INSTYTUT STEROWANIA
I SYSTEMÓW INFORMATYCZNYCH

Code optimization techniques for On-Device Mixed Reality Applications

Andrzej Czajkowski, Ph.D.

Institute of Control & Computation Engineering
University of Zielona Góra

CUCEE lecture series

28.04.2026

Lecture agenda:

- Introduction
  - Extended realities
  - Applications
- Major challenges in XR
- Ongoing Research in XR
  - Scene understanding
  - Results and Summary

Introduction

About me

Andrzej Czajkowski, Ph.D.
e-mail: a.czajkowski@issi.uz.zgora.pl
WWW: http://staff.uz.zgora.pl/aczajkow/

Teaching Areas:

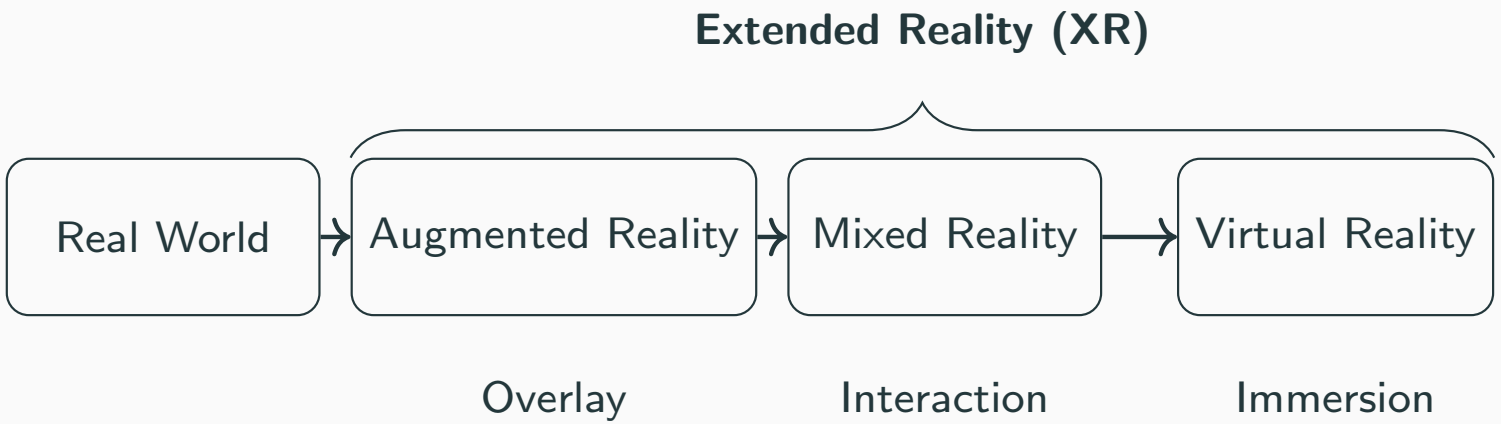
- programming techniques (Java and C#)
- 3D computer graphics (CGI)
- advanced graphics in advertising and games (Visualisations and game engines)

Research Areas:

- Artificial Intelligence
- Immersive Technologies
- 3D Scene Understanding and Spatial Interaction

Extended realities – Reality is merely an illusion

- Extended Reality (XR) includes AR, MR, and VR
- Mixed Reality (MR) lies between Augmented Reality and Virtual Reality
- It combines real-world perception with interactive virtual content
- Requires understanding and mapping of the physical environment

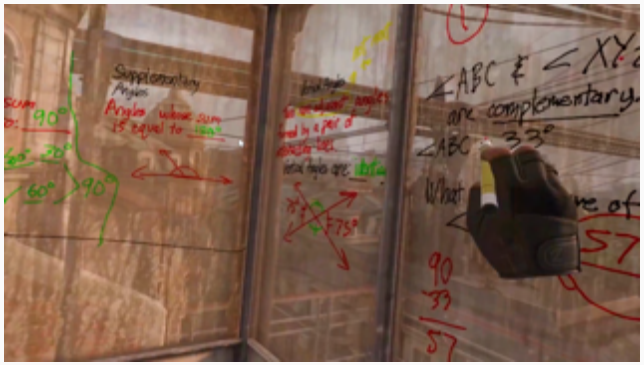


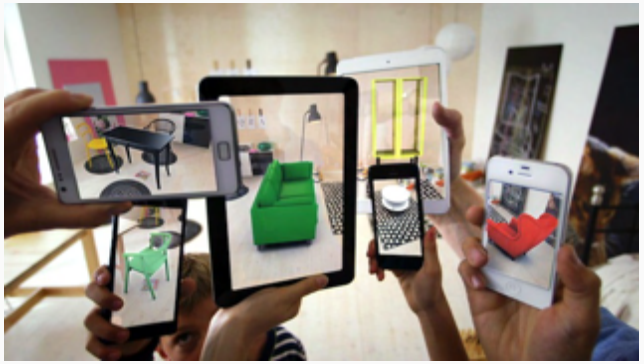
Increasing level of immersion and virtual content

Applications of Extended Reality (XR)

Media and Entertainment

- VR games (e.g., *Beat Saber*, *Half-Life: Alyx*)
- AR mobile games (e.g., *Pokémon GO*)
- Virtual film production (e.g., LED volume stages in *The Mandalorian*)
- AR-based advertising and retail experiences (e.g., IKEA Place)

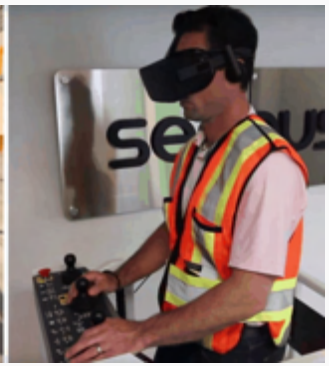


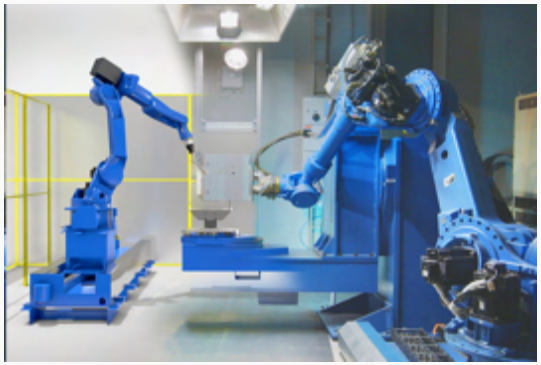


Applications of Extended Reality (XR)

Industrial Applications

- Worker training in VR simulations
- AR-based maintenance and repair instructions
- Digital twins of factories and machines



Applications of Extended Reality (XR)

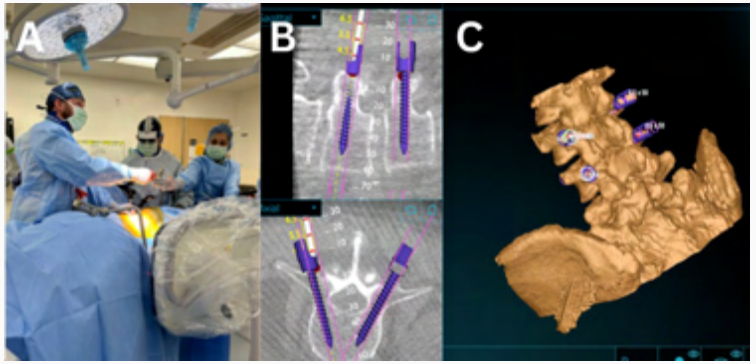
Automotive

- AR head-up displays (navigation and alerts)
- VR-based vehicle design and prototyping
- Driver assistance visualization systems

Applications of Extended Reality (XR)

Healthcare

- VR surgical training systems
- AR-assisted surgery with medical imaging overlay
- Rehabilitation and therapy in immersive environments



Major challenges in XR

Challenges: Spatial Anchoring

- Spatial anchoring allows virtual objects to remain fixed in the real world
- Digital content is aligned with physical space and persists over time

Key concepts:

- **World coordinate system** – shared reference frame for real and virtual objects
- **Tracking** – continuous estimation of device position and orientation
- **Persistence** – anchors remain stable across frames or sessions

Example:

- A virtual chair placed in a room stays in the same position as the user moves

Challenges:

- Drift and tracking errors
- Dynamic environments
- Real-time performance constraints

Challenges: Scene Understanding

- Scene understanding enables devices to interpret the physical environment
- It allows virtual content to interact meaningfully with the real world

Key tasks:

- **Plane detection** – identifying floors, walls, and surfaces
- **Object recognition** – detecting and classifying real-world objects
- **Scene reconstruction** – building 3D representations of the environment

Example:

- Placing a virtual object on a real table with correct alignment and occlusion

Challenges:

- Noisy and incomplete sensor data
- Real-time processing requirements
- Limited computational resources on-device

Challenges: Dynamic Environment Tracking

- XR systems must handle environments that change over time
- The system continuously updates its understanding of the scene

Key capabilities:

- **Change detection** – identifying new or moving objects
- **Object tracking** – following objects or people over time
- **Scene updates** – updating spatial representations dynamically

Examples:

- A person enters the scene and is tracked in real time
- Virtual objects adapt when furniture is moved
- Interaction with dynamic elements in the environment

Challenges:

- Distinguishing static vs dynamic elements
- Real-time processing of changing spatial data
- Robust tracking under occlusions and noise

Challenges: Depth Occlusion

- Depth perception allows the system to understand distances in the scene
- Occlusion ensures correct visual overlap between real and virtual objects

Depth Estimation

- Obtained from depth sensors or stereo vision
- Provides information about the 3D structure of the environment
- Used for accurate placement of virtual objects

Occlusion Handling

- Virtual objects should be hidden when behind real objects
- Requires a depth-aware representation of the scene
- Essential for realism and user immersion

Example

- A virtual object correctly disappears behind a real table or wall

Challenges

- Noisy or incomplete depth data
- High computational cost for real-time processing
- Limited sensor accuracy on mobile devices

Challenges: Hand and Eye Tracking

- Enables natural and intuitive interaction with virtual content
- Replaces or complements traditional input devices (controllers)

Hand Tracking

- Detects hand position and gestures using cameras and ML models
- Allows direct manipulation of virtual objects
- Example: grabbing, pointing, or scaling objects in MR

Eye Tracking

- Estimates gaze direction and focus point
- Enables gaze-based interaction and attention tracking
- Supports techniques like foveated rendering

Challenges

- Accuracy and latency requirements
- Occlusions and varying lighting conditions
- Real-time processing on resource-constrained devices

Ongoing Research in XR

ICAT-EGVE 2025

The presented research and obtained results have been published in:

- On-Device 3D Point Cloud Segmentation for Mixed Reality Applications, Andrzej Czajkowski , Marek Kowal , Rafael Greszczynski, in:ICAT-EGVE International Conference on Artificial Reality and Telexistence & Eurographics Symposium on Virtual Environments(3-5.12.2025): Proceedings, Karlskrona, Sweden, 2025,

Research Motivation

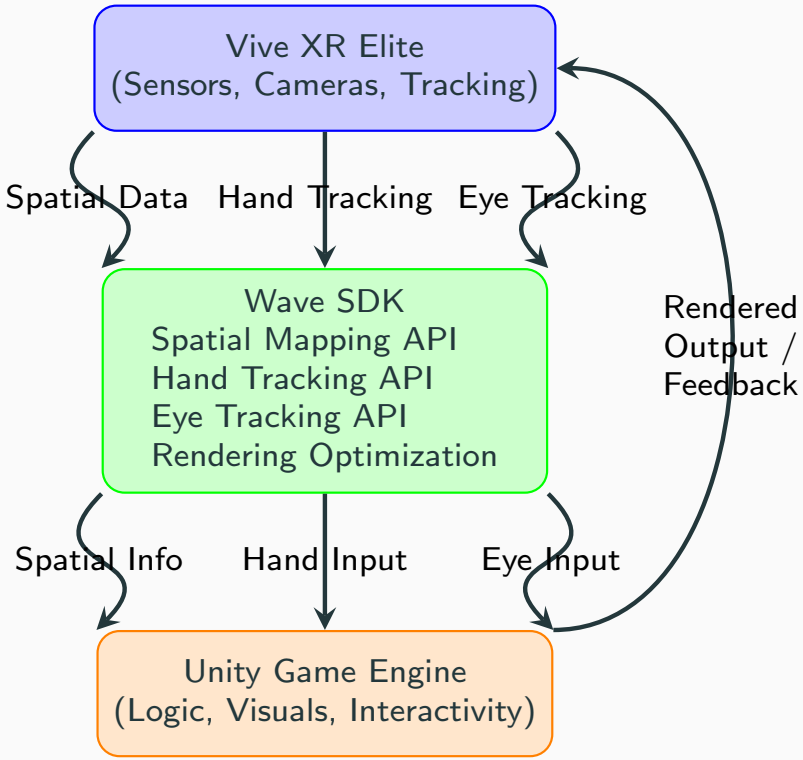
- Rapid development of **Mixed Reality (MR)** technologies requires accurate understanding of the environment
- Crucial for:
  - user immersion
  - safety in constrained environments
- Current solutions:
  - high-quality, but often operate offline or require significant computational resources
  - on-device solutions are not fully automated
- Problem:
  - limited access to spatial data in PCVR systems
  - hardware constraints of AR devices (CPU, memory, energy)
- Research objective:
  - development of **efficient on-device 3D segmentation**
  - maintaining a balance between **accuracy and performance**

Hardware used

- **HTC Vive XR Elite**
  - depth sensor + RGB cameras
  - 6DoF tracking
  - resolution: 1920 $\times$ 1920 per eye (3840 $\times$ 1920 total)
  - on-device processing (standalone)
- **SoC – Snapdragon XR2**
  - CPU: 8 cores (Kryo 585, up to 2.84 GHz)
  - GPU: Adreno 650
  - memory: 12 GB LPDDR4X
- Available data:
  - triangle mesh
  - 3D point cloud

Software Stack

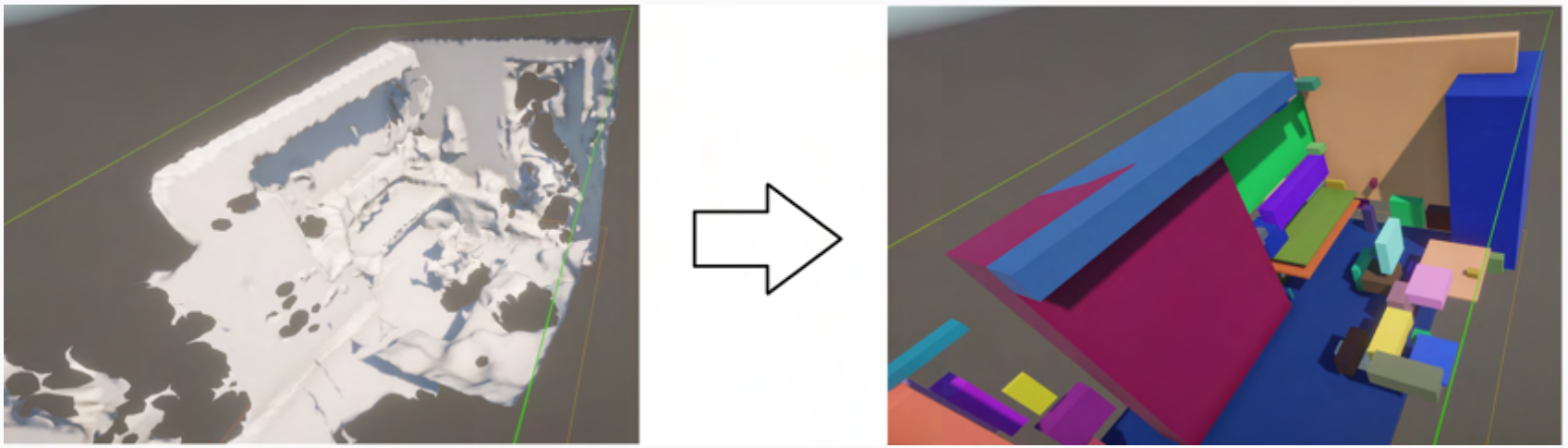
- **Unity**
  - 3D game engine for developing AR/VR applications
  - manages application logic, visualization, and interaction
  - integration with XR devices
- **Wave SDK (HTC)**
  - interface between hardware and application
  - provides access to:
    - spatial mapping
    - hand tracking and eye tracking



Project Objectives

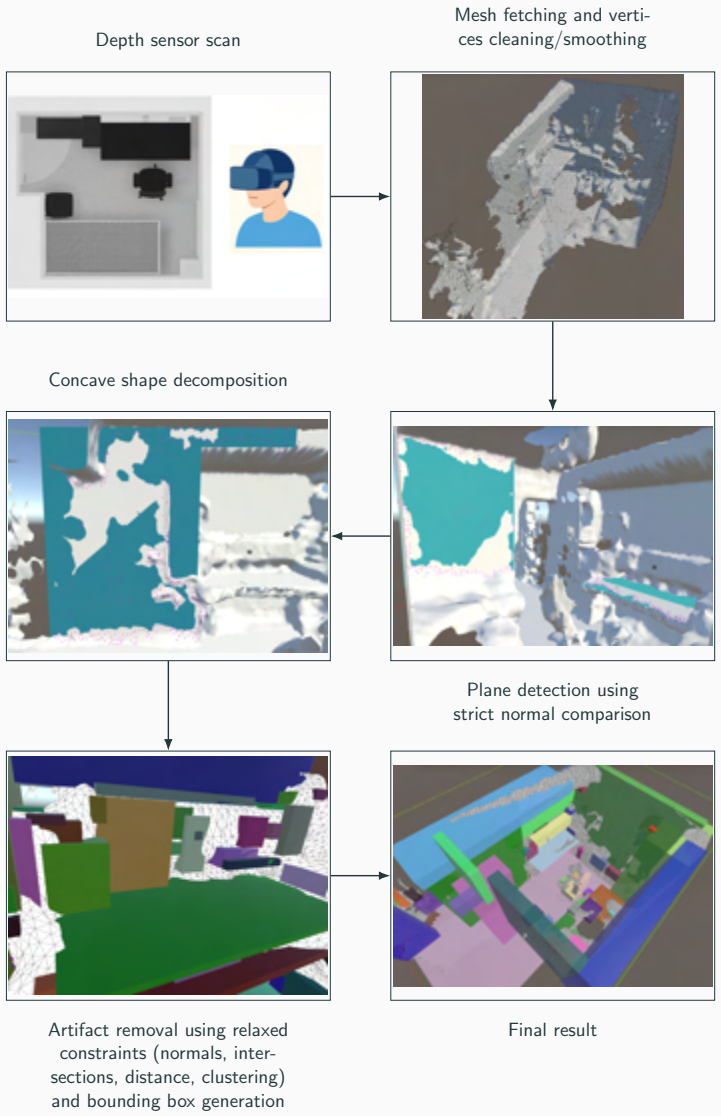
Main goals of the research:

- Detection of planar surfaces in a 3D mesh obtained from spatial mapping
- Application of geometric methods
  - analysis of unit normal vectors of adjacent surface polygons
- Detection of objects in the scene and their representation using **bounding boxes**
- Optimization enabling real-time execution directly on the device (**on-device processing**)



Procedure

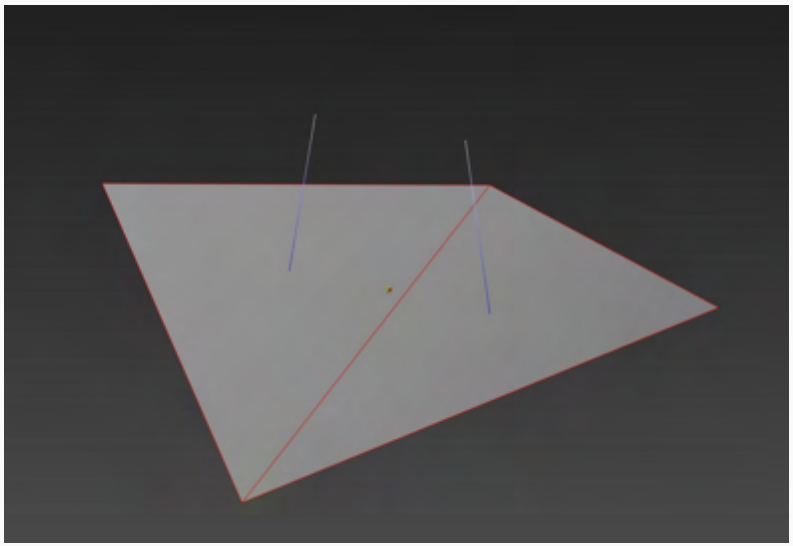
1. Room scanning using a depth sensor
2. Initial mesh cleaning and smoothing
3. Plane detection
4. Decomposition of L-shaped concavities
5. Artifact removal and projection of detected planes onto bounding boxes
6. Final room segmentation



The entire detection pipeline (excluding room scanning) is executed within a single frame, without distributing computations over time.

Estimating Deviation Between Polygons

1. Computation of a vector perpendicular to the polygon using the cross product of edge vertices
2. Normalization of vectors to obtain unit normal vectors
3. Computation of the dot product between normal vectors
4. Determination of coplanarity based on a fixed threshold:



threshold value	angle tolerance
0.999	$\sim 2.6^\circ$
0.99	$\sim 8.1^\circ$
0.95	$\sim 18.2^\circ$

Computational challenge

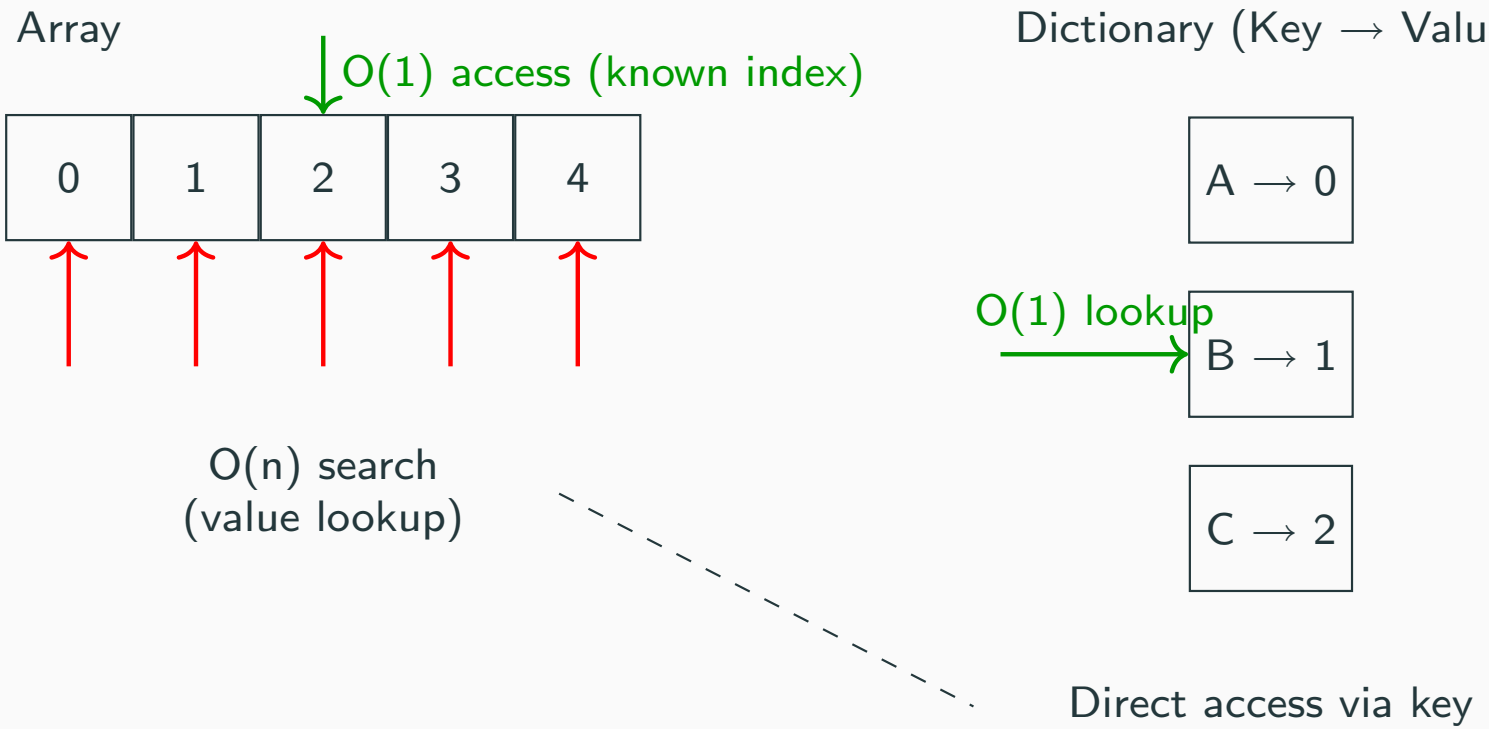
Depth scan details:

- Test environment: approximately 190k vertices / 65k polygons
- array data structures:
  - vertex array – `Vector3[] vertices`
  - triangle index array – `int[] triangles`
  - indexing scheme: the first triangle uses indices 0,1,2; the second 3,4,5; etc.
- Duplicated vertices
- Holes and scan artefacts

Applied optimisations:

- Removal of duplicated vertices and mesh reconstruction
- Mesh smoothing
- Using dictionaries instead of arrays for efficient lookup

Arrays vs Dictionaries: Access Performance



- Arrays: fast indexed access, inefficient search
- Dictionaries: efficient key-based lookup

Applied Optimizations

1. Simplification of the initial 3D mesh:
    - Crucial for fast lookups:
        - creation of a dictionary:

          `Dictionary<Vector3, int> vertDict`

          using vertices as keys to remove duplicates
        - remapping triangle indices:

          `triangles[i] = vertDict[vertices[triangles[i]]]`

          where `i` is the triangle index

Applied Optimizations

2. Plane smoothing:
   - Creation of three dictionaries where the key is the triangle index and the values are its vertices:
     ```
     Dictionary<int, Vector3> vertOne;
     Dictionary<int, Vector3> vertTwo;
     Dictionary<int, Vector3> vertThree;
     ```
   - Creation of key collections using `Lookup`:
     ```
     var lookupOne = vertOne.ToLookup(a => a.Value);
     var lookupTwo = vertTwo.ToLookup(a => a.Value);
     var lookupThree = vertThree.ToLookup(a => a.Value);
     ```
   - Lookups enable reverse mapping from vertices to triangle indices

   These structures allow efficient lookup of vertices for neighboring polygons and enable the smoothing process (removing redundant triangles/vertices and mesh artifacts).

Applied Optimizations

3. Plane detection:
    - Creation of a list of vertex sets for detected planes to avoid duplicate vertices:

      ```
      List<HashSet<Vector3>> detectedPlaneList
      ```

    - Reuse of dictionary and Lookup for efficient search of neighboring triangles:

      ```
      Dictionary<int, Vector3> vertsInTris
      var lookup = vertsInTris.ToLookup(a => a.Value);
      ```

    - Use of a set structure to track processed triangles:

      ```
      HashSet<int> checkedTriangles
      ```

The proposed algorithm

Input: Vertices array V, triangle index array T
Output: List of detected planes as sets of vertices
Initialize *detectedPlaneList* $\leftarrow \emptyset$
Initialize dictionary *vertsInTris*
for $i \leftarrow 0$ **to** $|T| - 1$ **do**
    vertsInTris$[i] \leftarrow V[T[i]]$

lookup $\leftarrow$ group *vertsInTris* by vertex value
Initialize *checkedTriangles* $\leftarrow \emptyset$
for $i \leftarrow 0$ **to** $|T| - 1$ *step* 3 **do**
    if $i \notin$ *checkedTriangles* **then**
        $p_1, p_2, p_3 \leftarrow V[T[i]], V[T[i+1]], V[T[i+2]]$
        $n_1 \leftarrow normalize(cross(p_2 - p_1, p_3 - p_1))$
        trianglesToCheck $\leftarrow \{i\}$
        vertsInPlane $\leftarrow \emptyset$
        while *trianglesToCheck* $\neq \emptyset$ **do**
            Remove an element t from *trianglesToCheck*
            Add t to *checkedTriangles*
            $q_1, q_2, q_3 \leftarrow V[T[t]], V[T[t+1]], V[T[t+2]]$
            $n_2 \leftarrow normalize(cross(q_2 - q_1, q_3 - q_1))$
            if $dot(n_1, n_2) \geqslant$ *thresholdValue* **then**
                Add q_1, q_2, q_3 to *vertsInPlane*
                foreach *point p in* $\{q_1, q_2, q_3\}$ **do**
                    foreach *triangle r sharing p from lookup* **do**
                        rIdx $\leftarrow r - (r \bmod 3)$
                        if *rIdx* $\notin$ *checkedTriangles* **then**
                            Add *rIdx* to *trianglesToCheck*
        Add *vertsInPlane* to *detectedPlaneList*

return *detectedPlaneList*

The room scan and C# code are available at: `https://github.com/aczajkowskiuz/3dpointcloudsegmentation`

Plane and Bounding Box Construction

Goal: obtaining a compact geometric representation of a detected planar region

1. **Centroid computation**
   For each detected plane, the geometric center of its points is calculated.

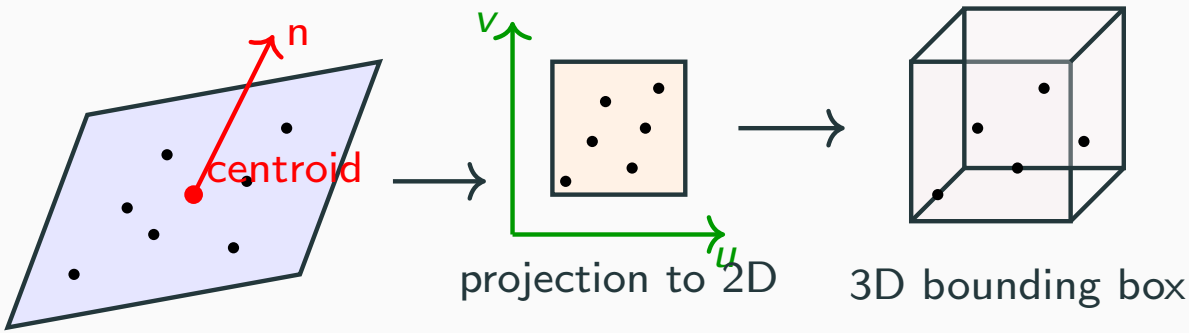
2. **Normal estimation**
   The normal vector n describing the approximate surface orientation is computed using the covariance matrix.

3. **Local coordinate system construction**
   A local coordinate system (u, v, n) is defined.

4. **Projection to 2D**
   Points are projected onto the (u, v) plane.

5. **Region boundary estimation**
   The values $u_{\min}$, $u_{\max}$, $v_{\min}$, $v_{\max}$ are computed to determine the region corners.

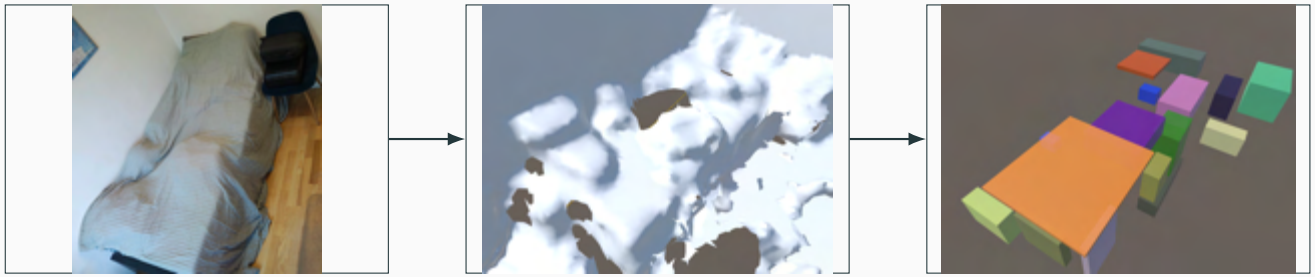
6. **Back-projection to 3D**
   The corners are projected back into 3D space, forming a rectangular representation of the region that preserves its orientation.



Results for Example Objects

(a) Physical Object (b) Scan with depth sensor (c) Segmentation with bounding boxes

(a) Physical Object (b) Scan with depth sensor (c) Segmentation with bounding boxes



(a) Physical Object (b) Scan with depth sensor (c) Segmentation with bounding boxes

Summary

- The lecture covered key aspects of **on-device Mixed Reality (MR)** systems

Key topics presented:

- Extended Reality (XR) ecosystem and applications
- Scene understanding in real-time 3D environments
- Spatial data processing (meshes and point clouds)
- Plane detection and geometric reasoning in 3D space
- Object representation using bounding boxes

Core challenge:

- Processing complex spatial data under strict **computational and energy constraints**
- Ensuring real-time performance on standalone XR devices

Optimization strategies:

- Mesh simplification and vertex deduplication
- Efficient data structures (dictionaries, lookups, hash sets)
- Reduction of redundant geometric computations
- Single-frame processing pipeline design

Conclusion:

- Efficient spatial processing enables **real-time, immersive, and safe XR experiences**
- Balancing accuracy and performance is critical for on-device MR systems

Thank you for your attention!!!

*

Q&A

a.czajkowski@issi.uz.zgora.pl