The HoloLens2ForCV Repository

D. Ungureanu, P. Sama, **F. Bogo**, S. Galliani

{dorinung, sasama, febogo, sigallia}@microsoft.com

Outline

- The HoloLens2ForCV Repository
- Building and Deploying UWP apps
- Samples: Sensor Visualization, Stream Recorder
- Cannon Library
- HoloLens for Computer Vision: Two examples

The HoloLens2ForCV Repository

- New project. Old HoloLensForCV repo not compatible with HoloLens 2!
- Apps and utilities for writing UWP apps using Research Mode
- Feedback and contributions welcome!

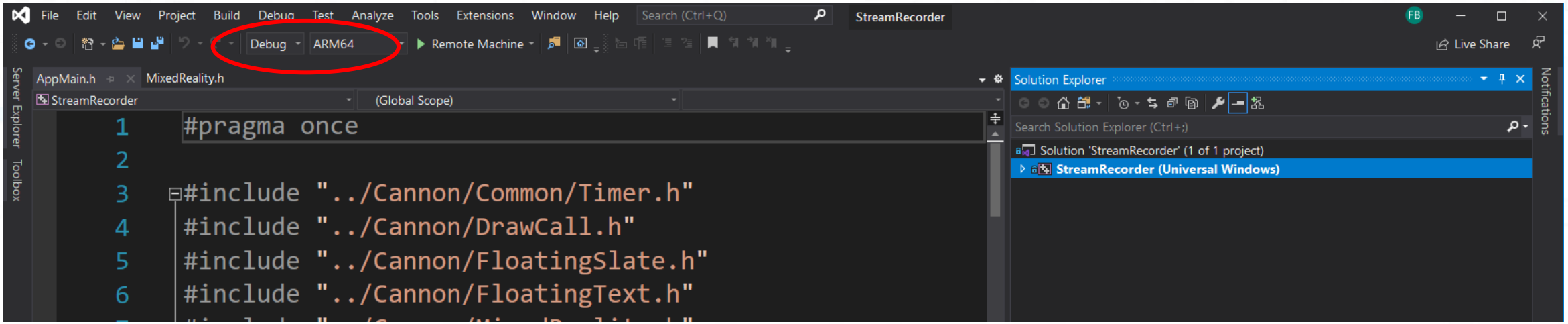
https://github.com/microsoft/HoloLens2ForCV

UWP Apps: Setup

- Follow the instructions at https://docs.microsoft.com/en-us/windows/mixed-reality/using-visual-studio
- Install the required tools (Visual Studio 2019, Windows 10 SDK)
- Don't forget to:
  - Enable Developer Mode on HoloLens (should be already enabled if you use Device Portal)
  - Enable Research Mode (https://docs.microsoft.com/en-us/windows/mixed-reality/research-mode#enabling-research-mode-hololens-1st-gen-and-hololens-2)

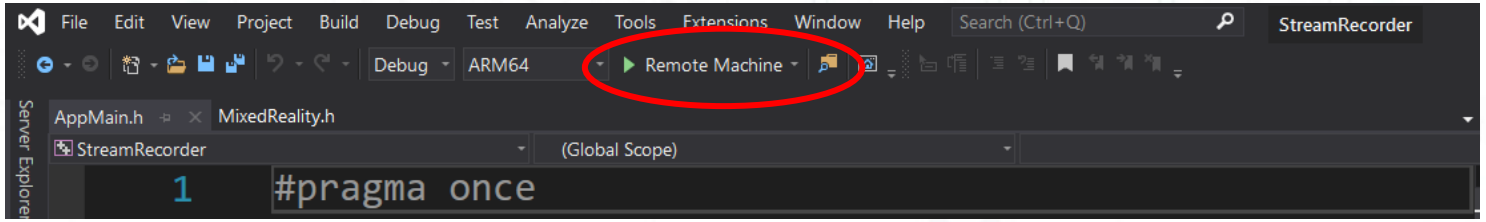
UWP Apps: Build

- From Visual Studio:
  - Open the Solution file
  - Choose ARM64 configuration
  - Build (e.g. Ctrl + Shift + B)

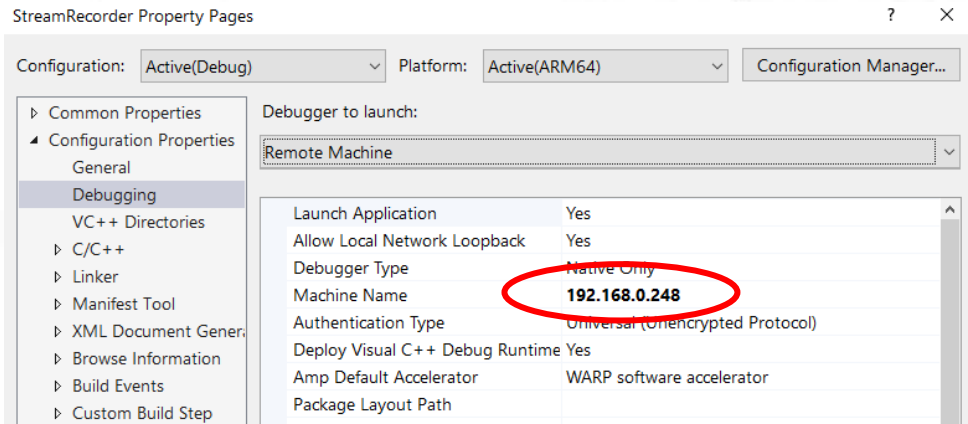


UWP Apps: Deploy

- Deploy on device over Wi-Fi or USB
- Over Wi-Fi (https://docs.microsoft.com/en-us/windows/mixed-reality/using-visual-studio#deploying-an-app-over-wi-fi--hololens-2):
  - Select Remote Machine



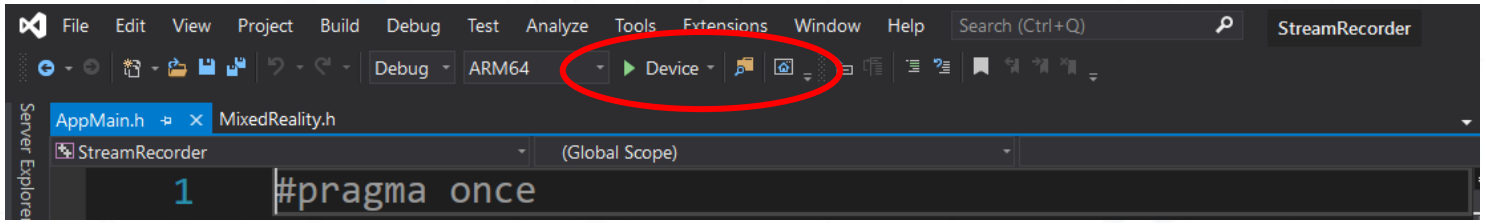
  - Got to Project → Properties → Configuration Properties → Debugging and set the HoloLens IP address



  - Deploy and start (F5 or Ctrl+F5)
  - The first time, device pairing might be required

UWP Apps: Deploy

- Deploy on device over Wi-Fi or USB
- Over USB (https://docs.microsoft.com/en-us/windows/mixed-reality/using-visual-studio#deploying-an-app-over-usb---hololens-2):
  - Select Device



  - Deploy and start (F5 or Ctrl+F5)
  - The first time, device pairing might be required

Sample Apps

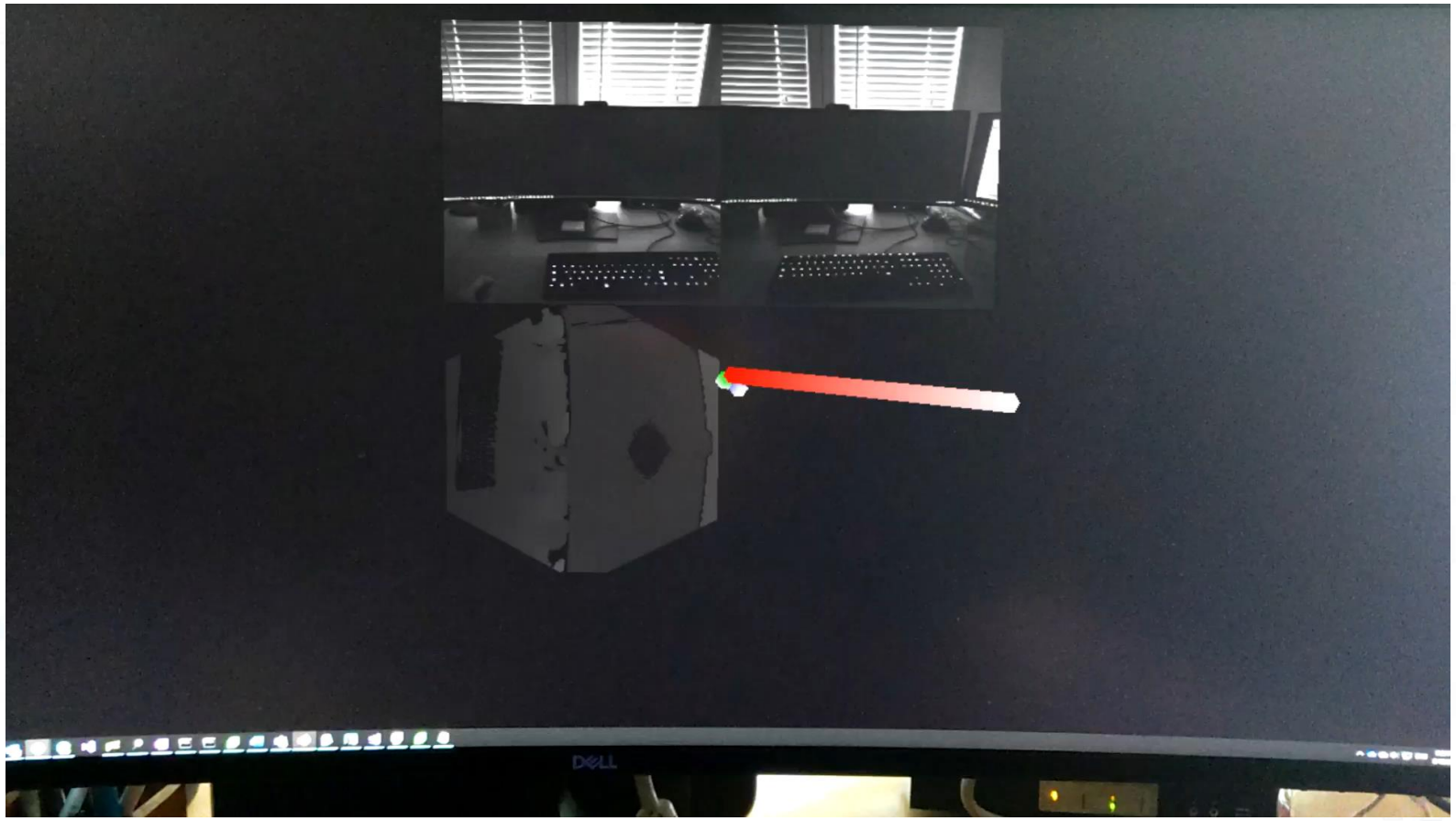
- Sensor Visualization:
  - Visualize Research Mode streams live on device
  - Calibration / arUco markers utilities
- Stream Recorder
  - Record Research Mode streams on device for offline postprocessing
  - Combine them with the HoloLens 2 RGB camera, head / hand / eye tracking

Sensor Visualization

- Example with the two frontal VLC cameras and Long Throw depth

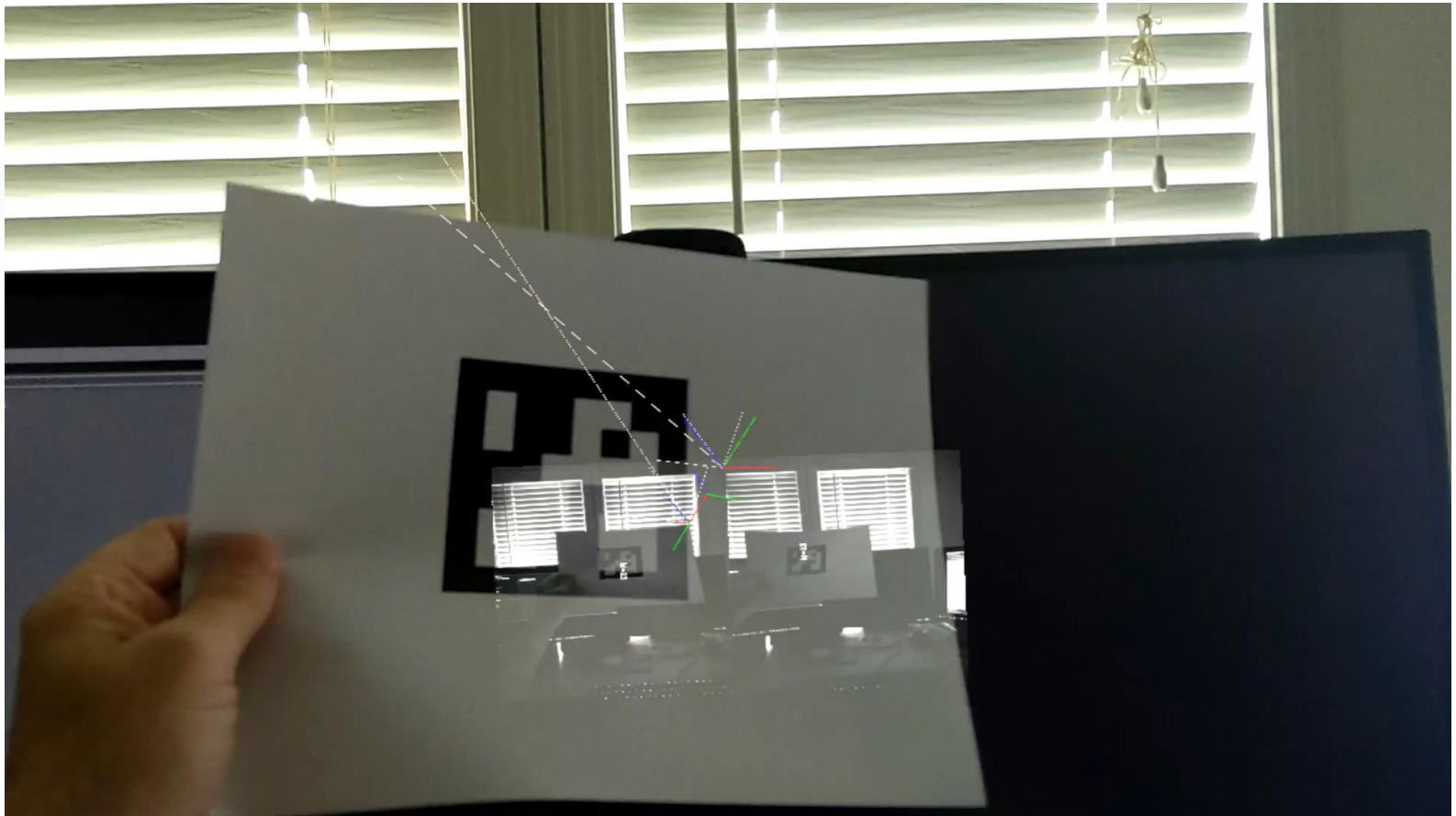
Sensor Visualization

- Example with cameras plus IMU acceleration visualization



OpenCV ArUco Detection and Triangulation

- Detection and triangulation of ArUco markers using OpenCV



Stream Recorder

- Captures streams and writes them to disk for offline postprocessing
- Caveat: Novice developer ☺
- Streams:
  - Research Mode Cameras (Long Throw depth, AHAT depth, VLC)
  - RGB Camera (PhotoVideo - PV)
  - Head, Hand, Eye tracking

Stream Recorder

- Two components:
  - UWP app (C++/winRT)
  - Utilities for downloading data from device / postprocessing (python)

Input Streams

- Streams to be selected in AppMain.cpp
- For example, to capture Long Throw Depth and PV (RGB):

AppMain.cpp

StreamRecorder → AppMain

```
13
14  std::vector<ResearchModeSensorType> AppMain::kEnabledRMStreamTypes = { ResearchModeSensorType::DEPTH_LONG_THROW };
15  std::vector<StreamTypes> AppMain::kEnabledStreamTypes = { StreamTypes::PV};
16
17
```

- IMU Sensors not used for now
- Different stream configurations require new build & deploy

Research Mode Streams

- Uses the RM API previously introduced
- *SensorScenario* initializes RM streams
- *RMCameraReader* implements threading logic for reading frames / writing them to disk
- Framerates:
  - Long Throw up to 5fps
  - AHAT up to 45fps
  - VLC up to 30fps

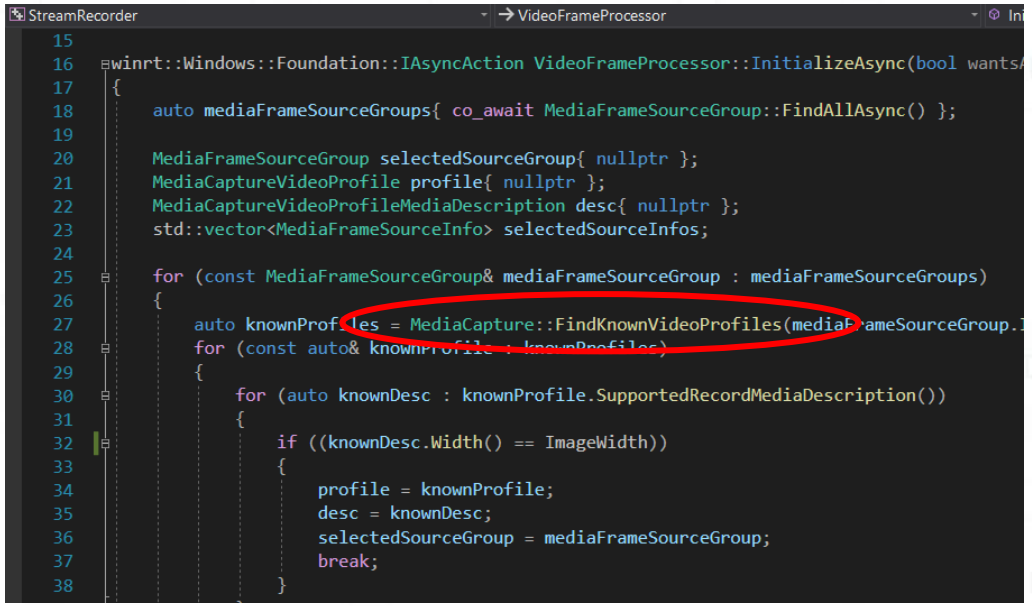
RMCameraReader.cpp

StreamRecorder → RMCameraReader

```
16
17  void RMCameraReader::CameraUpdateThread(RMCameraReader* pCameraReader, HANDLE camConsentGiven,
18                                          ResearchModeSensorConsent* camAccessConsent)
19  {
20      HRESULT hr = S_OK;
21
22      DWORD waitResult = WaitForSingleObject(camConsentGiven, INFINITE);
23
24      if (waitResult == WAIT_OBJECT_0)
25      {
26          switch (*camAccessConsent) { ... }
52      }
53      else
54      {
55          hr = E_UNEXPECTED;
56      }
57
58      if (SUCCEEDED(hr))
59      {
60          hr = pCameraReader->m_pRMSensor->OpenStream();
61
62          if (FAILED(hr)) { ... }
67
68          while (!pCameraReader->m_fExit && pCameraReader->m_pRMSensor)
69          {
70              HRESULT hr = S_OK;
71              IResearchModeSensorFrame* pSensorFrame = nullptr;
72
73              hr = pCameraReader->m_pRMSensor->GetNextBuffer(&pSensorFrame);
74
75              if (SUCCEEDED(hr))
76              {
77                  std::lock_guard<std::mutex> guard(pCameraReader->m_sensorFrameMutex);
78                  if (pCameraReader->m_pSensorFrame)
79                  {
80                      pCameraReader->m_pSensorFrame->Release();
81                  }
82                  pCameraReader->m_pSensorFrame = pSensorFrame;
```

RGB (PV) Stream

- Uses *Windows Media* APIs
  (e.g. see https://docs.microsoft.com/en-us/windows/uwp/audio-video-camera/process-media-frames-with-mediaframereader)
- *VideoFrameProcessor* initializes the stream and implements threading logic for reading frames / writing them to disk
- Resolution / framerate can be customized
  using *MediaCapture::FindKnownVideoProfiles*
  (https://docs.microsoft.com/en-us/windows/mixed-reality/locatable-camera)
- With default settings, framerate up to 30fps



Head, Hand, Eye Tracking

- Accessed via Windows APIs, relying on the Cannon library as wrapper
- Tracking results fetched at each *AppMain Update* call (up to 60fps)
- Fully articulated hand tracking: 26 hand joints (translation + rotation)

AppMain.cpp

StreamRecorder → AppMain

```
102
103     if (m_recording)
104     {
105         HeTHaTFrame frame;
106         frame.headTransform = m_hands.GetHeadTransform();
107         for (int j = 0; j < (int)HandJointIndex::Count; ++j) {
108             frame.leftHandTransform[j] = m_hands.GetOrientedJoint(0, HandJointIndex(j));
109             frame.rightHandTransform[j] = m_hands.GetOrientedJoint(1, HandJointIndex(j));
110         }
111         frame.leftHandPresent = m_hands.IsHandTracked(0);
112         frame.rightHandPresent = m_hands.IsHandTracked(1);
113         frame.timestamp = m_mixedReality.GetPredictedDisplayTime();
```

Eye Tracking

- Eye gaze tracking: origin + direction
- Distance from the origin computed by querying the Surface Mapping interface (will not work for virtual objects!)

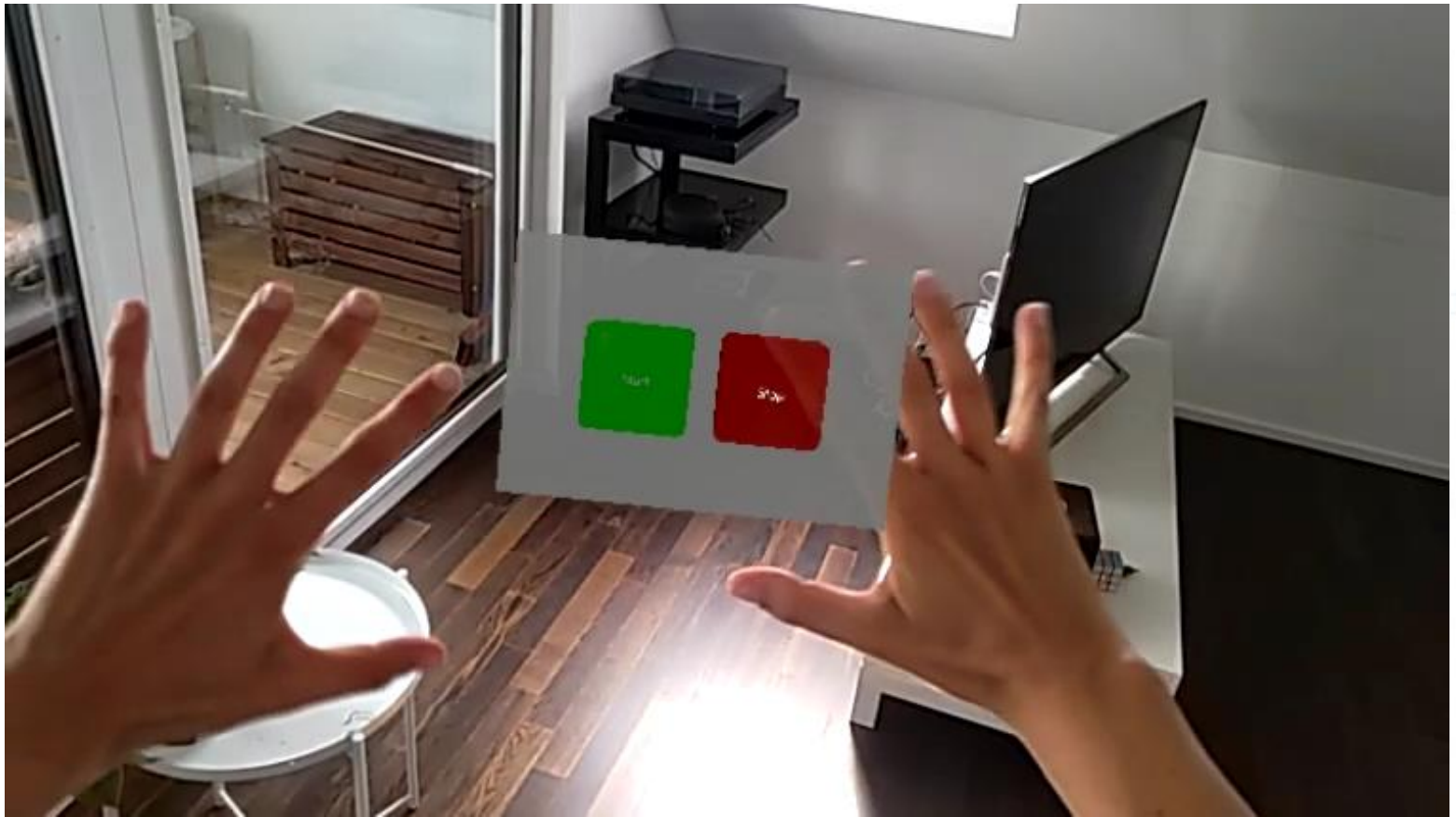
AppMain.cpp

StreamRecorder → AppMain → Update()

```
114
115         if (m_mixedReality.IsEyeTrackingEnabled() && m_mixedReality.IsEyeTrackingActive())
116         {
117             frame.eyeGazePresent = true;
118             frame.eyeGazeOrigin = m_mixedReality.GetEyeGazeOrigin();
119             frame.eyeGazeDirection = m_mixedReality.GetEyeGazeDirection();
120             if (m_mixedReality.IsSurfaceMappingActive())
121             {
122                 float distance;
123                 XMVECTOR normal;
124                 if (m_mixedReality.GetSurfaceMappingInterface()->TestRayIntersection(frame.eyeGazeOrigin, frame.eyeGazeDirection, distance, normal))
125                 {
126                     frame.eyeGazeDistance = distance;
127                 }
128                 else
129                 {
130                     frame.eyeGazeDistance = 0.0f;
131                 }
132             }
133         }
```

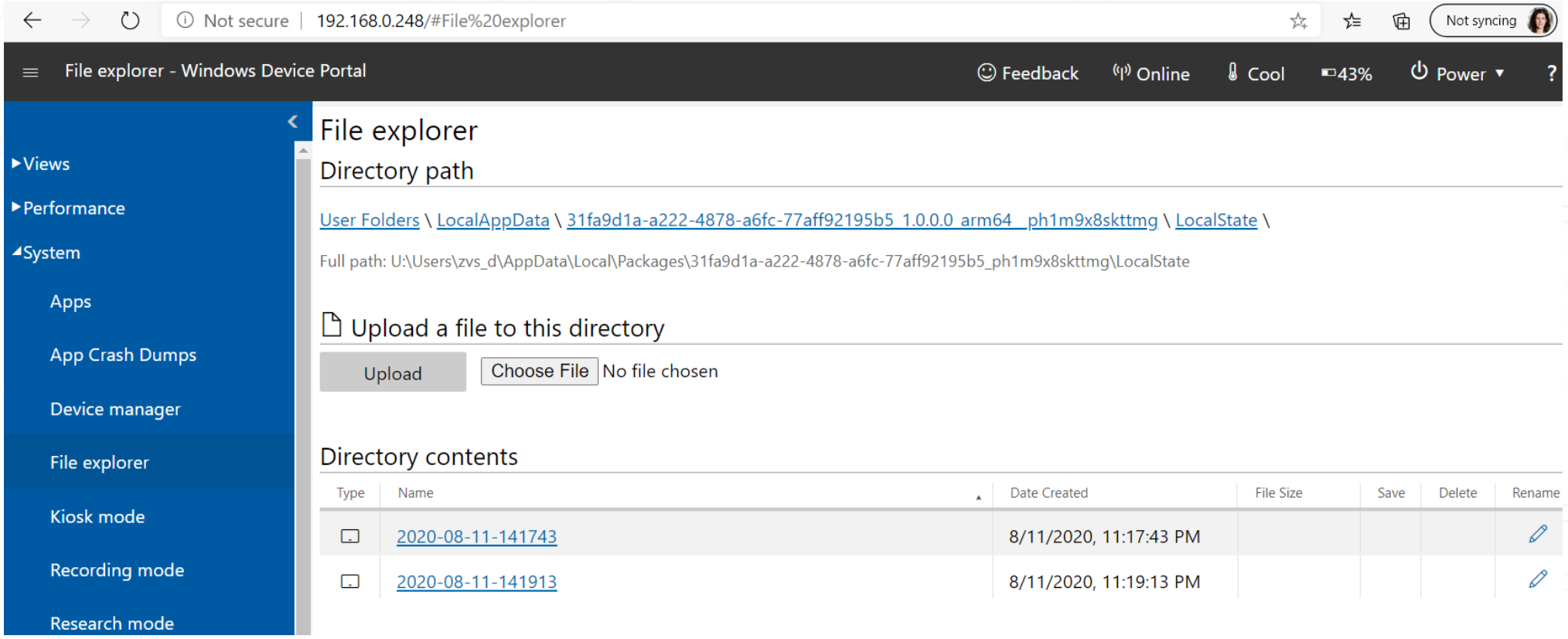
User Interface

- Two buttons for Start and Stop capture



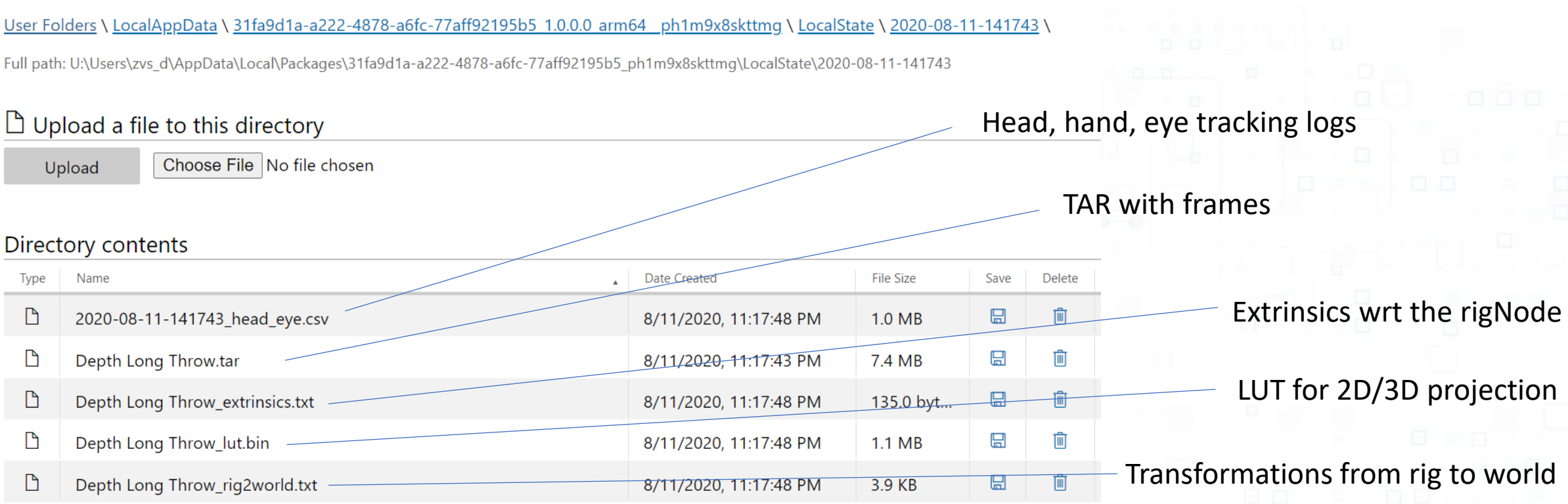
Retrieving Data on Device

- Captured data can be accessed / downloaded via Device Portal
- System → FileExplorer → LocalAppData → StreamRecorder → LocalState



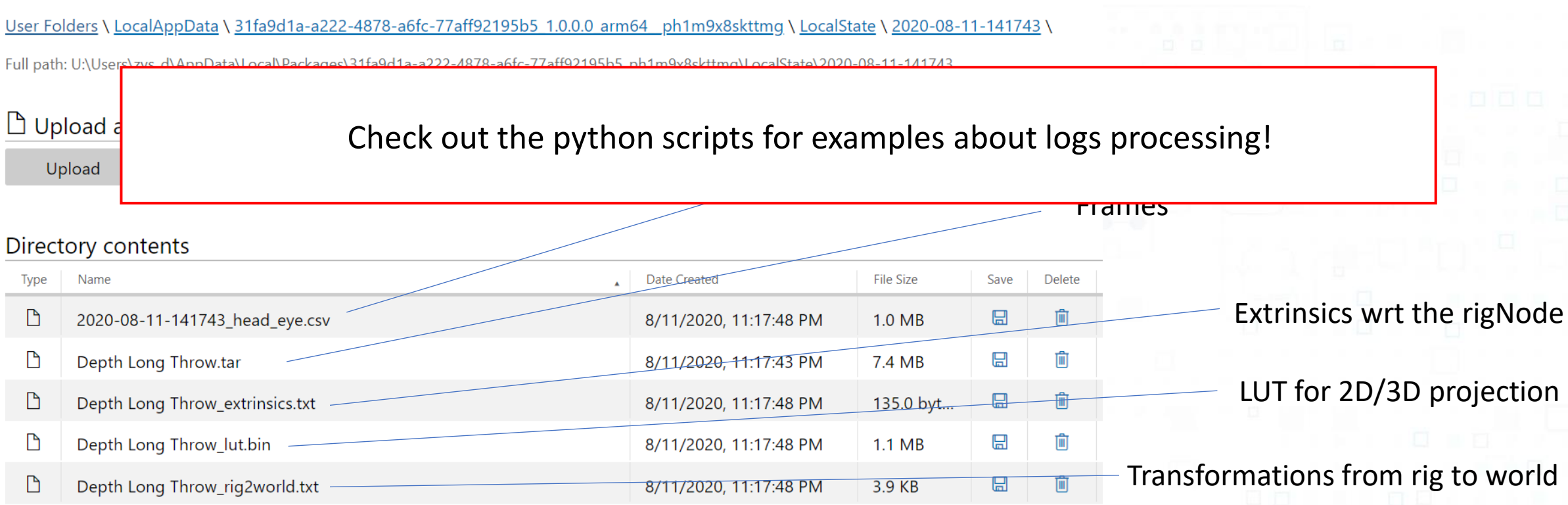
Retrieving Data on Device

- Example capture folder



Retrieving Data on Device

- Example capture folder



Head, Hand, Eye tracking logs

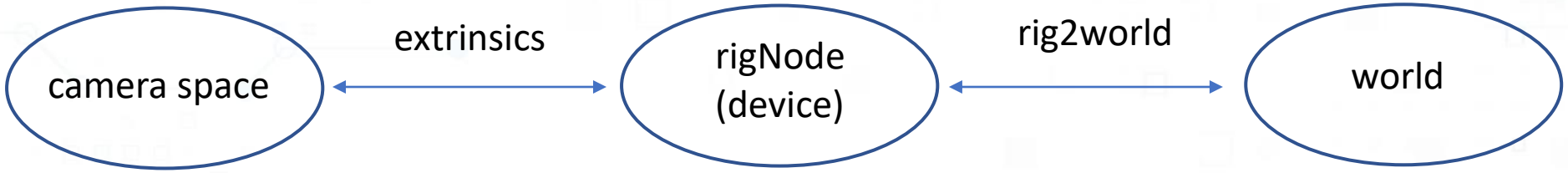
- One row per frame, collecting:
  - Timestamp in FILETIME
  - Head 6dof
  - Hand data: joint transforms if tracked (for both left and right hands)
    Tracking might be lost e.g. in proximity of surfaces!
  - Eye gaze data (if captured): origin / direction / distance
- Data in "*world space*" (origin depends on device location at app launch time)

Research Mode Camera Parameters

- Camera- (not capture-) dependent
- Intrinsics:
  - Provided as a LUT (lookup table) saved in binary
  - LUT allows one to convert from 2D pixels to 3D points in camera space by simply multiplying
- Extrinsics:
  - Provided as a 4x4 transformation wrt the *rigNode*
  - Identity transformation for the VLC LF camera

From Camera to World Space

- To go from camera space to world space, one can use the *rig2world* file
- One row per frame, collecting:
  - Frame timestamp in FILETIME
  - 4x4 transformation from *rigNode* to world space, obtained via Perception Spatial APIs (leverages the HoloLens 2 head tracking)



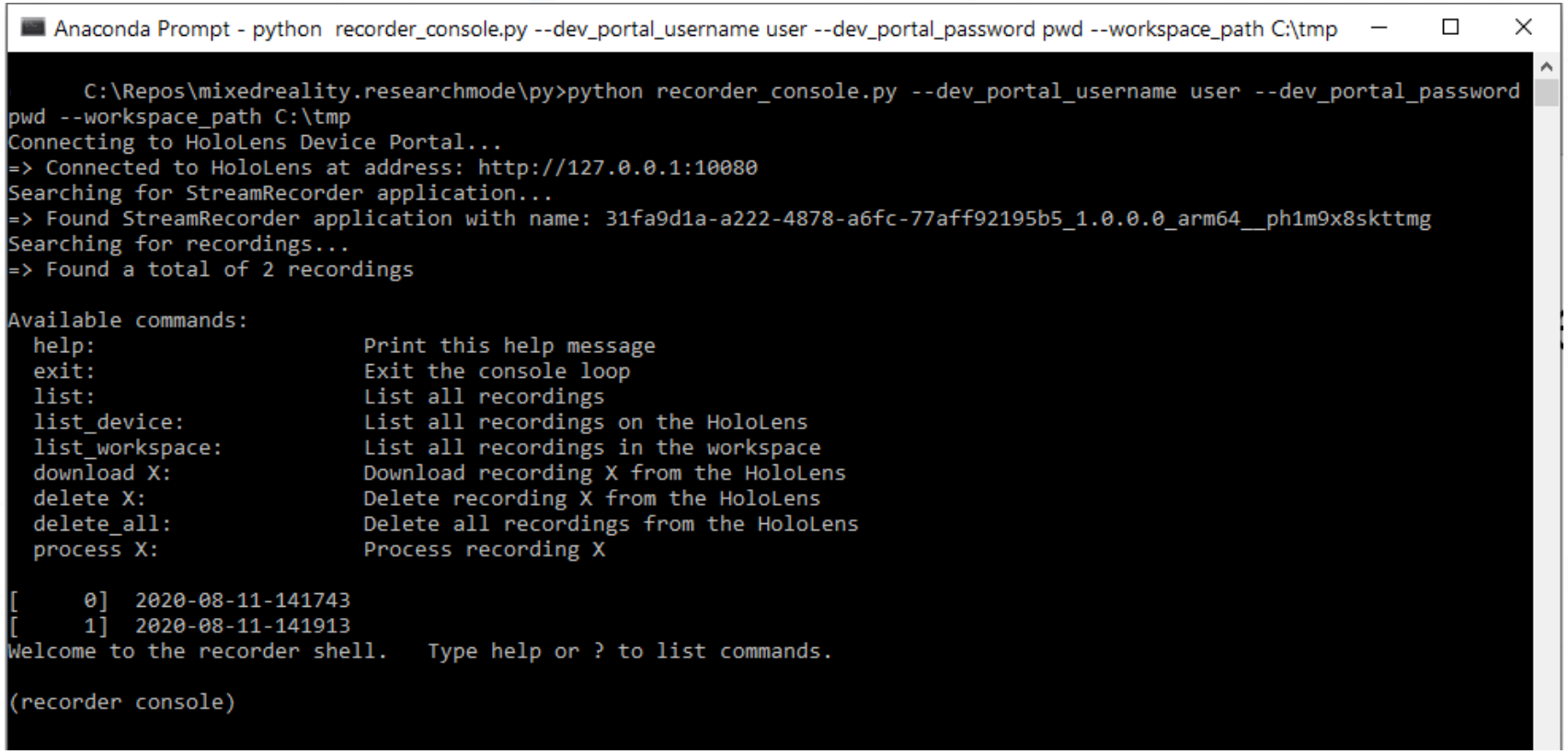
PV (RGB) Camera Parameters

- PV.txt log file:
  - First line: Principal point, image width and height
  - Then, one row per frame:
    - Focal length (changes over time!)
    - Camera rotation and translation in world space

Same coordinate system as Research Mode streams, hand and eye gaze!

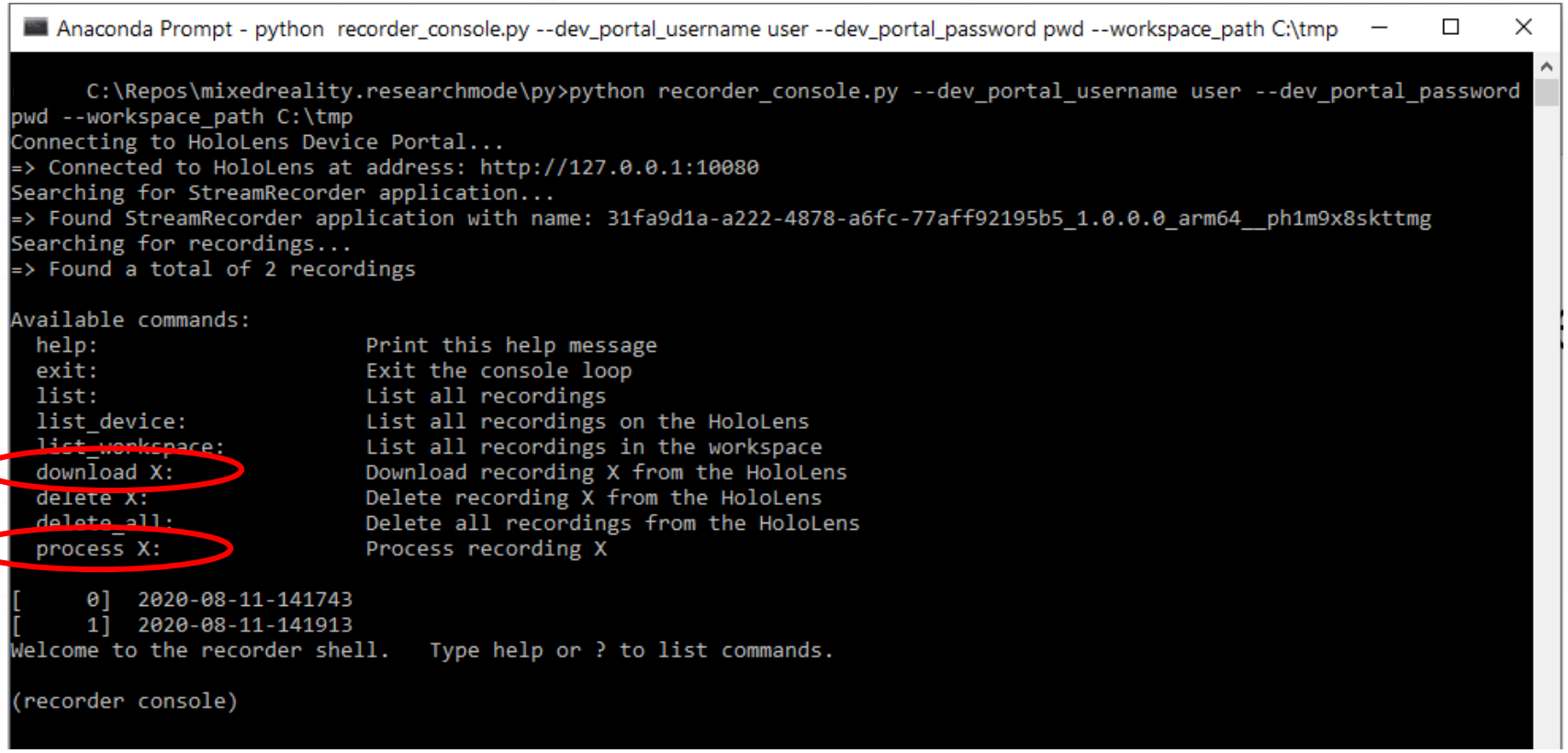
Using Recorder Console

- An alternative to manual download from Device Portal
- Download and postprocessing made easy!



Using Recorder Console

- An alternative to manual download from Device Portal
- Download and postprocessing made easy!

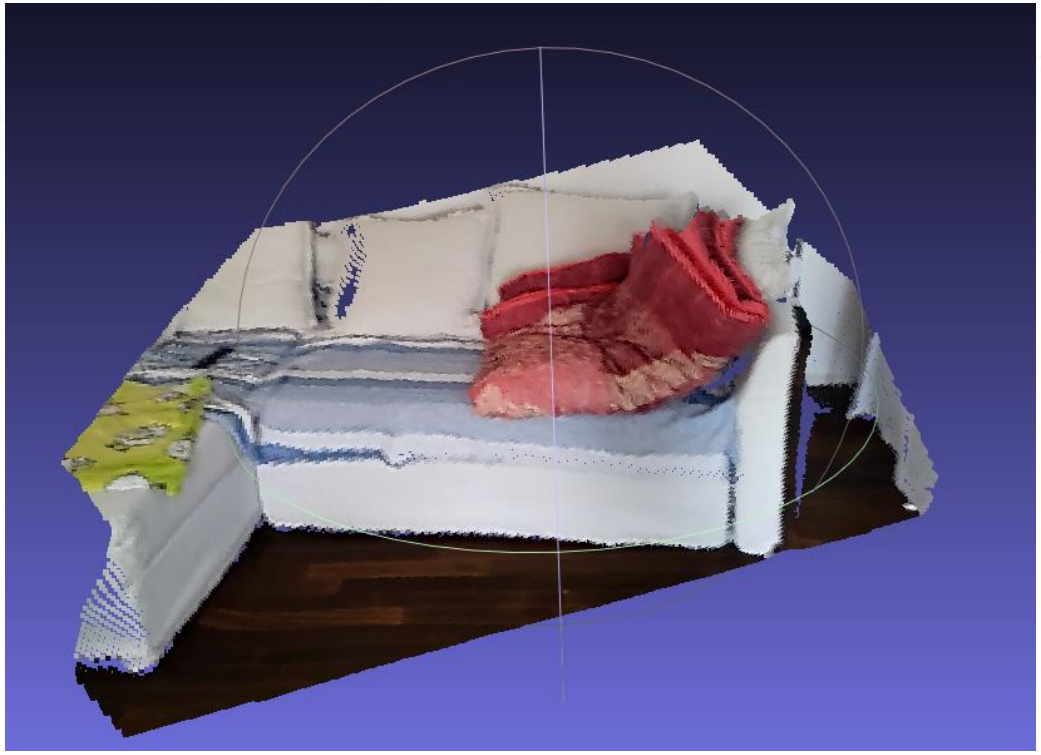


Using Recorder Console

- Automated processing:
  - Extract images from TAR files
  - Compute point clouds from depth, compute per-point color (if RGB is available), put them in world space
  - Project hand and eye gaze tracking results on RGB images

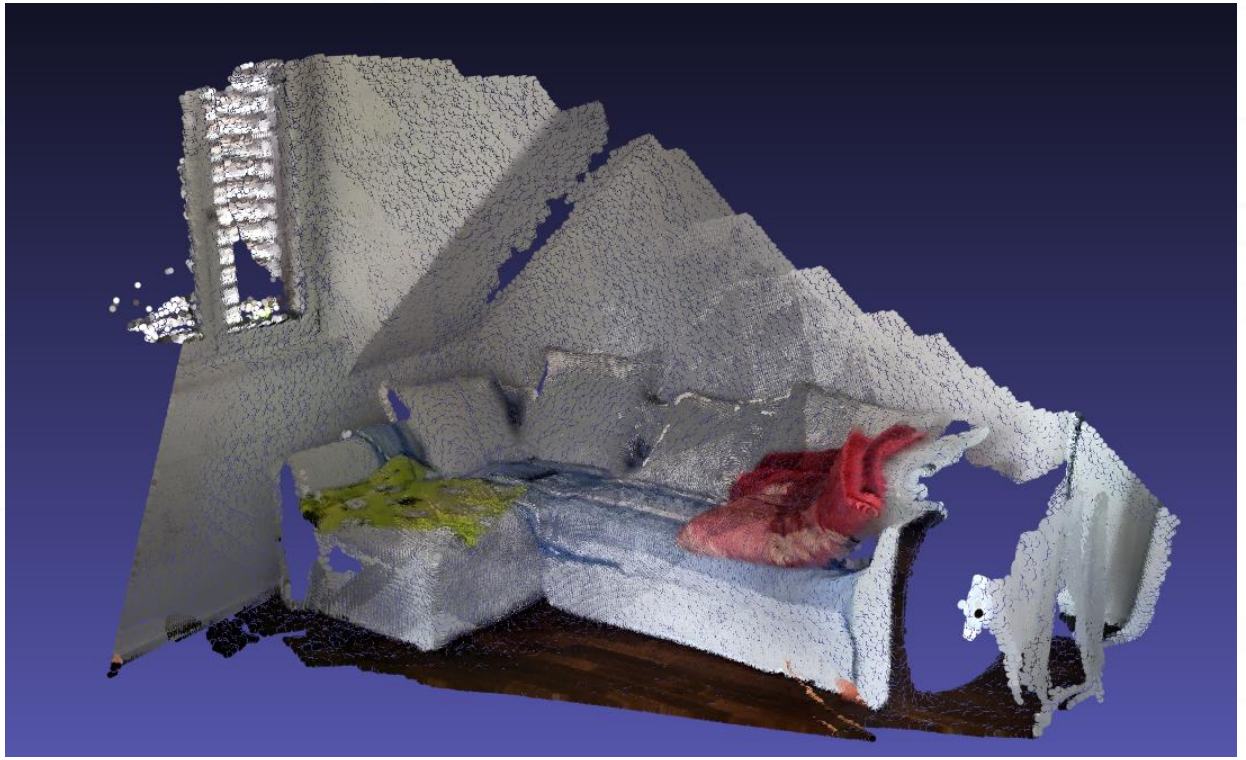
Long Throw Depth Processing

- Compute per-frame point clouds from Depth and PV (RGB) frames



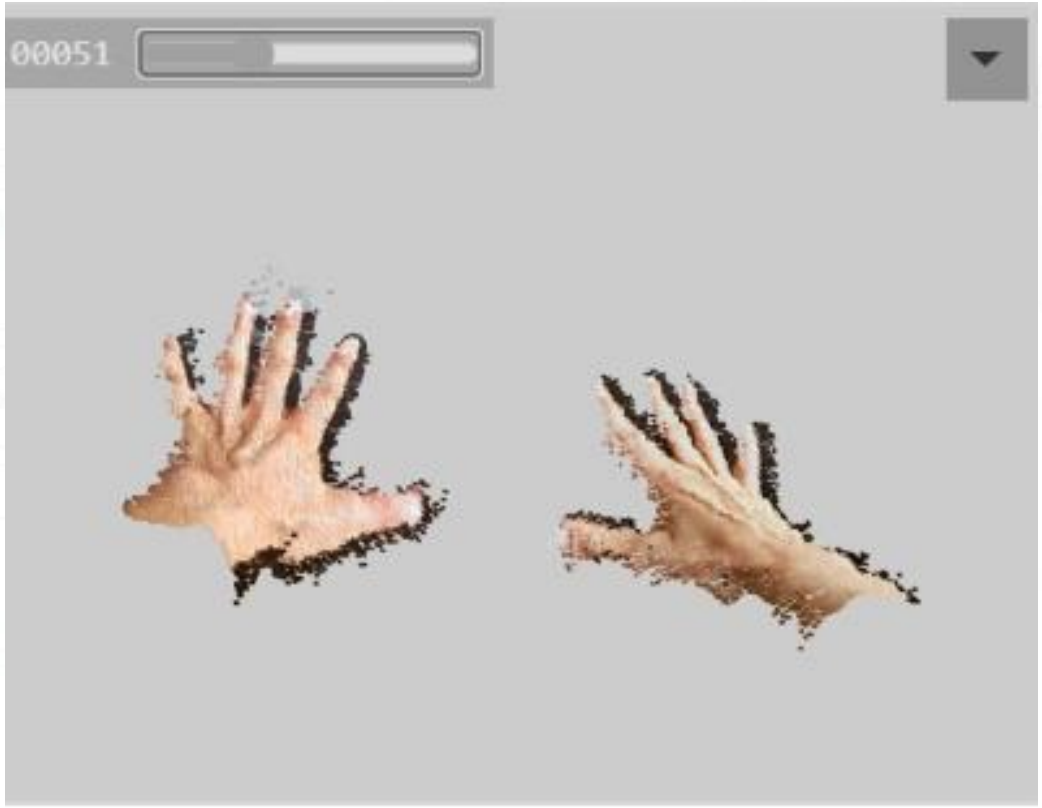
Long Throw Depth Processing

- Scene reconstruction by placing the point clouds in world coordinate system



AHAT Depth Processing

- AHAT sequence



Recall that AHAT captures might suffer from the “wrapping” problem

Naïve matching between AHAT-RGB based on closest timestamp

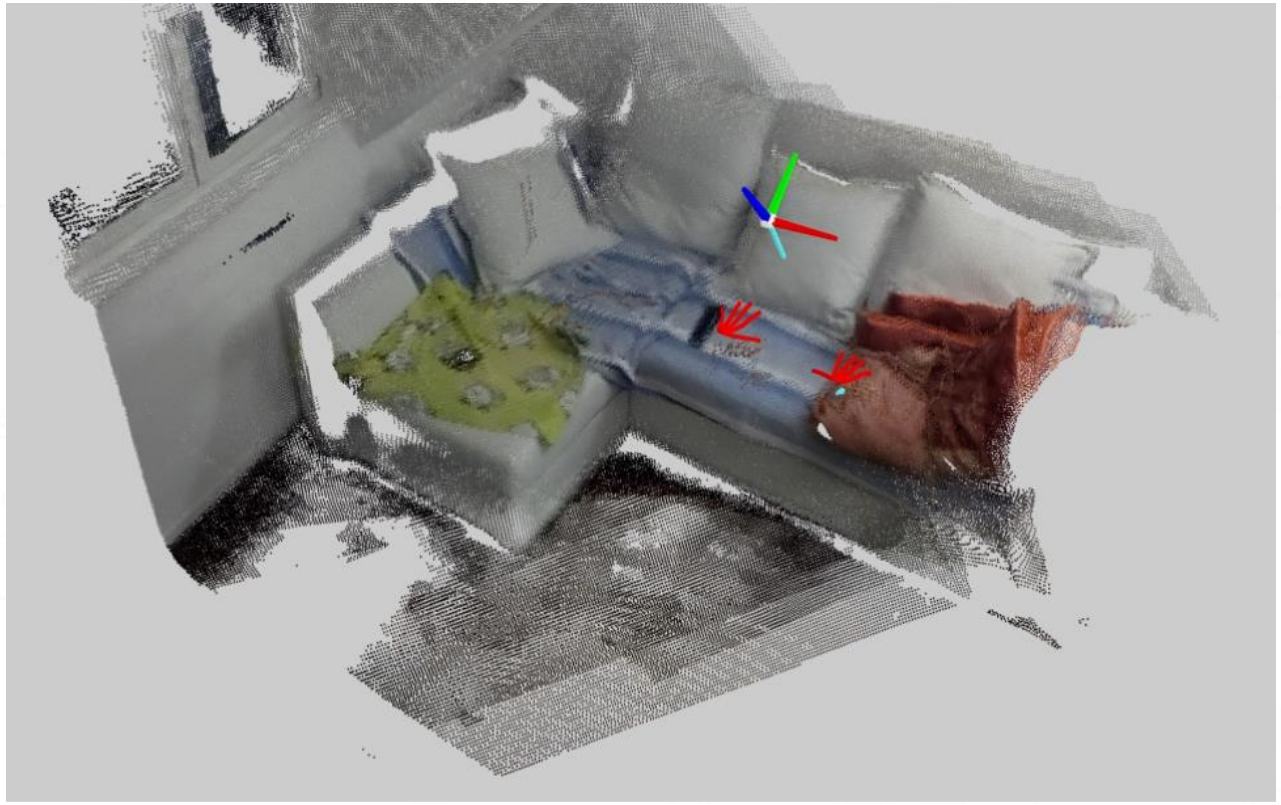
Hand, Eye, RGB Processing

- Projection of 3D left/right hand joints (and eye gaze) on 2D images
- Naïve hand – RGB frame matching based on closest timestamp…

Green: hand joints
Blue: eye gaze

All Streams Together

Red: hands
Blue: eye gaze
Axes: head 6dof



Cannon

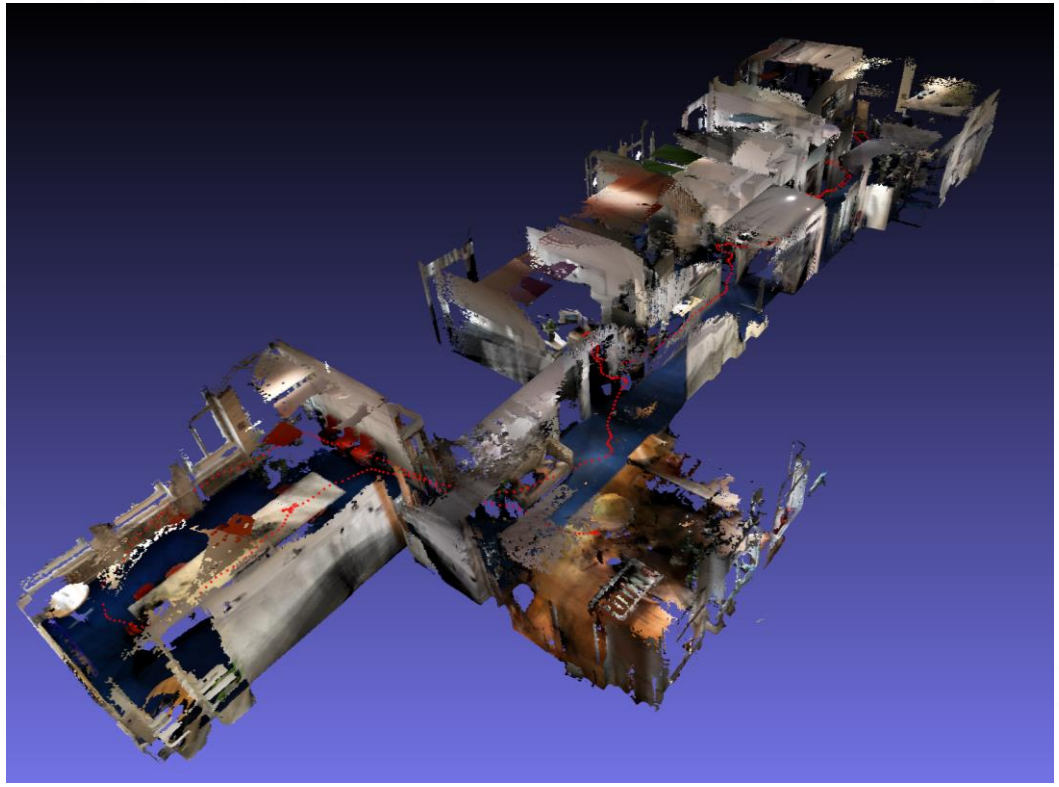
- Stream Recorder uses the Cannon library
- Cannon is a collection of wrappers and utility code for building native MR apps using C++, Direct3D, Windows Perception APIs
- Author: C. Meekhof (cmeekhof@microsoft.com)
- You can use it as-is for your apps!

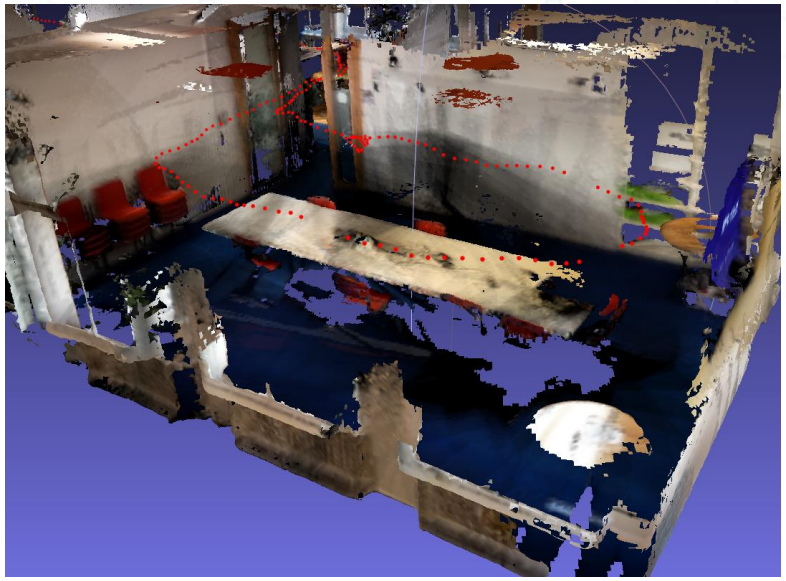
Build your CV Apps

- Research Mode and the HoloLens2ForCV repo are developed to make it easy to build computer vision applications
- Two examples here:
  - TSDF Volume Integration
  - SLAM

TSDF Volume Integration

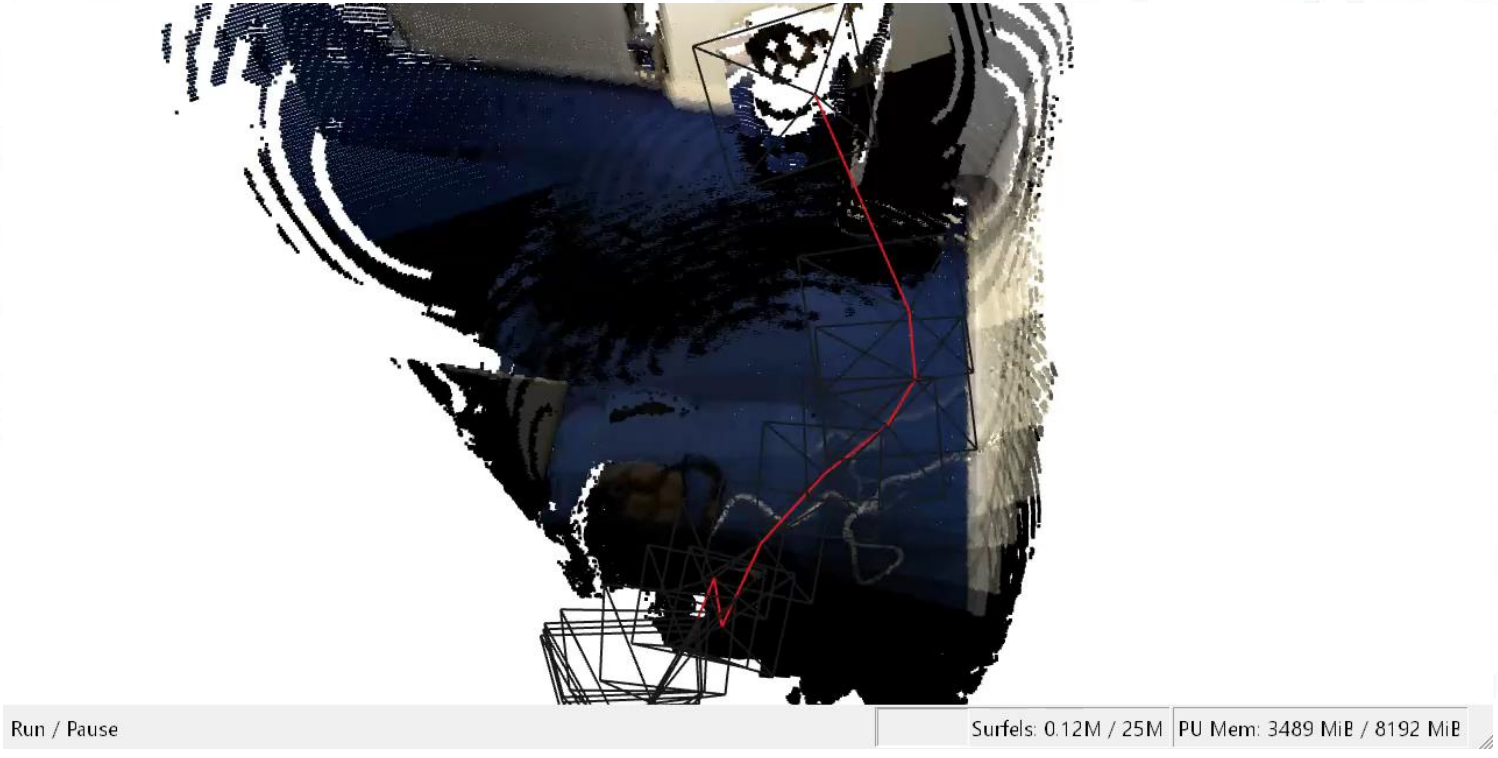
- Use Long Throw depth frames, RGB frames, head pose as input and an off-the-shelf library (open3d: http://www.open3d.org/docs/release/tutorial/Advanced/rgbd_integration.html#TSDF-volume-integration)





SLAM

- Use Long Throw depth frames and RGB frames as input and an off-the-shelf SLAM method
  (T. Schoeps, T. Sattler, M. Pollefeys. BAD SLAM: Bundle Adjusted Direct RGB-D SLAM. CVPR 2019)
- SLAM camera poses can be compared against HoloLens head poses



Summary

- HoloLens2ForCV repository: A collection of apps and tools in C++ / python
- A lot to build on top!
- Stream Recorder features in the pipeline:
  - QRcode-based coordinate system
  - Audio
- Feedback and contributions mostly welcome!