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

Andre Mühlenbrock*, Rene Weller and Gabriel Zachmann

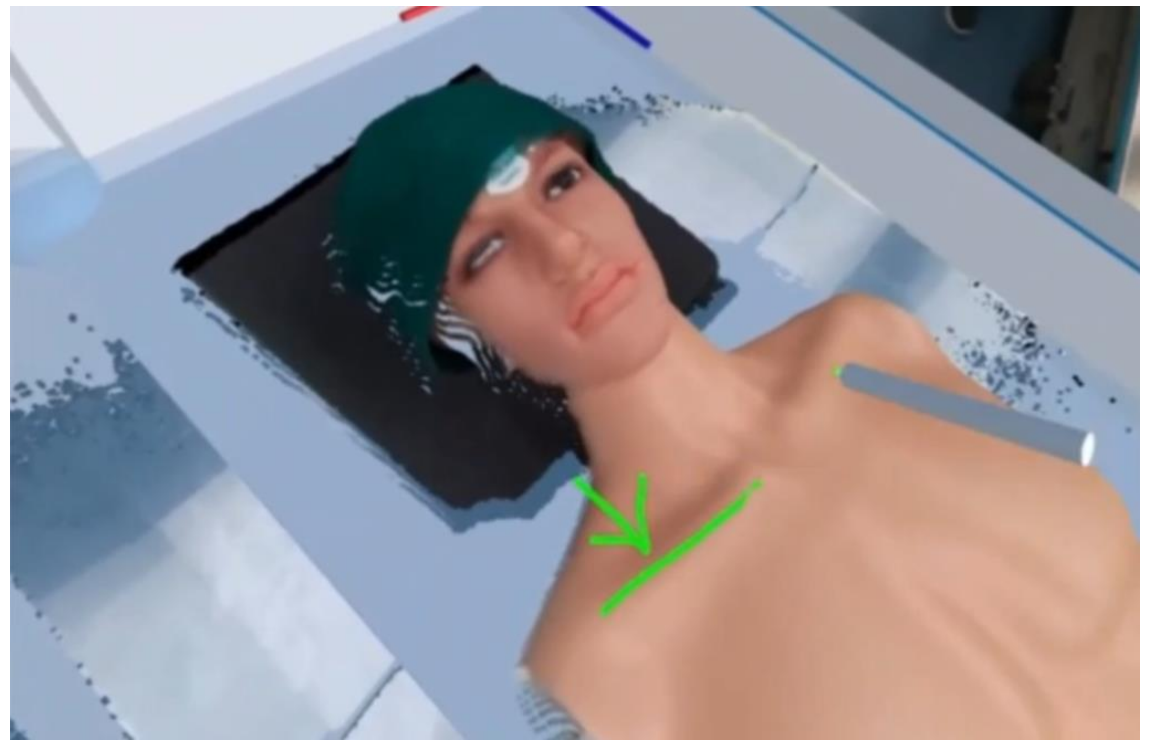
Computer Graphics and Virtual Reality Research Lab (CGVR), University of Bremen

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

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

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

Method

Step 1: Render Separate Continuous Surfaces

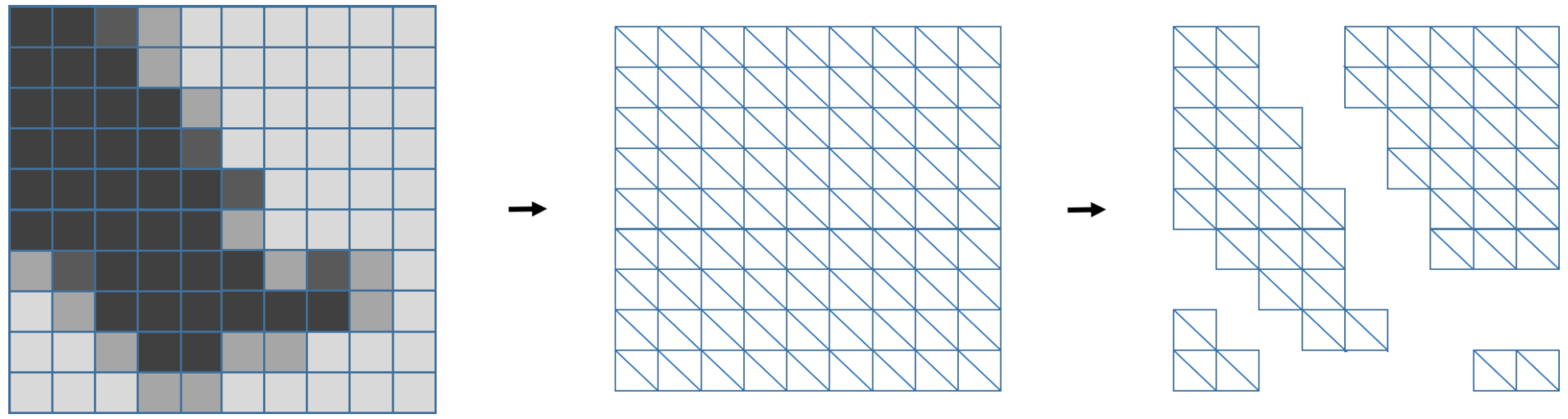
Surface 1

Surface 2

Step 1: Create Separate Continuous Surfaces

a) Create a mesh in naïve order of the depth image

- High precision & very performant



Depth image → Naïve order mesh → Naïve order mesh (disjoined)

Method

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

Method

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]

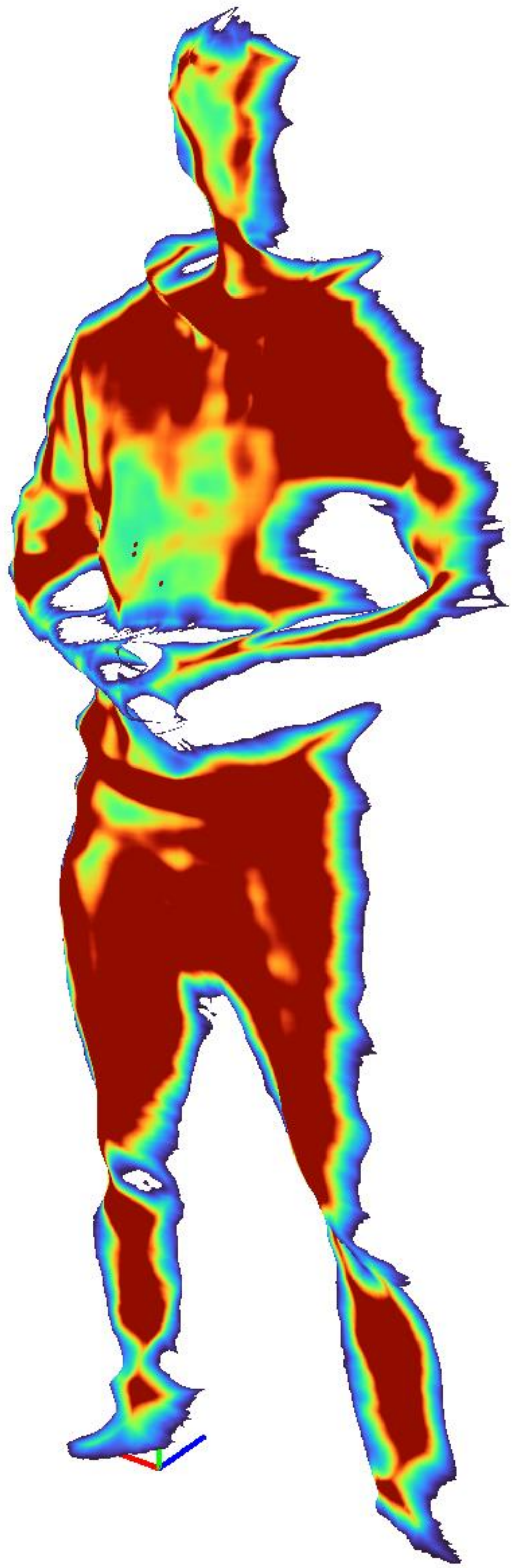
[1] J. Klein, G. Zachmann, "Proximity Graphs for Defining Surfaces over Point Clouds," SPBG 2004.

Method

Step 1: Create Separate Continuous Surfaces

d) Estimate Accuracy for each vertex, based on:

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



Method

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

Method

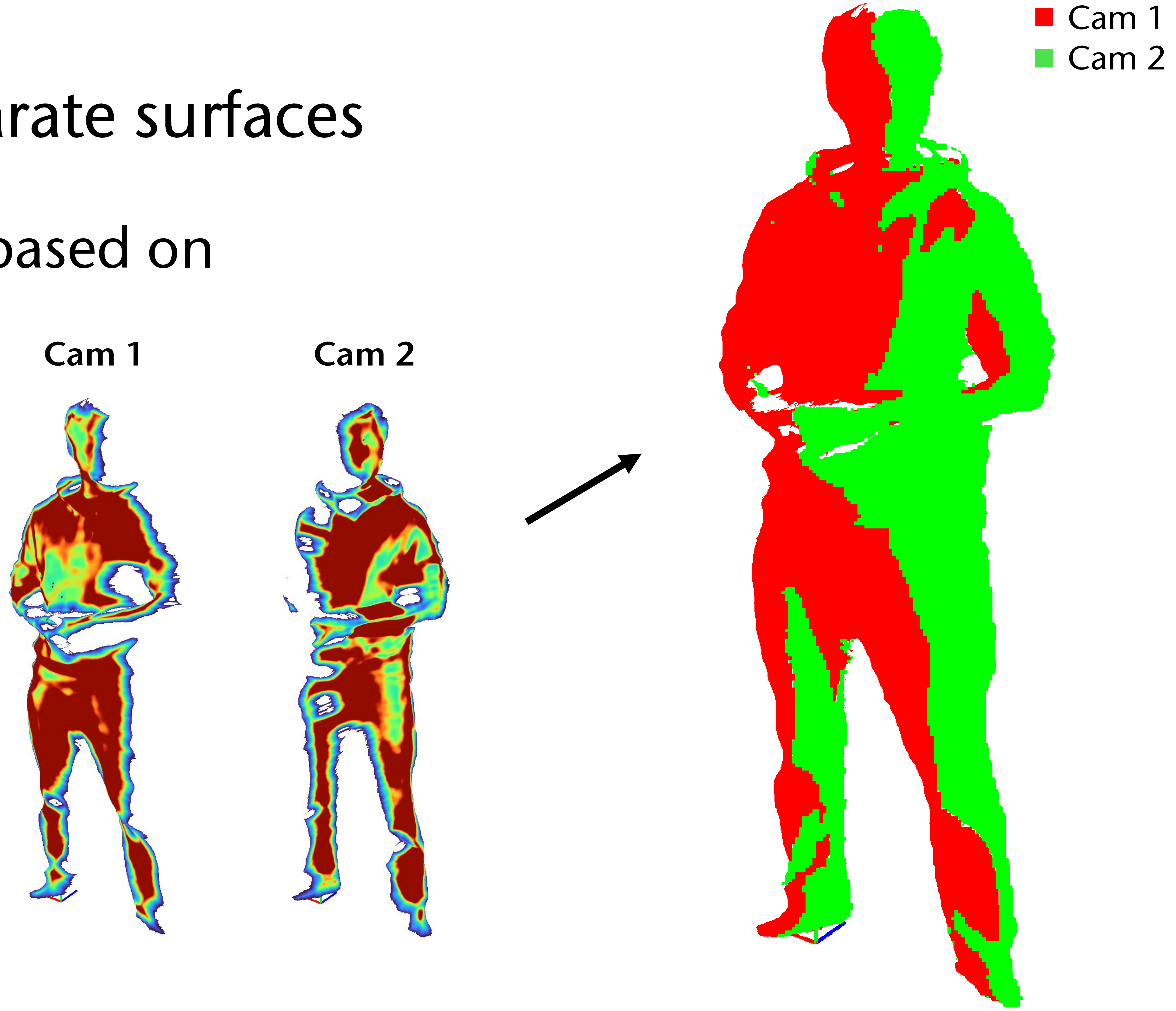
Step 2: Selectively blend Separate Surfaces

Camera 1 Camera 2 Blended

Method

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

Method

Step 2: Selectively blend separate surfaces

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

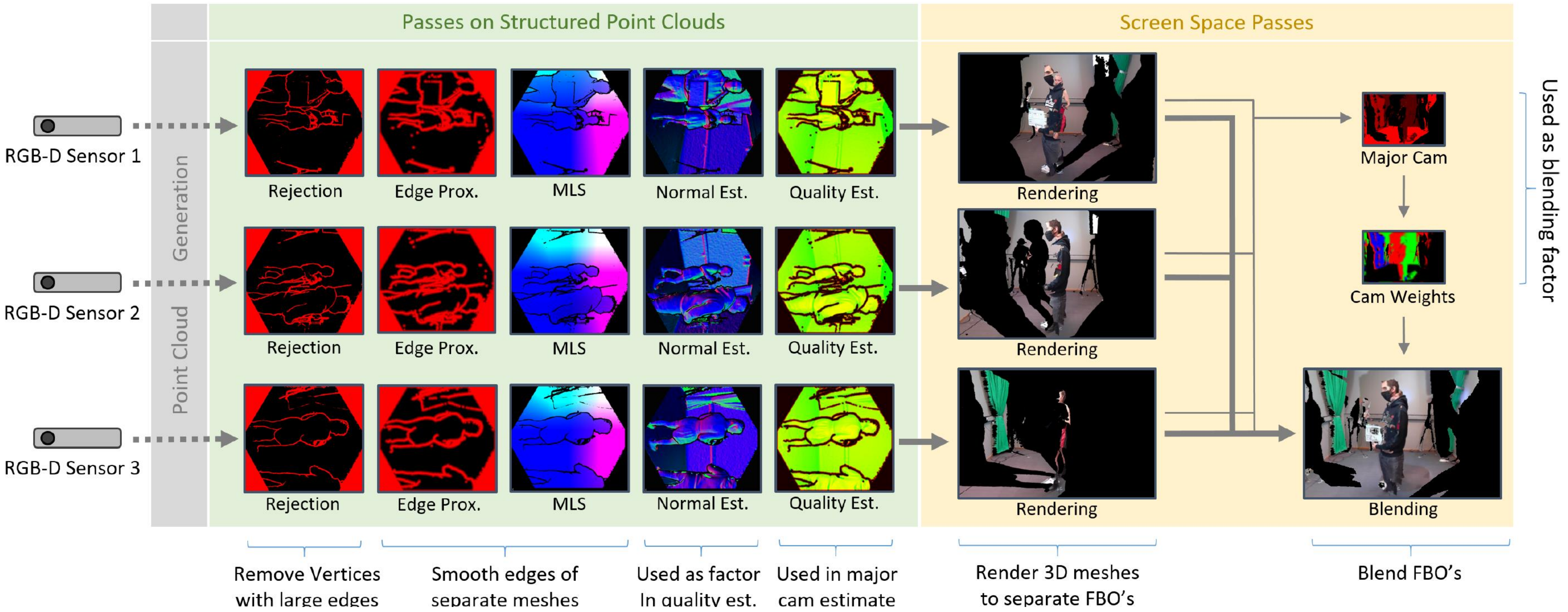
b) Calculate **camera weights** in screen space

- For seamless transitions

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

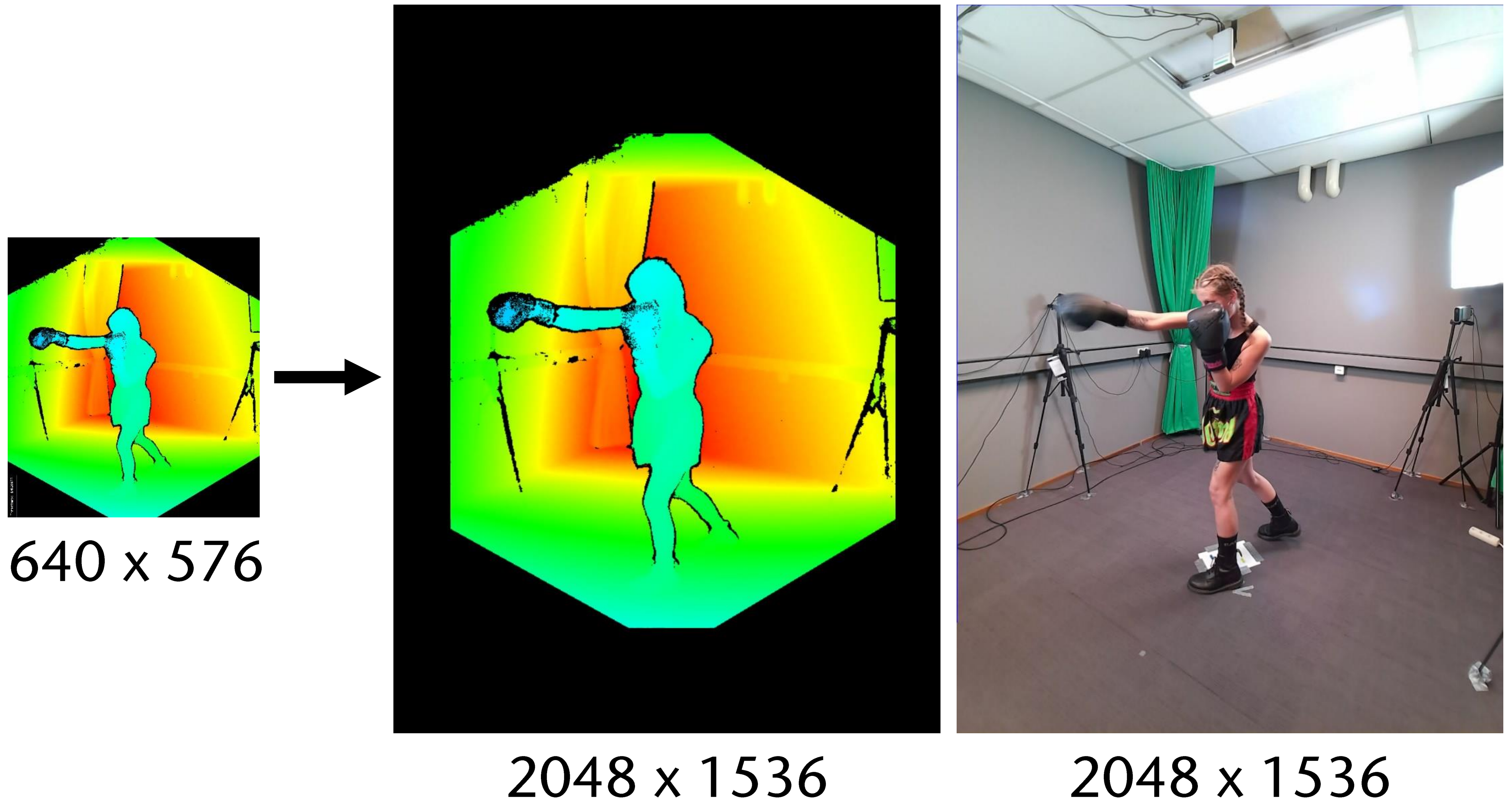


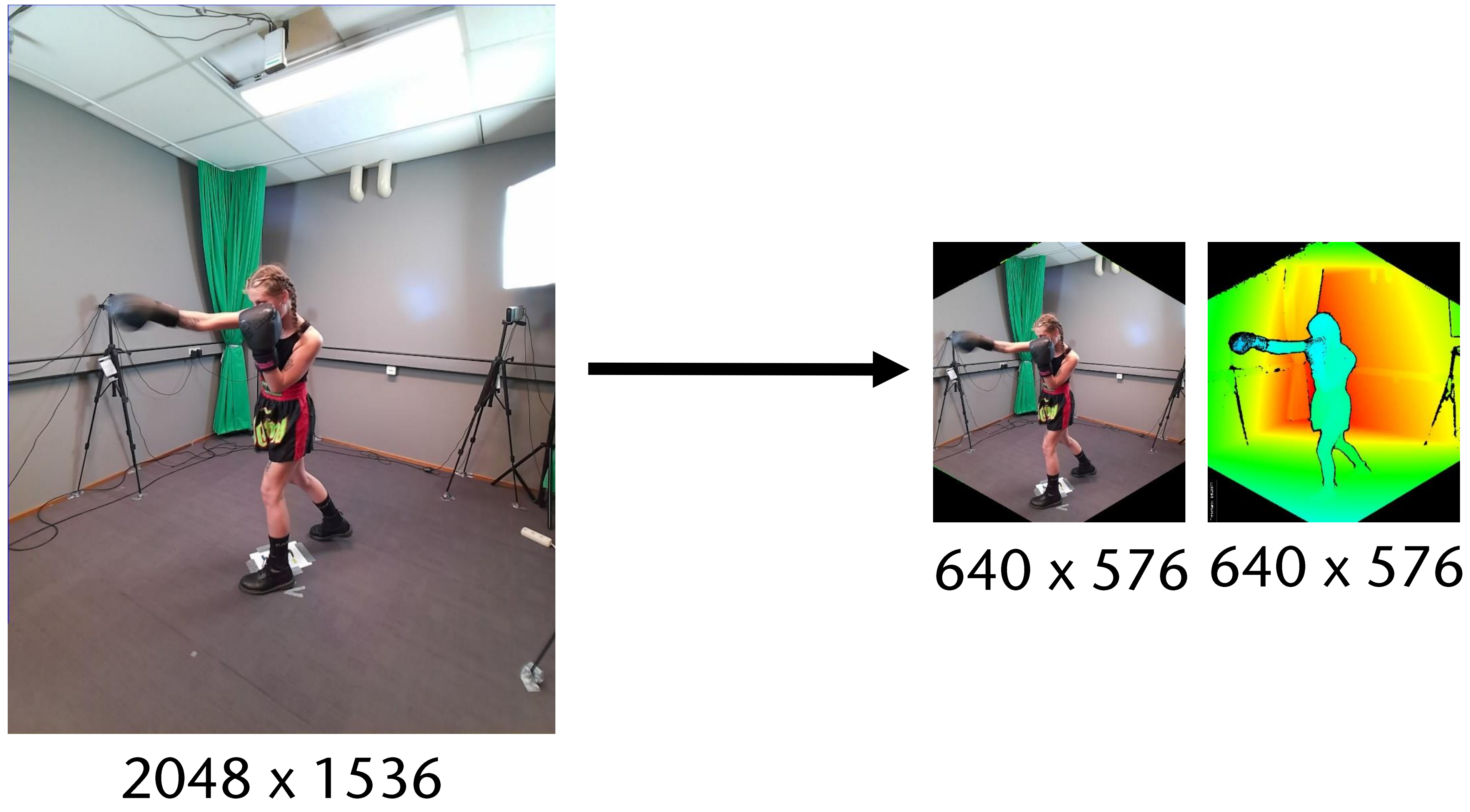
High Resolution (HR) Encoding

Further improving quality

High Resolution (HR) Encoding

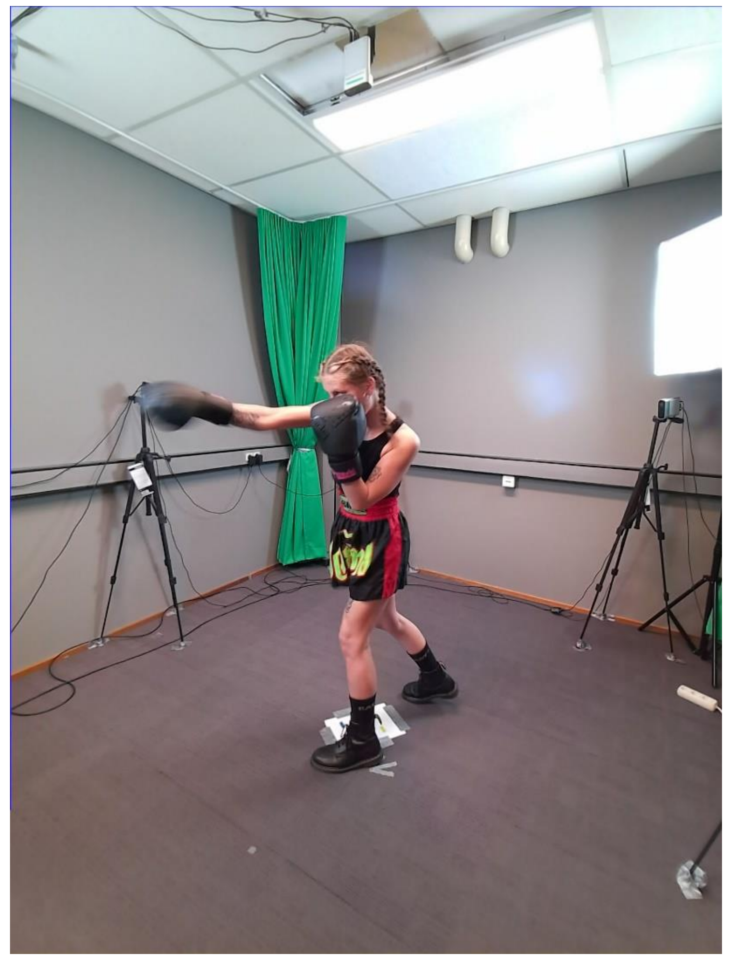
- When working with RGB-D cameras
  - Depth image can be mapped onto the color image
    - ➢ Point Cloud gets huge; bad for performance
  - Color image can be mapped onto the depth image
    - ➢ Color resolution is (significantly) reduced





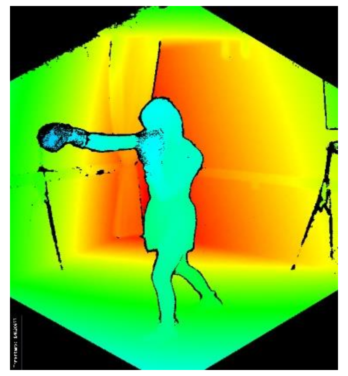
High Resolution (HR) Encoding

- 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

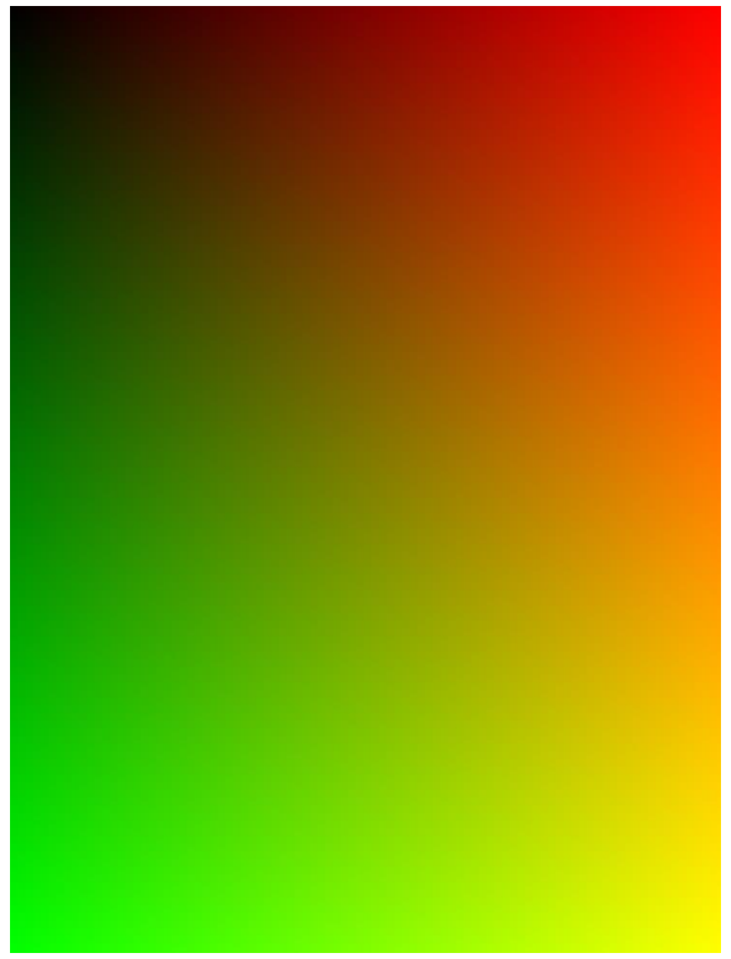
use as texture



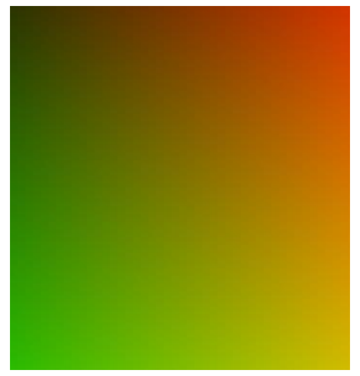
640 x 576

High Resolution (HR) Encoding

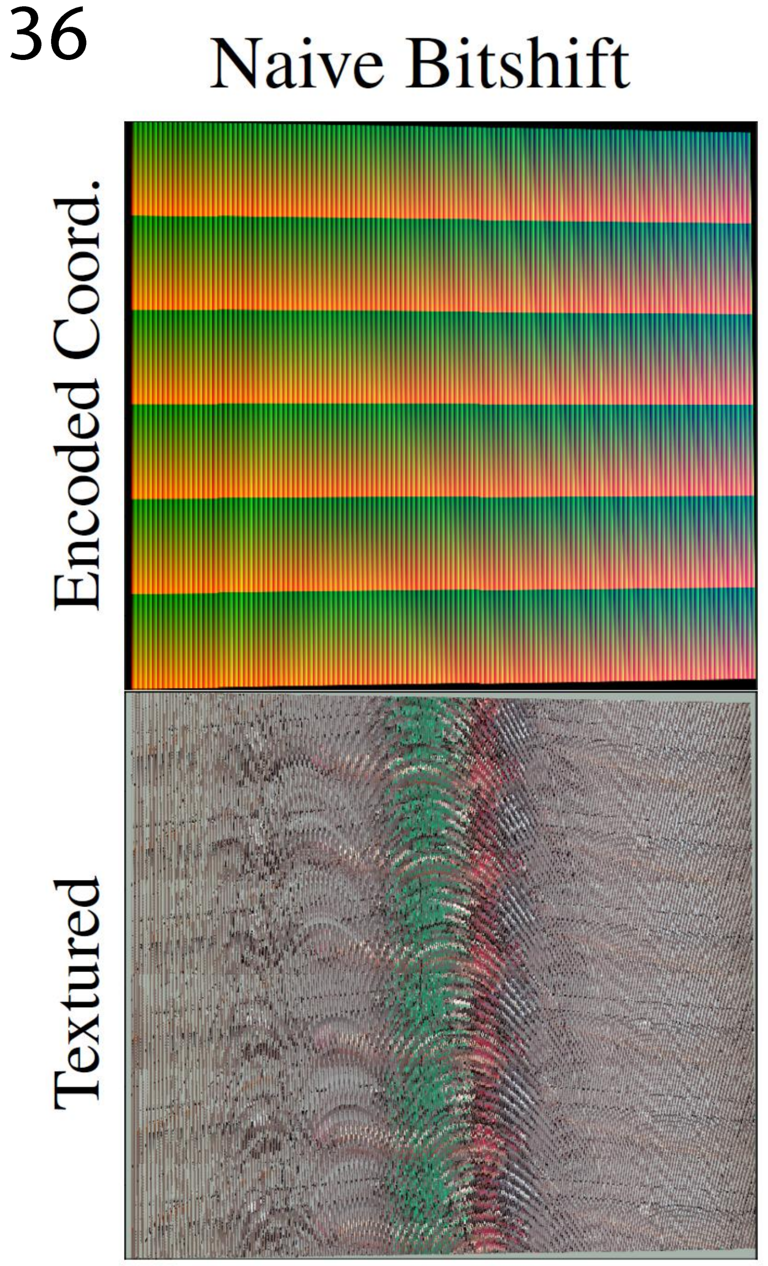
- Obtain UV coordinates:
  - We can transform an image of UV coordinates onto the depth image
  - However, the Azure Kinect SDK only allows to transform RGBA32 images onto depth
    - We can encode **2x 16 bit UV** into **4 x 8 bit RGBA**
    - However, due to linear interpolation, this information is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



2048 x 1536

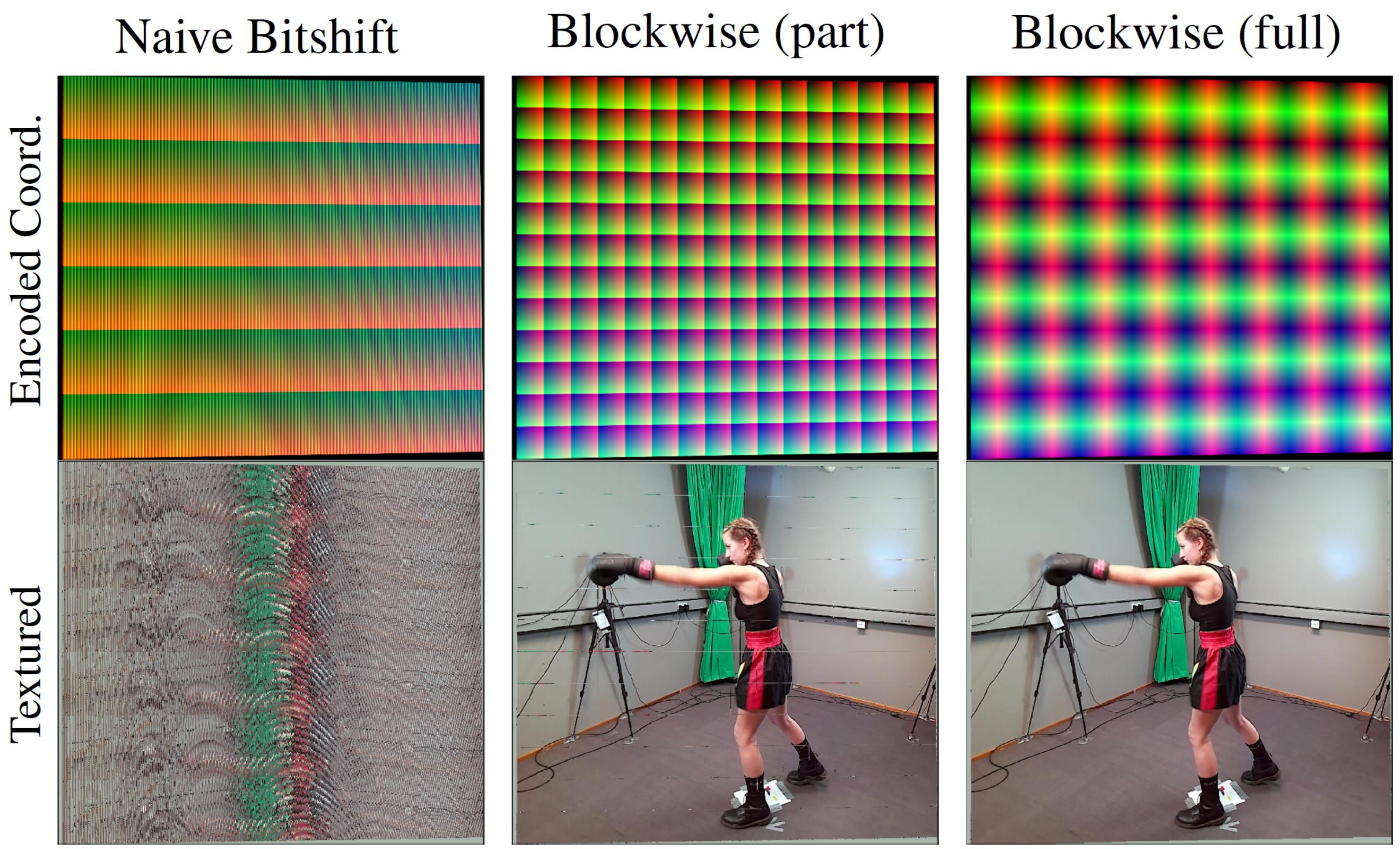


640 x 576



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

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

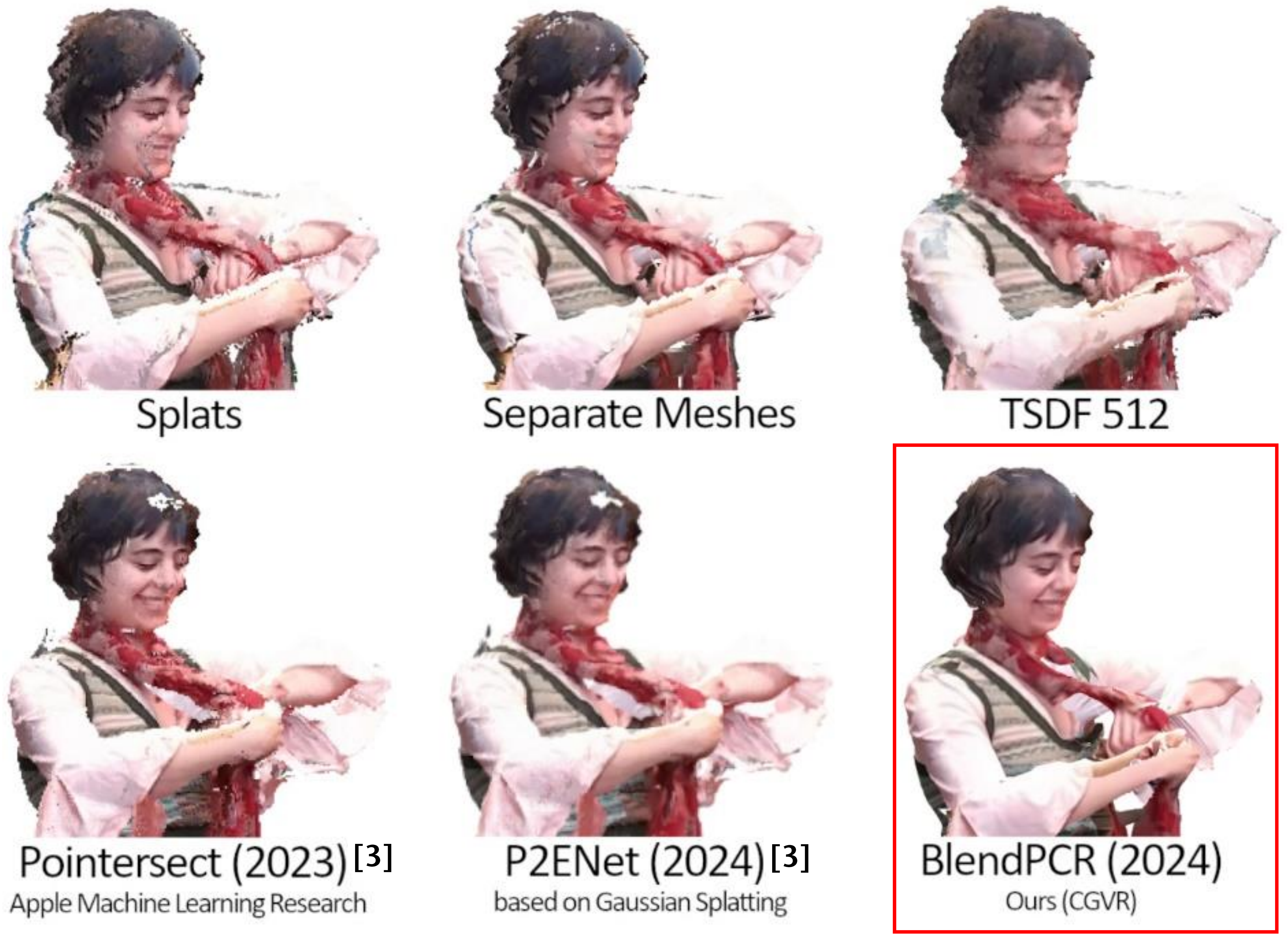
- CWIPC-SXR, S13 Card Trick Scene

Separate Meshes

BlendPCR

Results

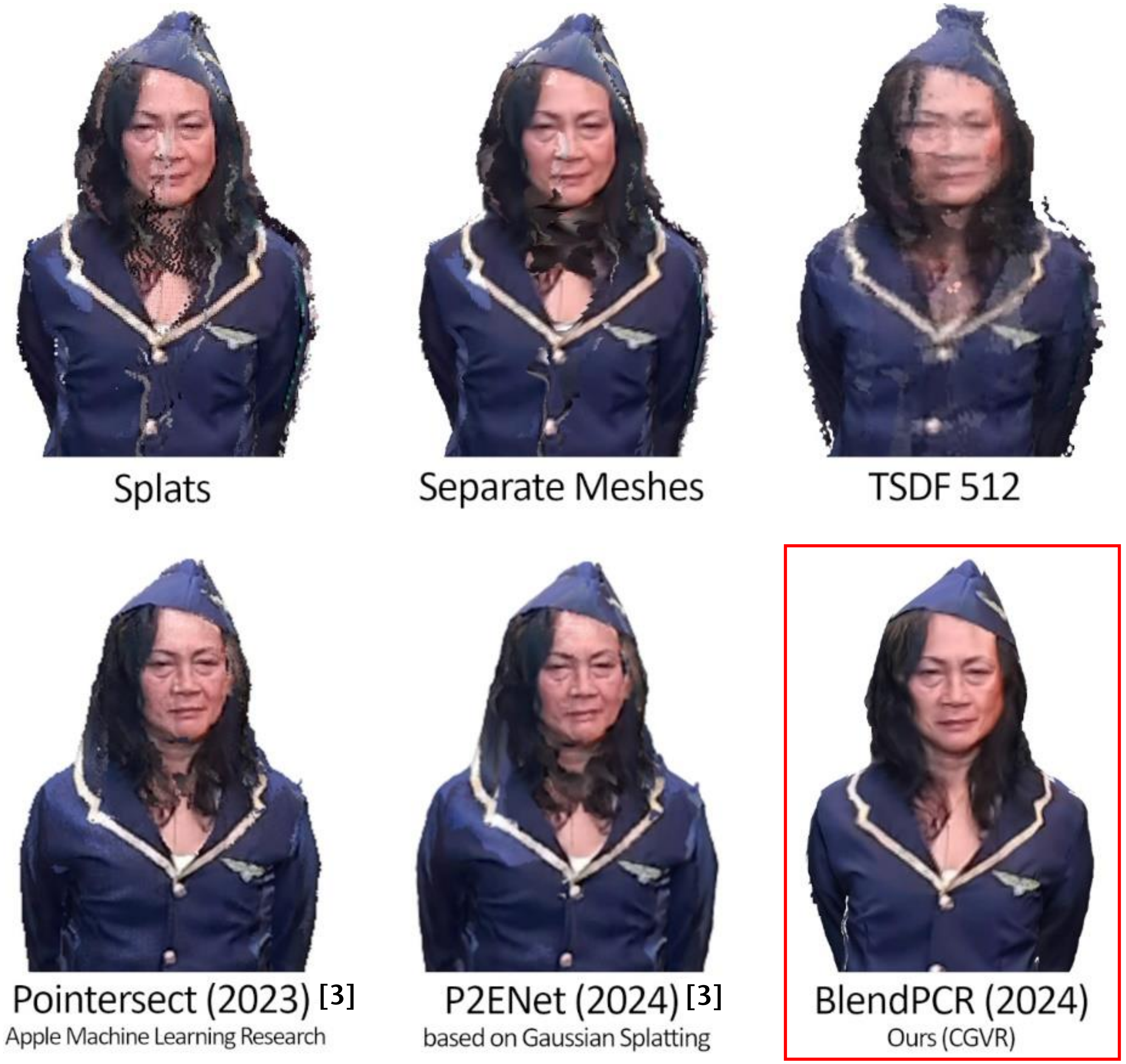
- CWIPC-SXR, S7 Scarf Dressing



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

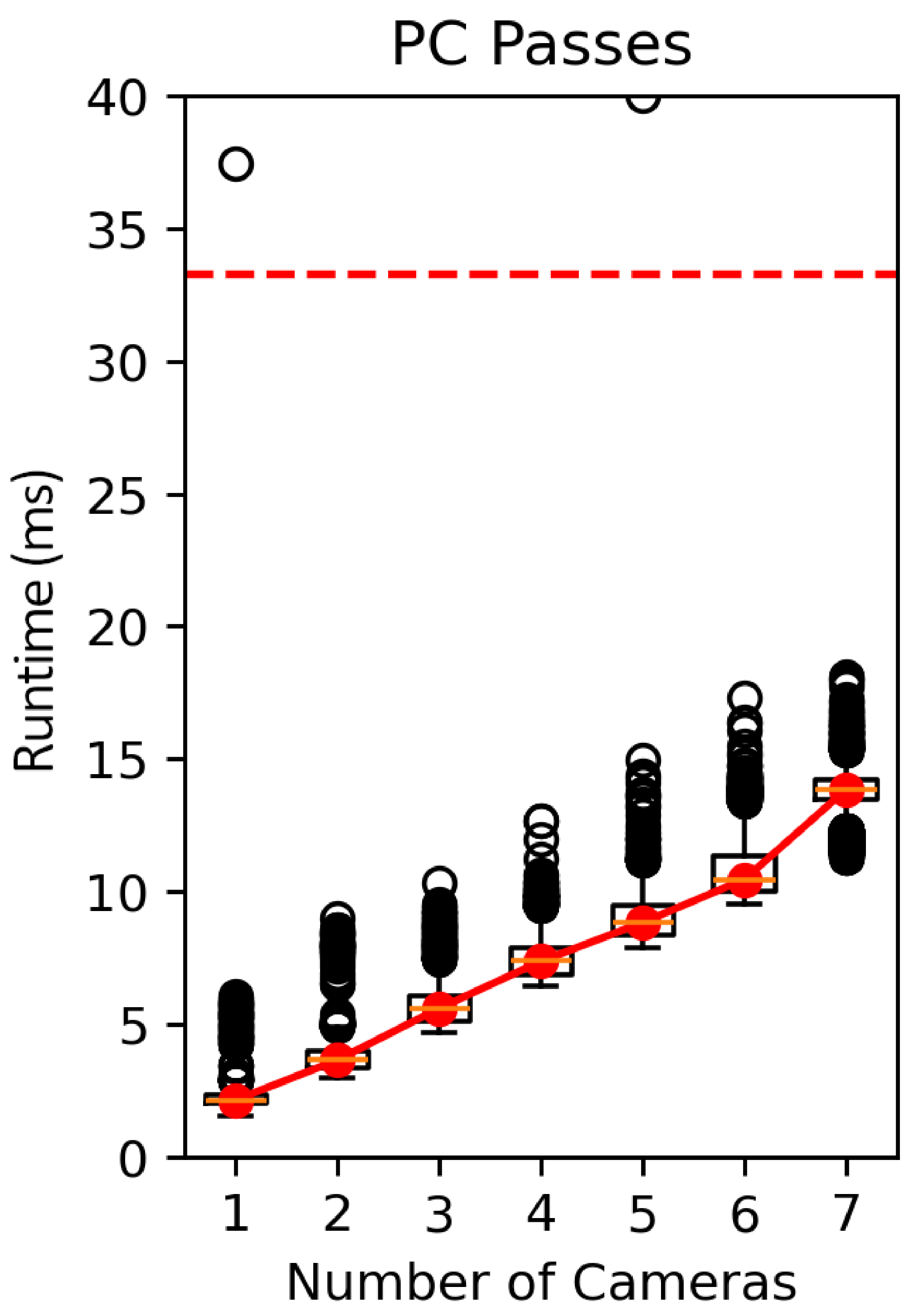
Results

- CWIPC-SXR, S3 Flight Attendant

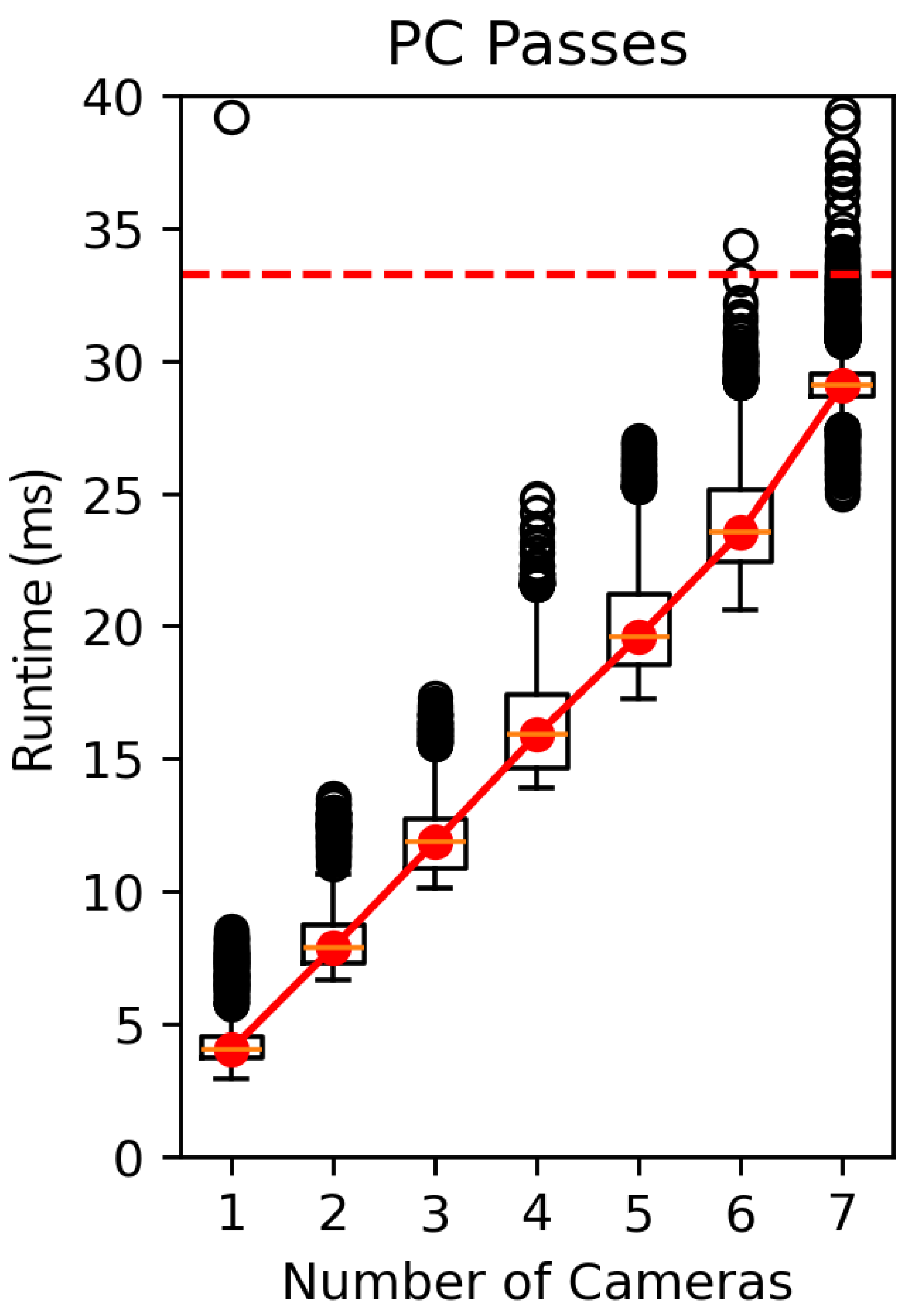


[3] From: Y Hu et al., "Low Latency Point Cloud Rendering with Learned Splatting," CVPR Workshop 2024.

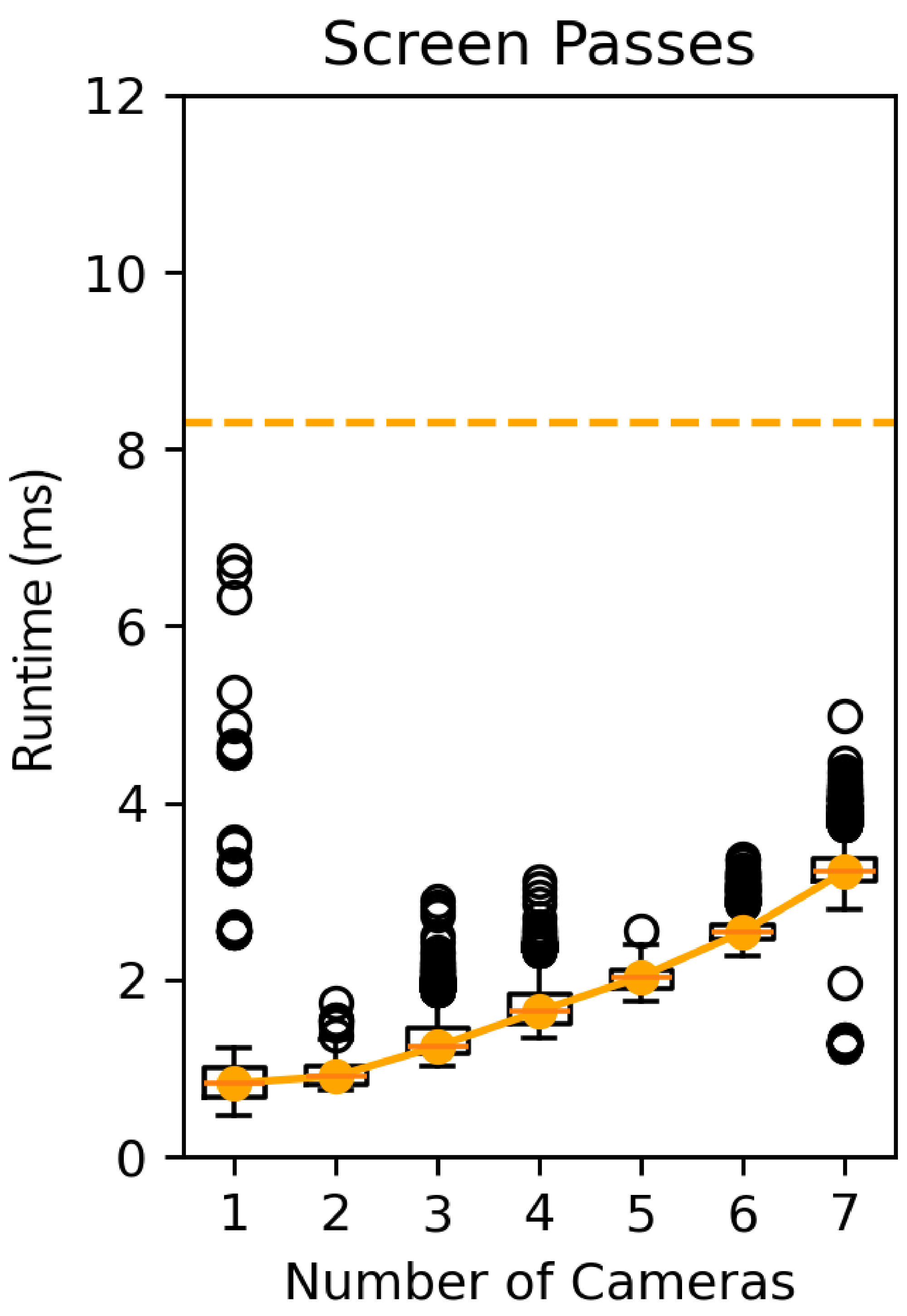
Performance



BlendPCR



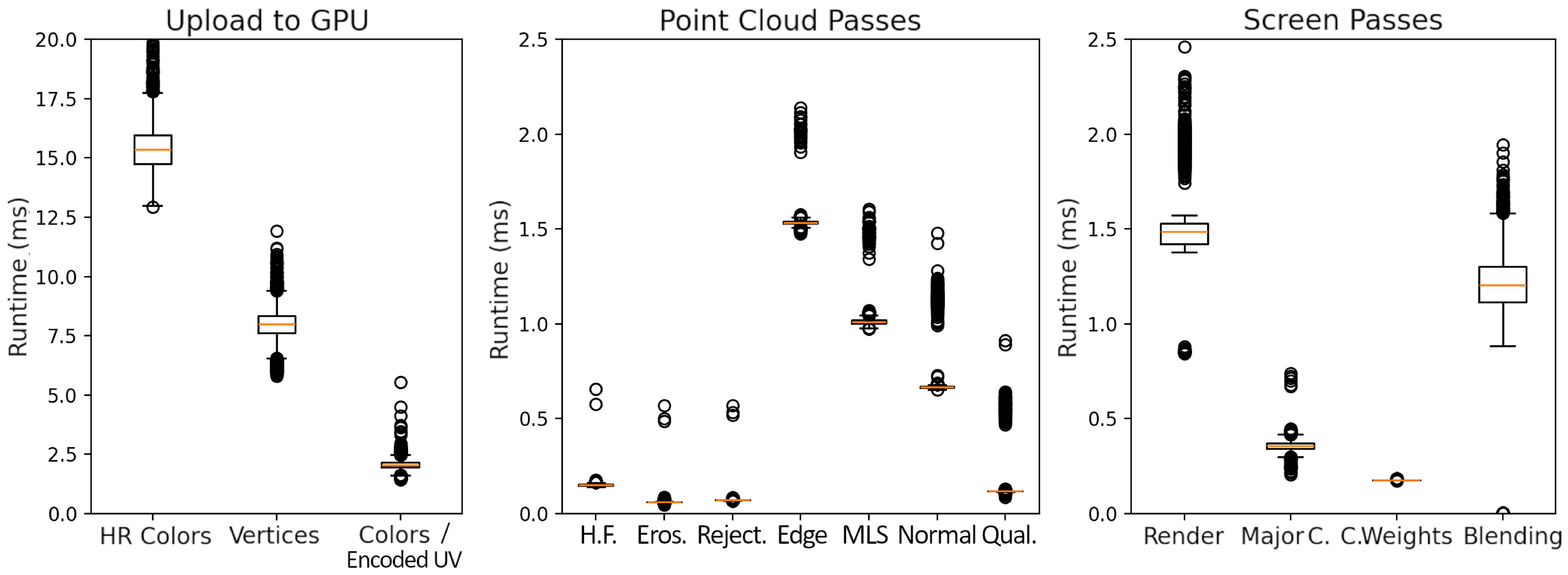
BlendPCR (HR)



Both

Using an NVIDIA GeForce RTX 4090 @ 3580 x 2066

Performance

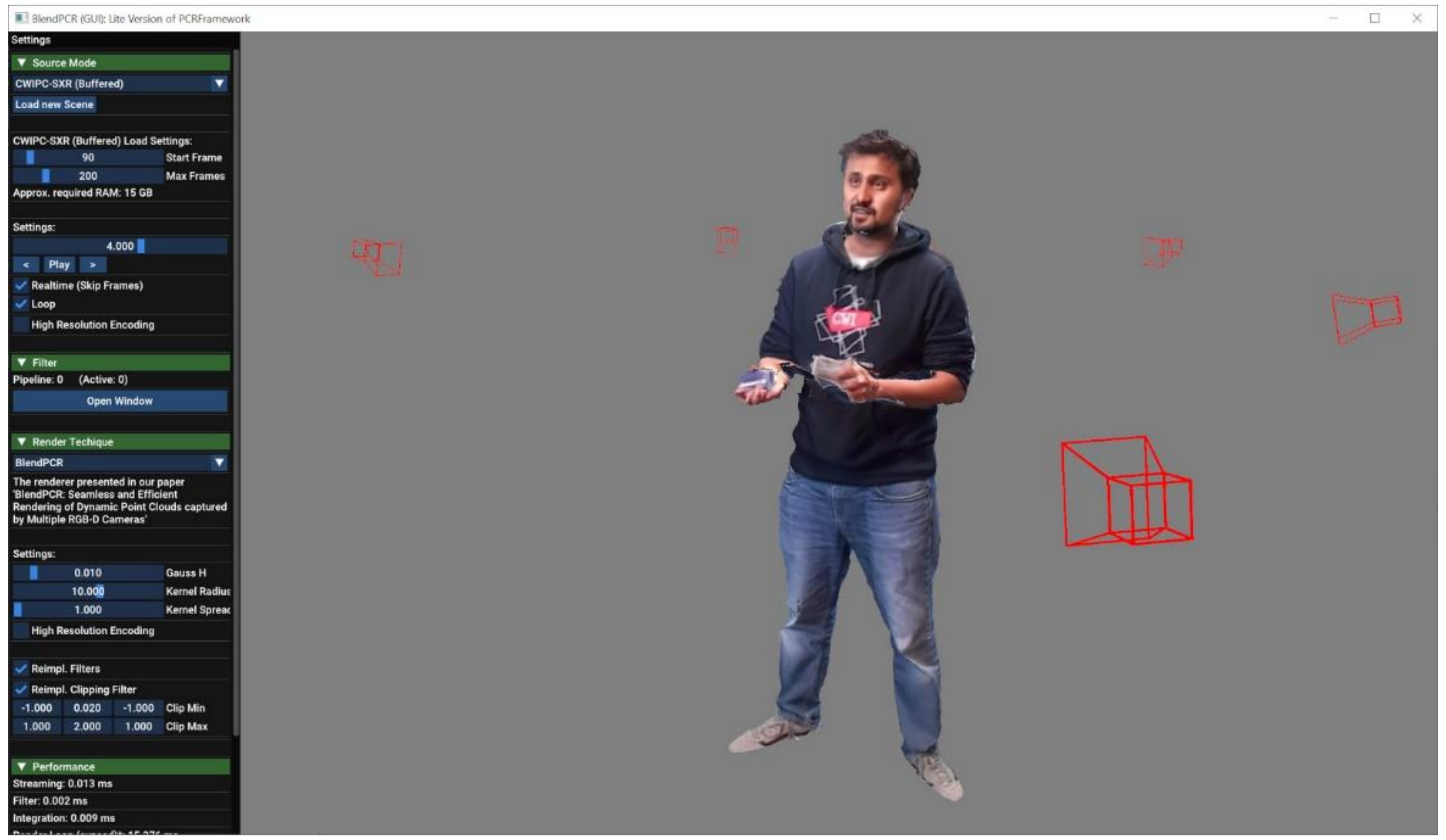


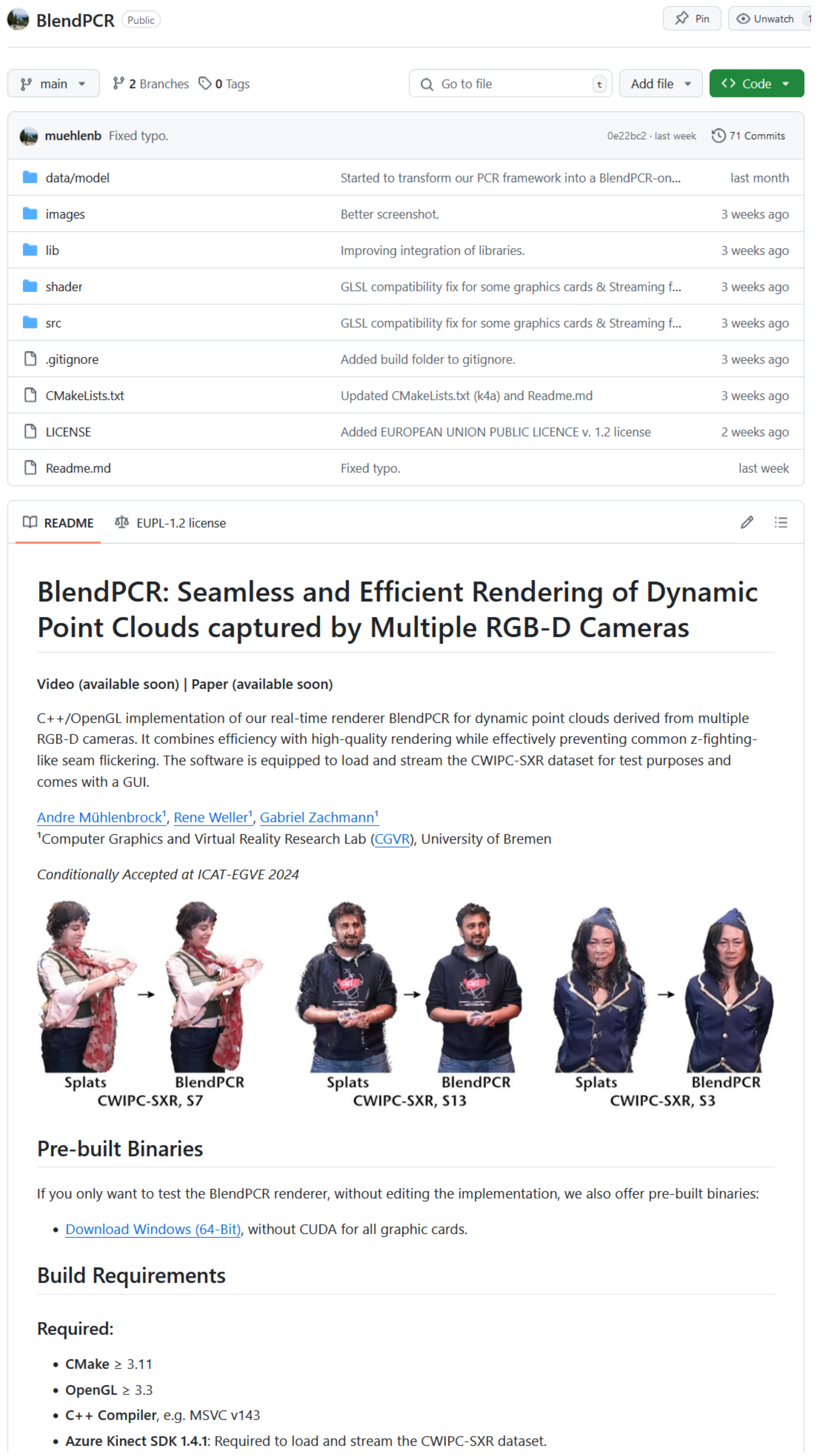
More details in the paper

Using an NVIDIA GeForce RTX 4090 @ 3580 x 2066

Source Code

- Github: https://github.com/muehlenb/blendpcr
  - ➢ Pure **OpenGL 3.3** implementation
  - ➢ Pre-built binaries available





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

Future Work

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

Pointersect (2023)
Apple Machine Learning Research

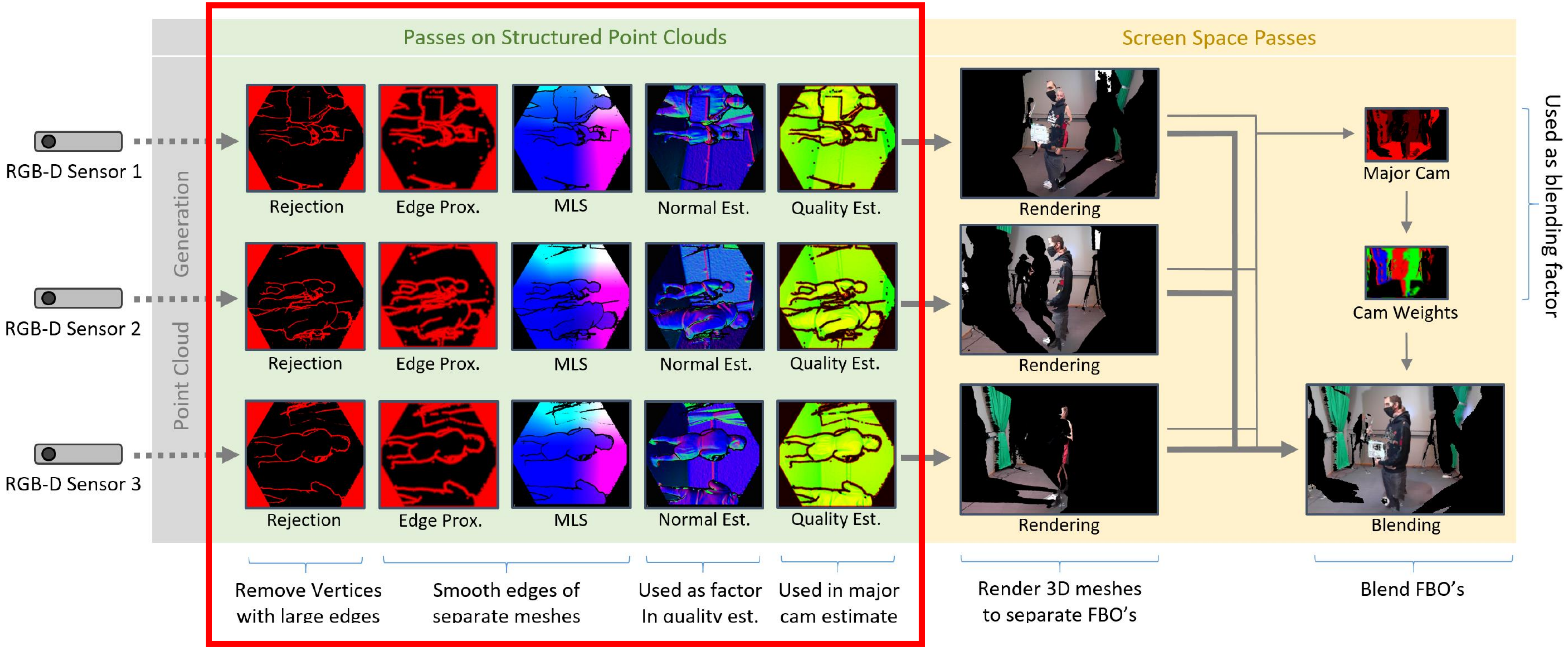
P2ENet (2024)
based on Gaussian Splatting

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.

Discussion

Thank you for your attention!

Separate Meshes

BlendPCR

Contact: muehlenb@uni-bremen.de