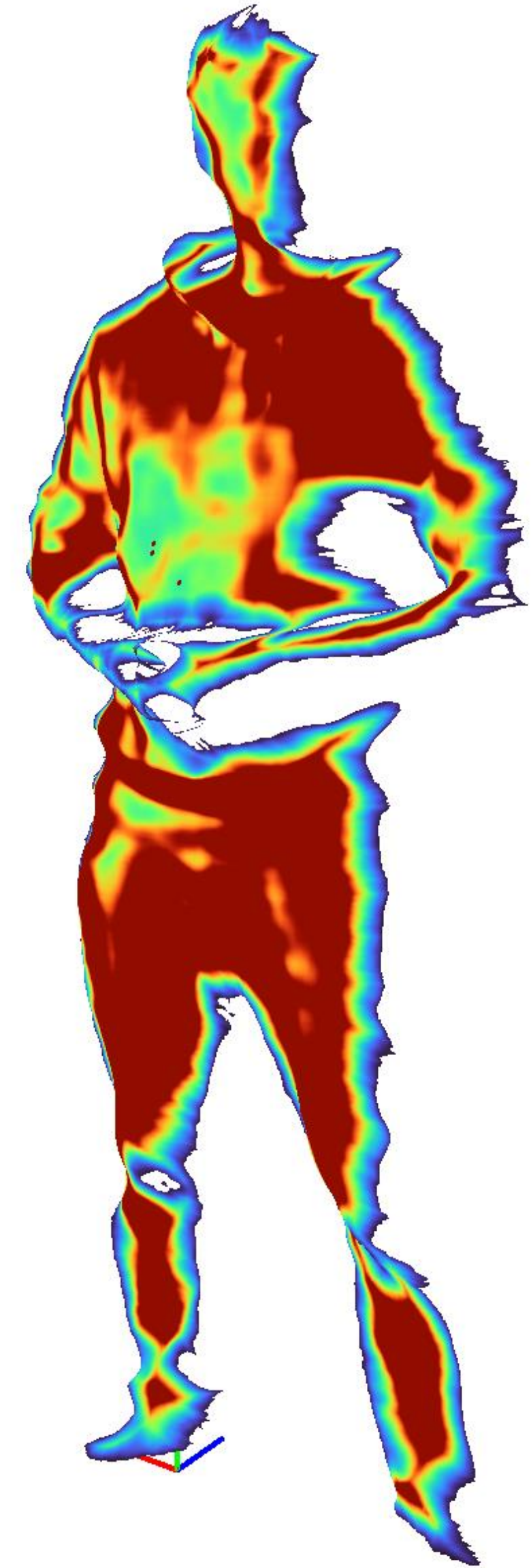




Step 1: Create Separate Continuous Surfaces

d) Estimate Accuracy for each vertex, based on:

- Distance to camera
- Surface normal
- Proximity to the edge





Step 1: Create Separate Continuous Surfaces

d) Estimate Accuracy for each vertex, based on:

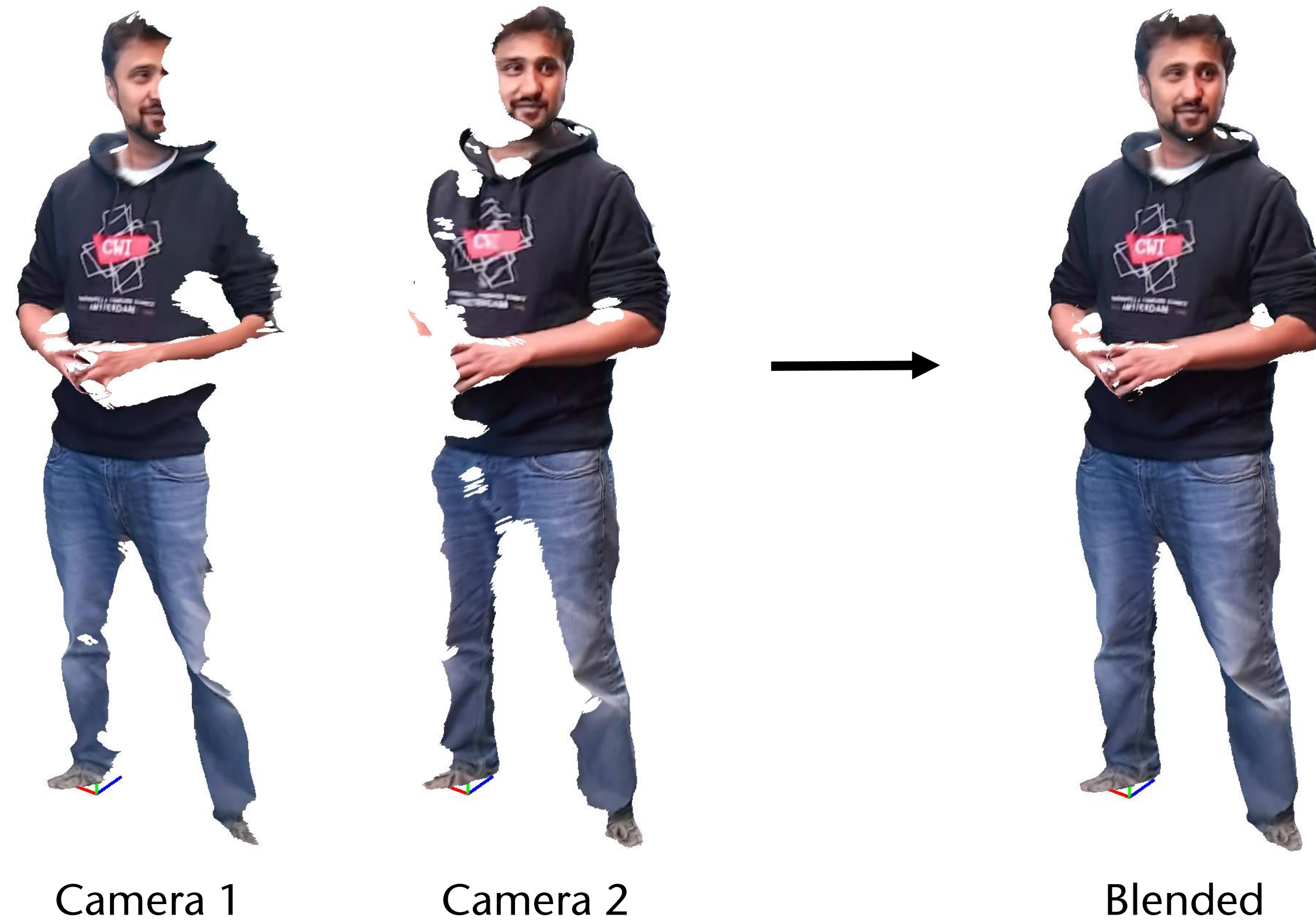
- Distance to camera
- Surface normal
- Proximity to the edge

e) Render to individual framebuffer





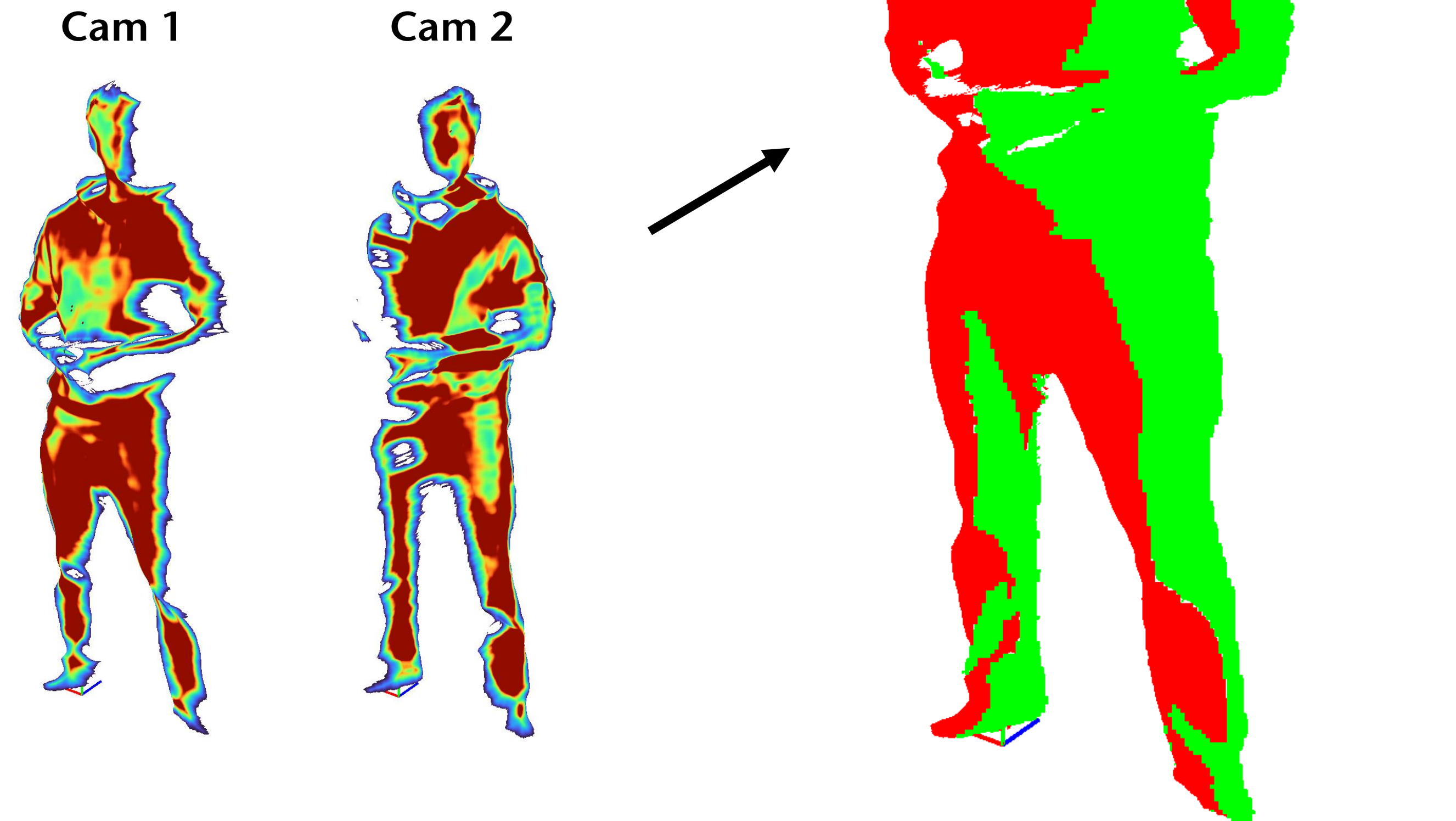
Step 2: Selectively blend Separate Surfaces





Step 2: Selectively blend separate surfaces

- a) Choose the **major camera** based on estimated accuracy



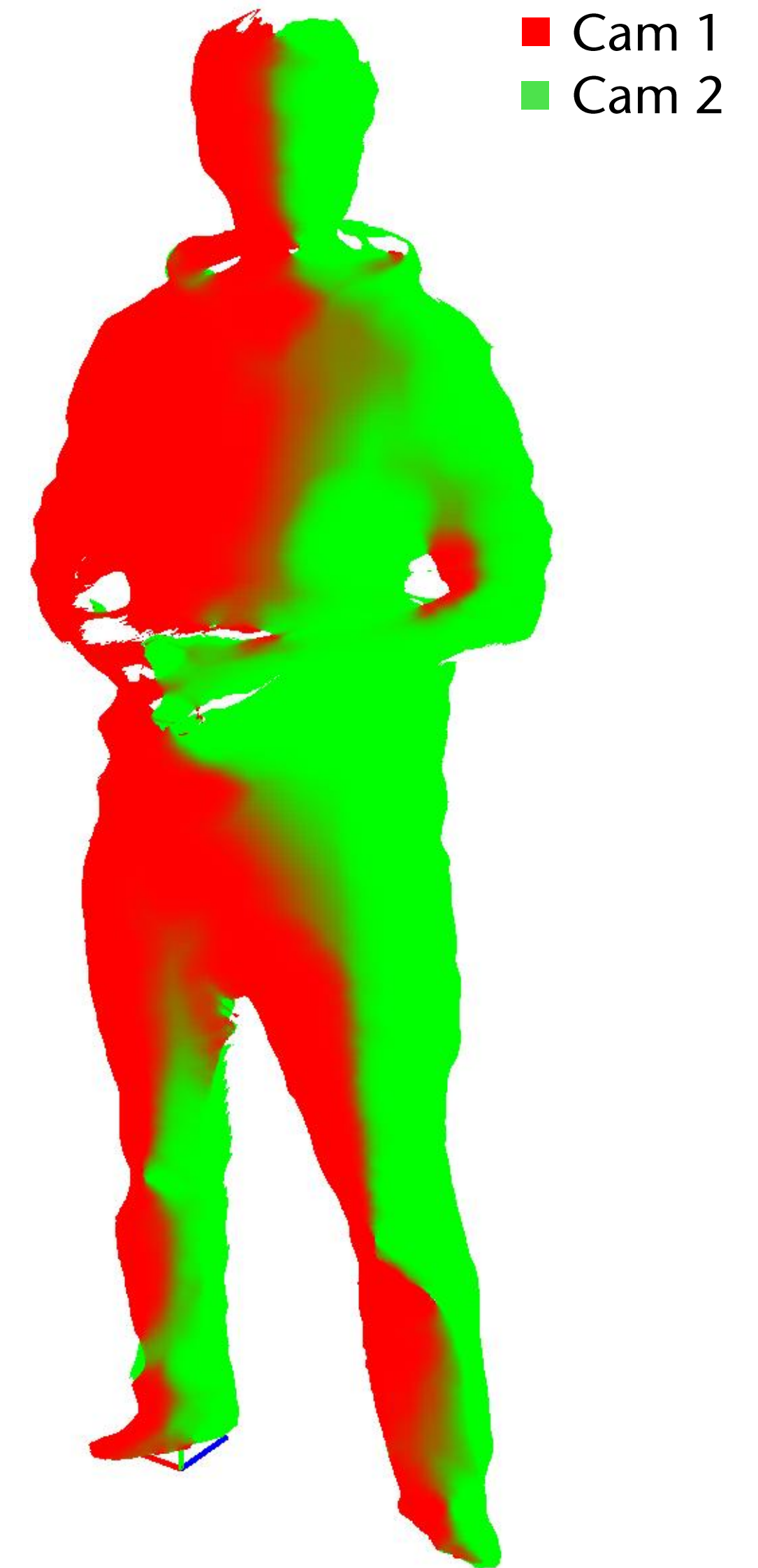


Method



Step 2: Selectively blend separate surfaces

- a) Choose the **major camera** based on estimated accuracy
- b) Calculate **camera weights** in screen space
 - For seamless transitions





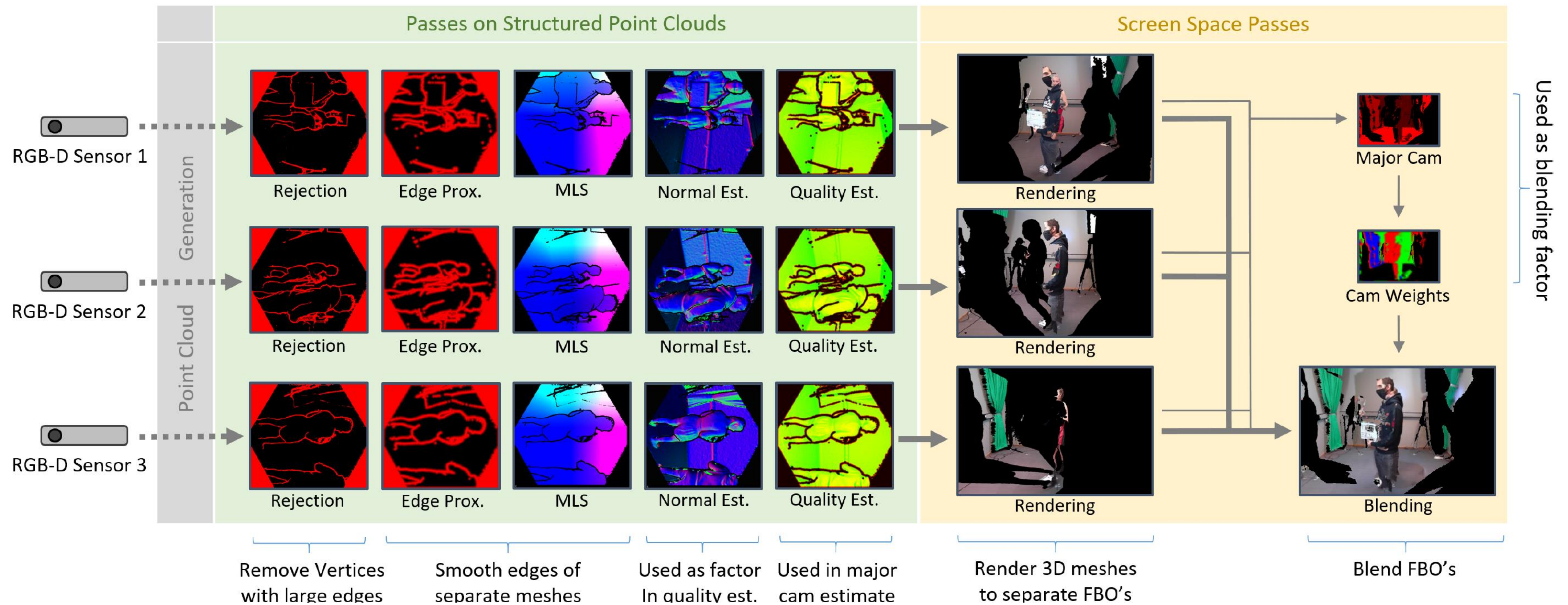
Step 2: Selectively blend separate surfaces

- a) Choose the **major camera** based on estimated accuracy
- b) Calculate **camera weights** in screen space
 - For seamless transitions
- c) Blend separate meshes based on (b)
 - Prevents seam-flickering, only draws the information with the highest accuracy, and no blurring





Overview of our Pipeline





High Resolution (HR) Encoding



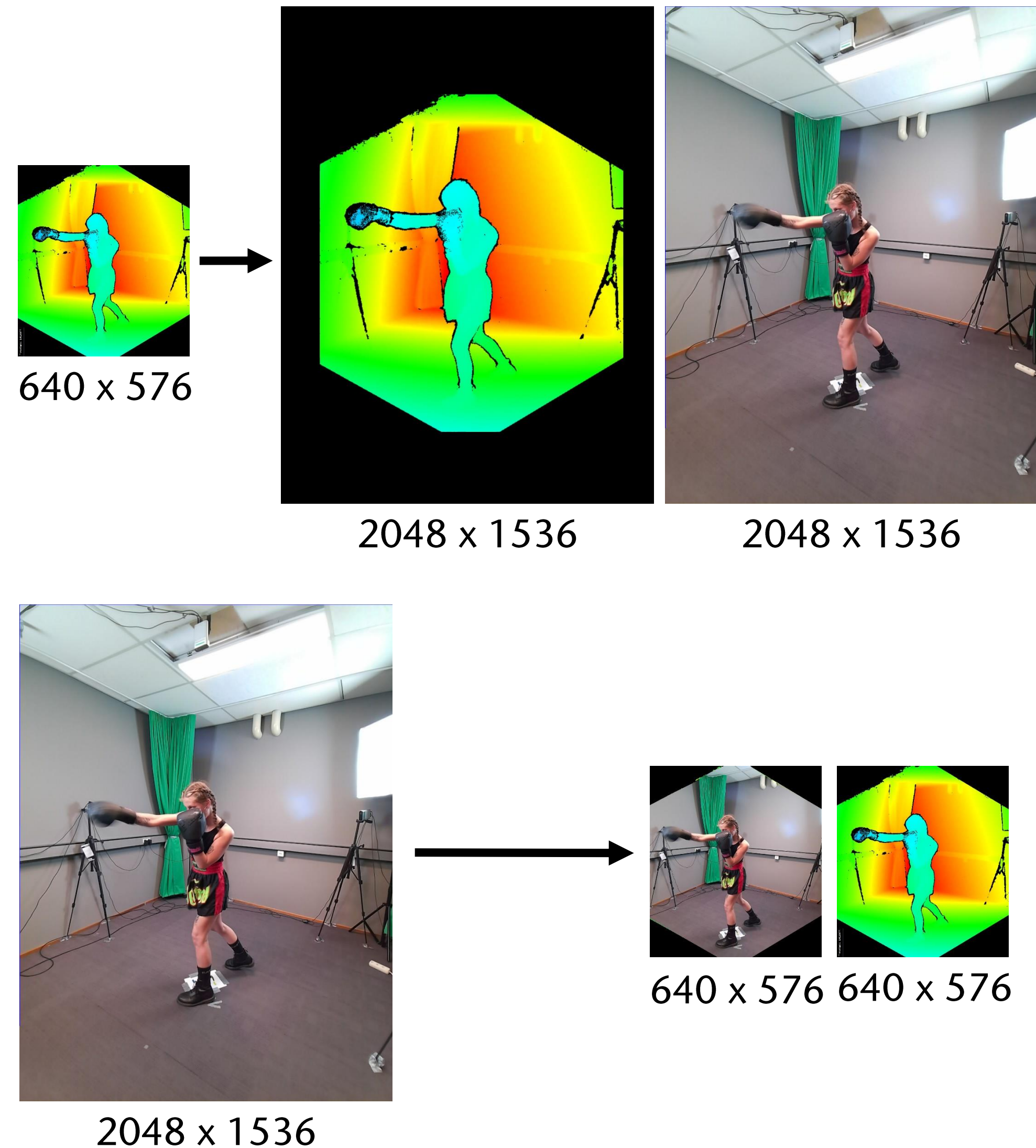
Further improving quality



High Resolution (HR) Encoding



- When working with RGB-D cameras
 - Depth image can be mapped onto the color image
 - Point Cloud gets huge; bad for performance
 - Color image can be mapped onto the depth image
 - Color resolution is (significantly) reduced





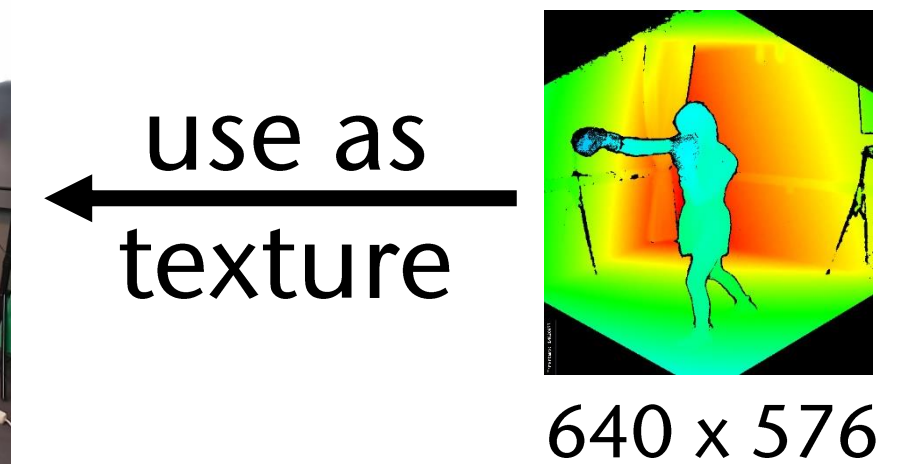
High Resolution (HR) Encoding



- Texture mapping
 - Using high resolution color image as texture for low resolution point cloud
 - Is not a problem, since we work with meshes
- But how to get the UV coordinates?



2048 x 1536

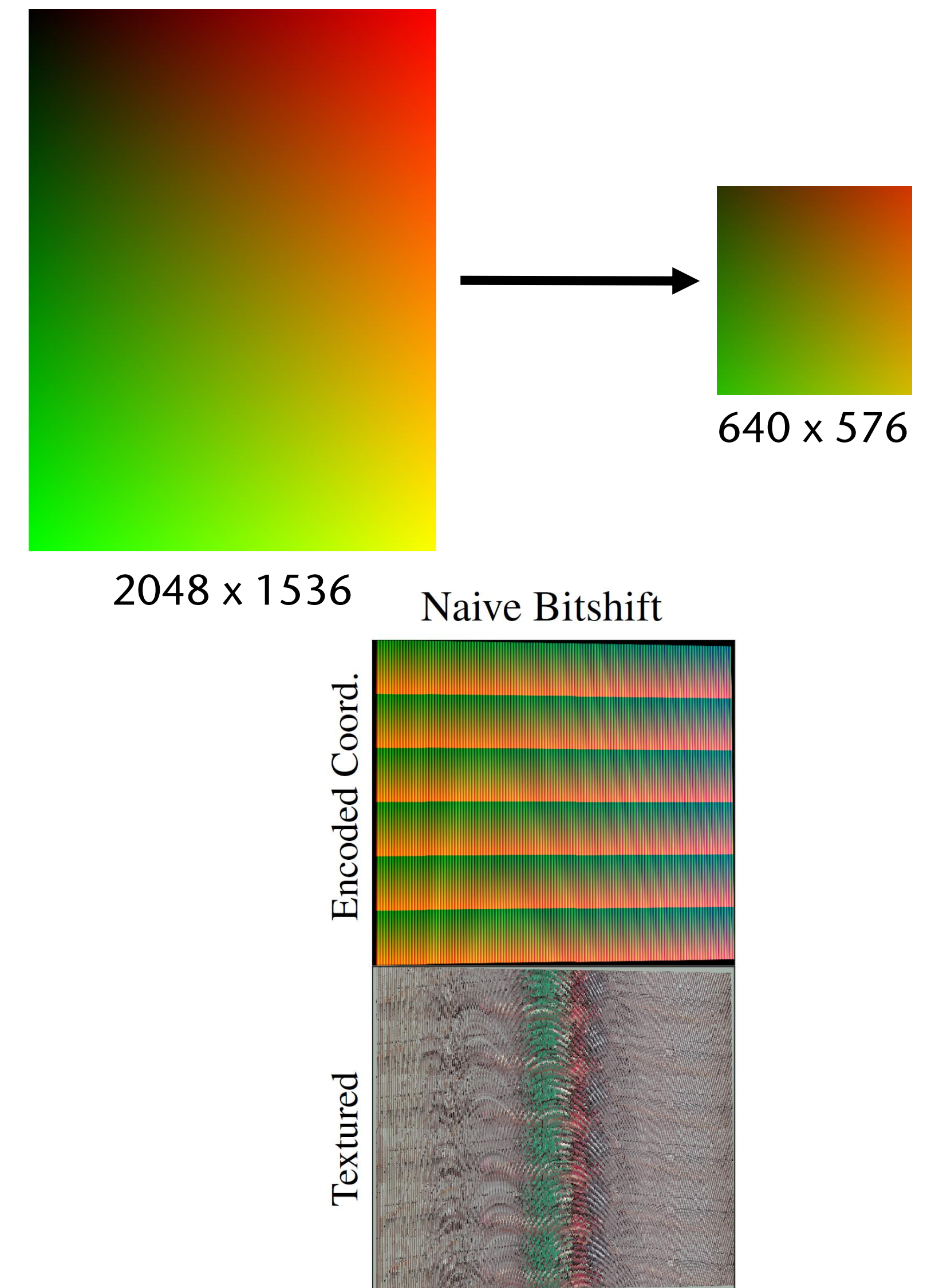




High Resolution (HR) Encoding



- Obtain UV coordinates:
 - We can transform an image of UV coordinates onto the depth image
 - However, the Azure Kinect SDK only allows to transform RGBA32 images onto depth
 - We can encode **2x 16 bit UV** into **4 x 8 bit RGBA**
 - However, due to linear interpolation, this information is destroyed

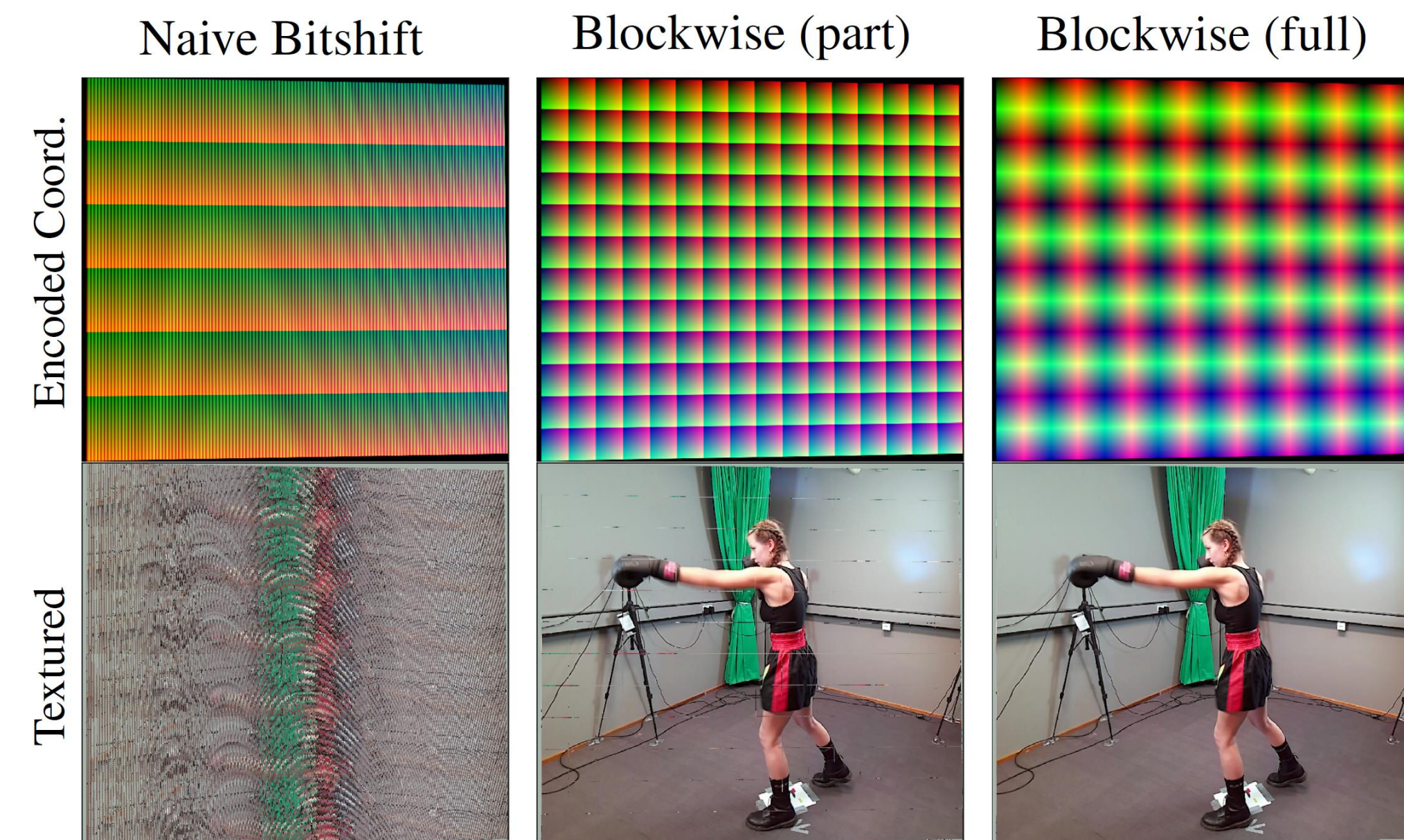




Interpolation-Resistant Encoding Scheme



- We created a novel interpolation-resistant encoding scheme
 - To encode **2 x 16 bit UV** coordinates into **4 x 8 bit RGBA** values
 - This finally allows us to generate UV coordinates using arbitrary SDK's color-to-depth transformation functionality.
 - Details in the paper





High Resolution (HR) Encoding



Low Resolution Color, 640 x 576



High Resolution Color, 2048 x 1536



Results

- Tested on CWIPC-SXR dataset [2]
 - Used seven Microsoft Azure Kinect
 - 45 social XR scenarios



[2] I. Reimat et al., "CWIPC-SXR: Point Cloud Dynamic Human Dataset for Social XR," ACM MMSys 2021.



Results



- CWIPC-SXR, S1 3 Card Trick Scene



Separate Meshes

BlendPCR



Results



- CWIPC-SXR, S7 Scarf Dressing

Uniform Splats



Separate Meshes



TSDf 512³



BlendPCR



Splats



Separate Meshes



TSDf 512



Pointersect (2023)^[3]
Apple Machine Learning Research



P2ENet (2024)^[3]
based on Gaussian Splatting



BlendPCR (2024)
Ours (CGVR)

[3] From: Y Hu et al., "Low Latency Point Cloud Rendering with Learned Splatting," CVPR Workshop 2024.



Results



- CWIPC-SXR, S3 Flight Attendant

Uniform Splats



Separate Meshes



TSDF 512³



BlendPCR (HR)



Splats



Separate Meshes



TSDF 512



Pointersect (2023) [3]
Apple Machine Learning Research



P2ENet (2024) [3]
based on Gaussian Splatting

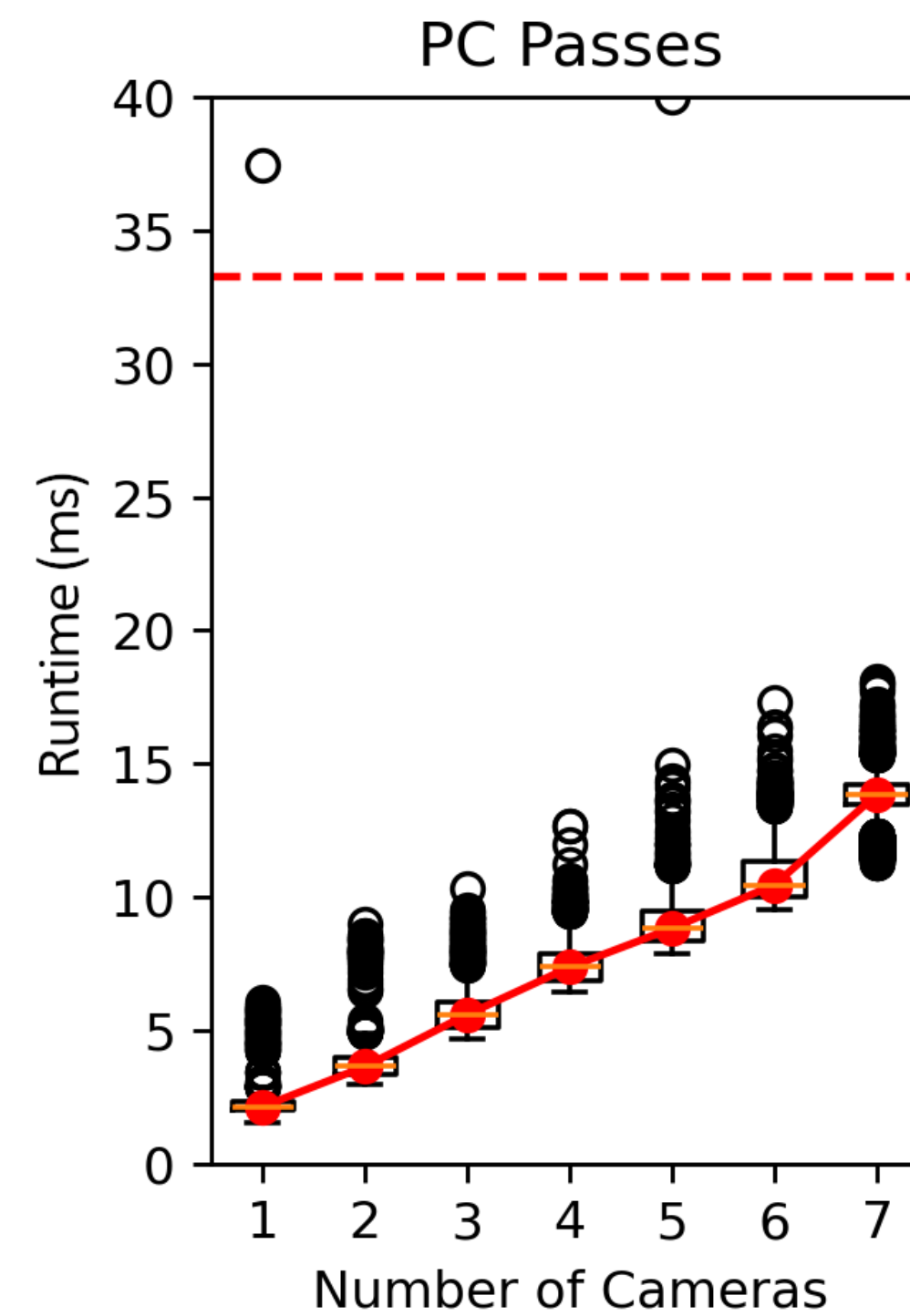


BlendPCR (2024)
Ours (CGVR)

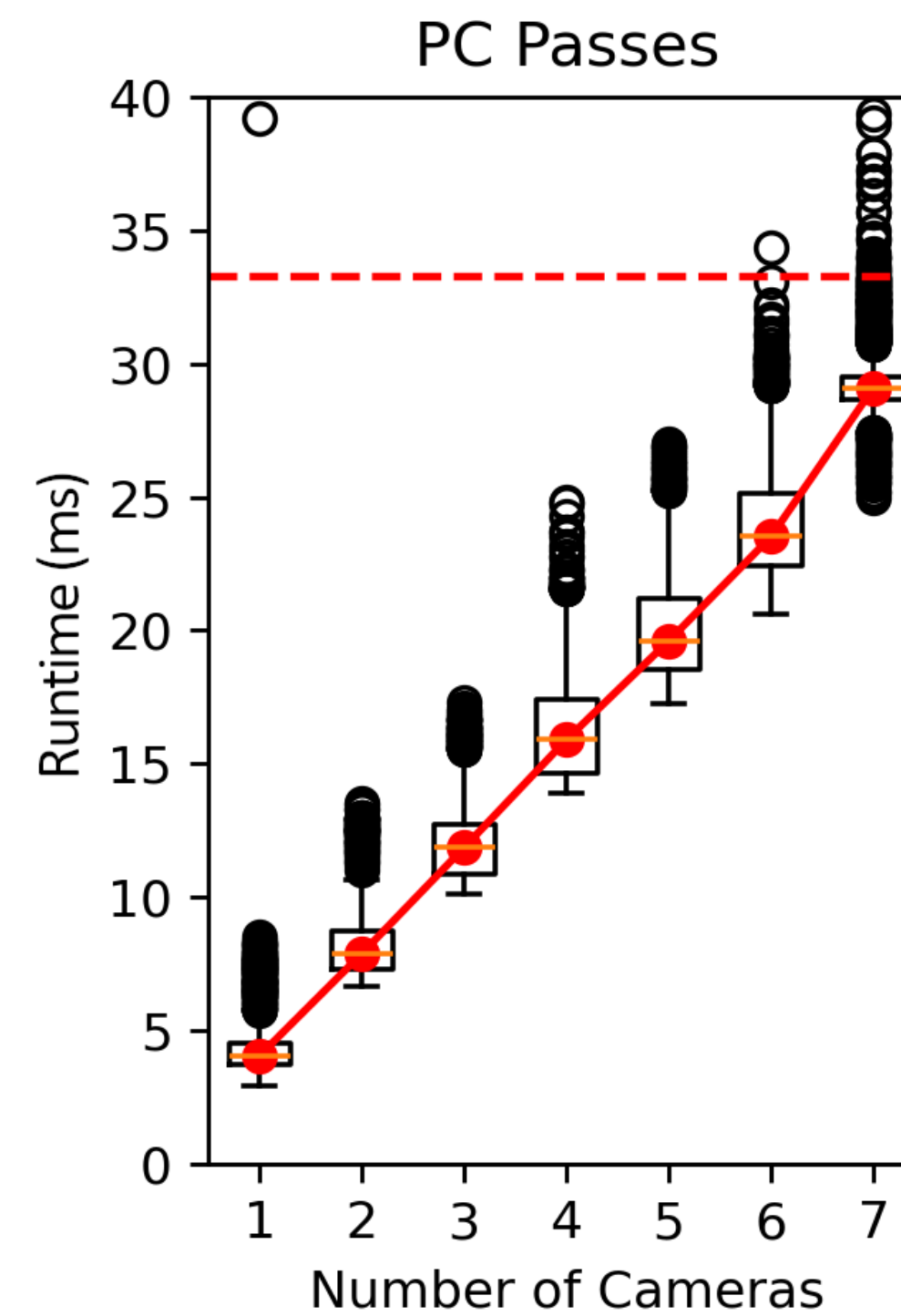
[3] From: Y Hu et al., "Low Latency Point Cloud Rendering with Learned Splatting," CVPR Workshop 2024.



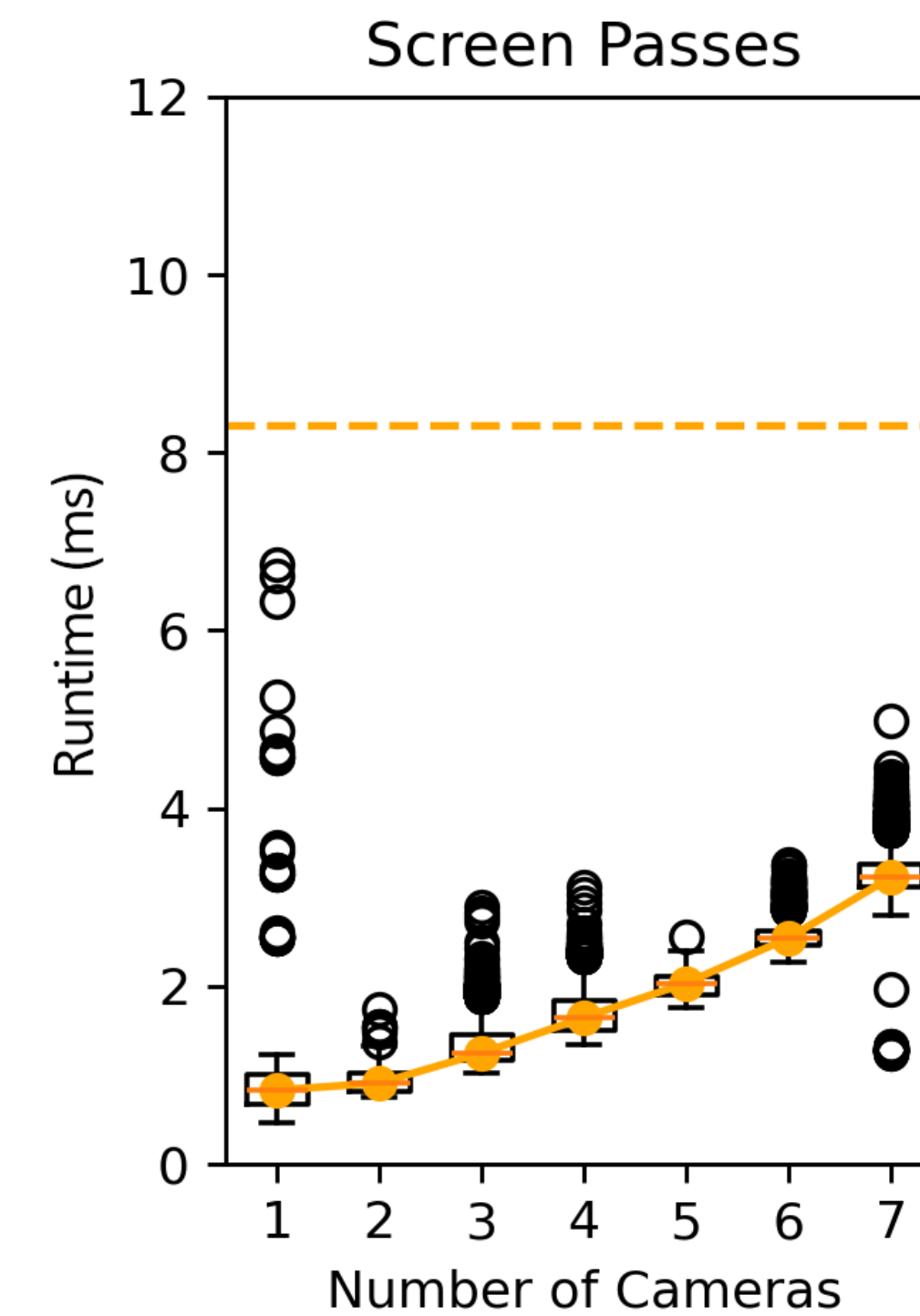
Performance



BlendPCR



BlendPCR (HR)

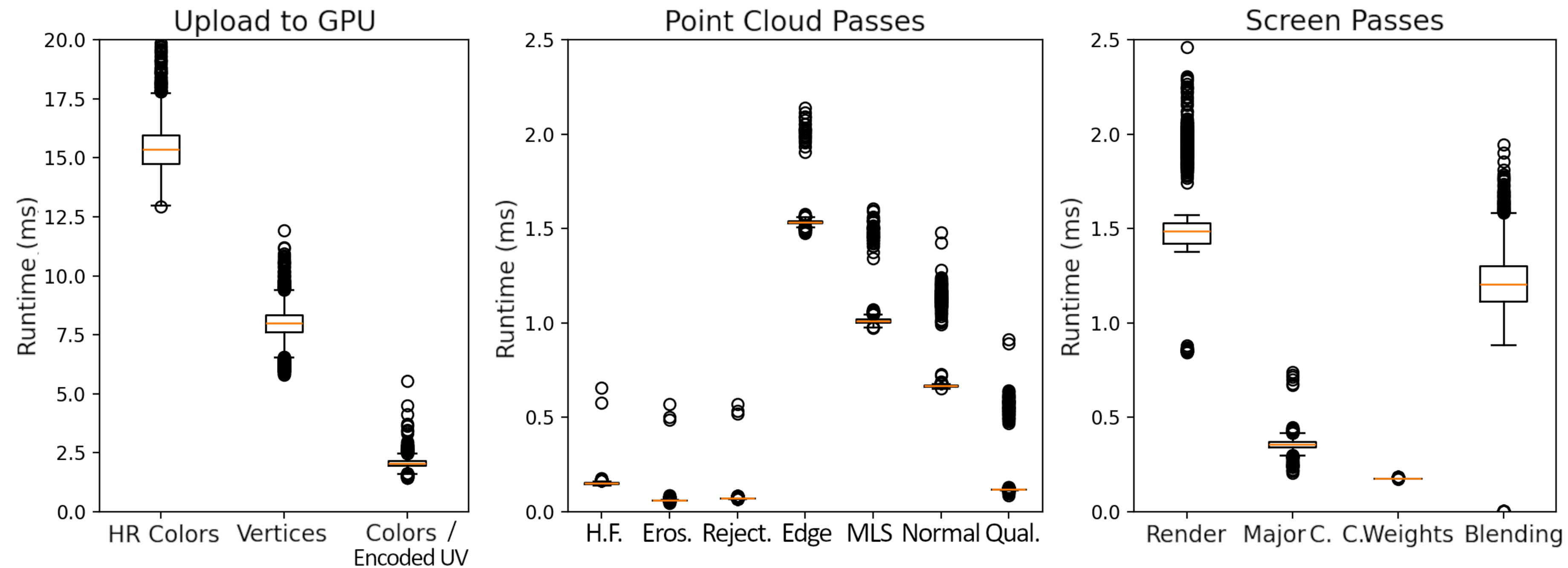


Both

Using an NVIDIA GeForce RTX 4090 @ 3580 x 2066



Performance



Using an NVIDIA GeForce RTX 4090 @ 3580 x 2066

More details
in the paper



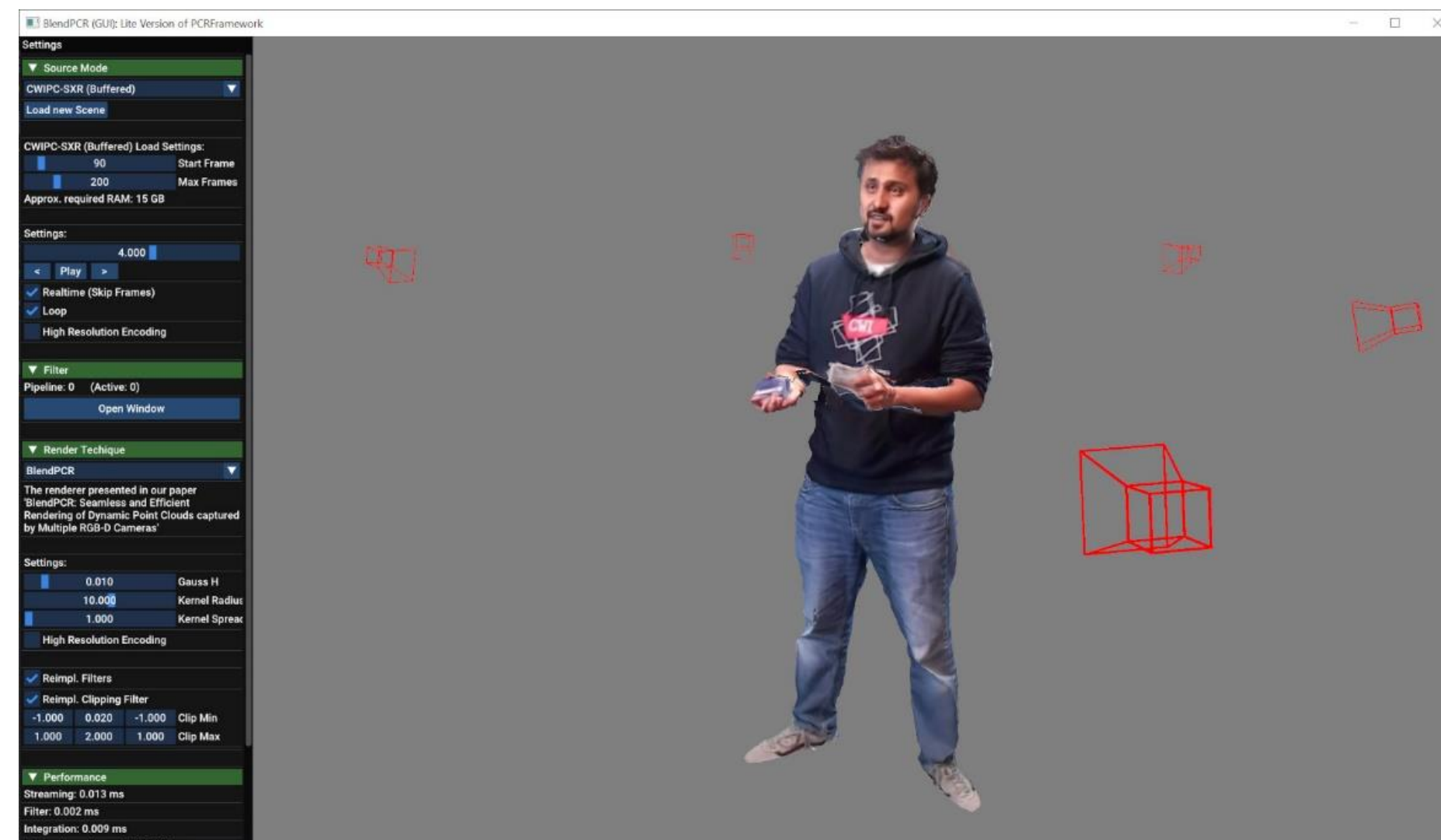
Source Code



- Github: <https://github.com/muehlenb/blendpcr>

➤ Pure OpenGL 3.3 implementation

➤ Pre-built binaries available



BlendPCR

Public

Pin

Unwatch

main

2 Branches

0 Tags

Go to file

Add file

Code

muehlenb

Fixed typo.

0e22bc2 · last week

71 Commits

data/model	Started to transform our PCR framework into a BlendPCR-on...	last month
images	Better screenshot.	3 weeks ago
lib	Improving integration of libraries.	3 weeks ago
shader	GLSL compatibility fix for some graphics cards & Streaming f...	3 weeks ago
src	GLSL compatibility fix for some graphics cards & Streaming f...	3 weeks ago
.gitignore	Added build folder to gitignore.	3 weeks ago
CMakeLists.txt	Updated CMakeLists.txt (k4a) and Readme.md	3 weeks ago
LICENSE	Added EUROPEAN UNION PUBLIC LICENCE v. 1.2 license	2 weeks ago
Readme.md	Fixed typo.	last week

README

EUPL-1.2 license


BlendPCR: Seamless and Efficient Rendering of Dynamic Point Clouds captured by Multiple RGB-D Cameras


Video (available soon) | Paper (available soon)


C++/OpenGL implementation of our real-time renderer BlendPCR for dynamic point clouds derived from multiple RGB-D cameras. It combines efficiency with high-quality rendering while effectively preventing common z-fighting-like seam flickering. The software is equipped to load and stream the CWIPC-SXR dataset for test purposes and comes with a GUI.


[Andre Mühlenbrock¹](#), [Rene Weller¹](#), [Gabriel Zachmann¹](#)
¹Computer Graphics and Virtual Reality Research Lab ([CGVR](#)), University of Bremen


Conditionally Accepted at ICAT-EGVE 2024



Splats
CWIPC-SXR, S7


BlendPCR
CWIPC-SXR, S7


Splats
CWIPC-SXR, S13


BlendPCR
CWIPC-SXR, S13


Splats
CWIPC-SXR, S3


BlendPCR
CWIPC-SXR, S3

Pre-built Binaries

If you only want to test the BlendPCR renderer, without editing the implementation, we also offer pre-built binaries:

- [Download Windows \(64-Bit\)](#), without CUDA for all graphic cards.

Build Requirements

Required:

- CMake ≥ 3.11
- OpenGL ≥ 3.3
- C++ Compiler, e.g. MSVC v143
- Azure Kinect SDK 1.4.1: Required to load and stream the CWIPC-SXR dataset.



Conclusion



- Novel rendering technique for dynamic point clouds of multiple RGB-D camera
 - No seam-flickering or seams
 - Always uses the most accurate data
 - Very performant & applicable for VR
- Encoding scheme for high resolution textures



➤ Objective measurement to state-of-the-art techniques



Pointersect (2023)
Apple Machine Learning Research



P2ENet (2024)
based on Gaussian Splatting



BlendPCR (2024)
Ours (CGVR)

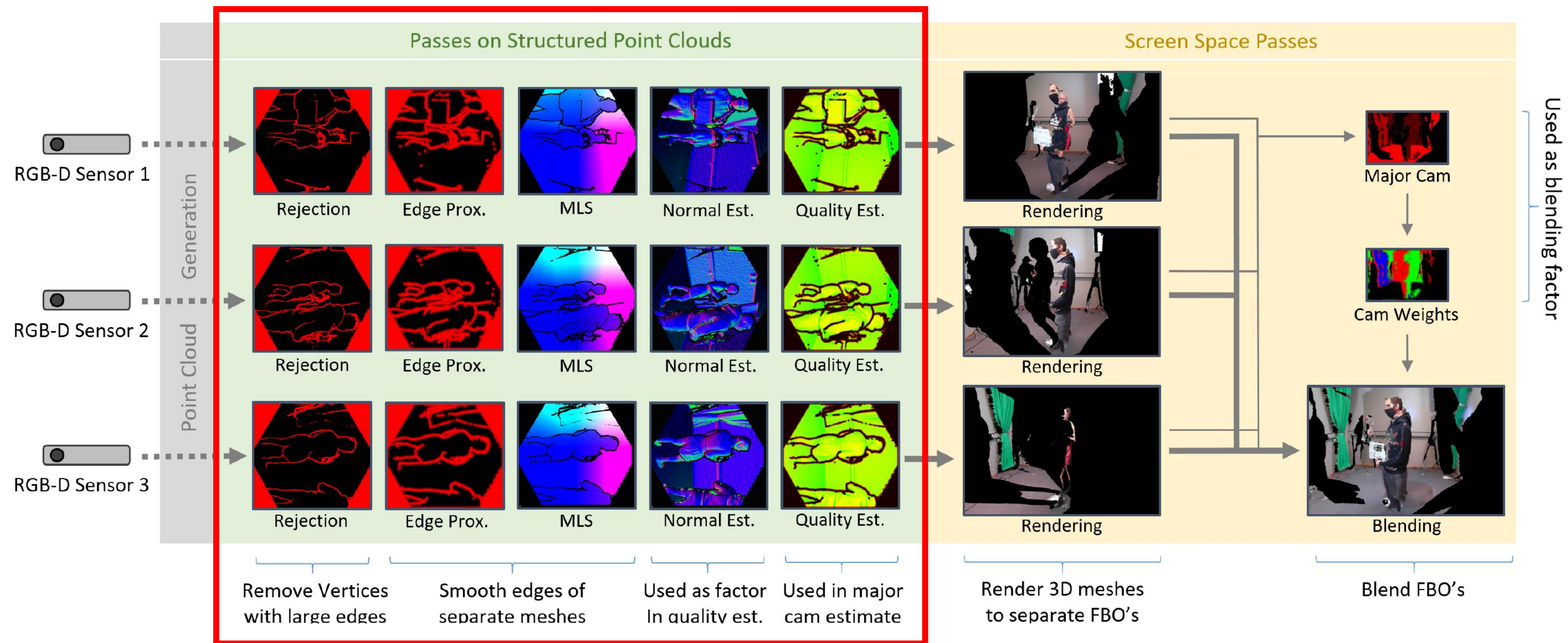
➤ Subjective comparison study in VR environment



Future Work



➤ Finally: Integrate with other State-of-the-Art techniques



Integrate Gaussian Splatting, NeRF's, etc.



Thank you for your attention!



Separate Meshes

BlendPCR

Contact: muehlenb@uni-bremen.de