Studium możliwości wykorzystania silnika Unity i technologii VR w stworzeniu symulacji medycznej do zakładania szwów.

Spis treści

Wstęp

Wyjaśnienie tytułu pracy i uzasadnienie aktualność tematu.
Wyjaśnienie terminów: symulacja medyczna i zakładanie szwów.
Uzasadnienie wyboru technologii VR i silnika Unity.
Uzasadnić aktualność tematu: wyniki badań ukazujące korzyści płynące ze stosowania szkoleń medycznych w technologii VR zestawione ze statystykami wykorzystywania tej technologii do szkolenia personelu medycznego; przewidywania rynkowe ukazujące prognozowany wzrost rynku produkcji w technologii VR.

Cel pracy
Studium możliwości wykorzystania silnika Unity i technologii VR w tworzeniu symulacji medycznej zakładania szwów.

Tezy
Wykorzystanie systemów i mechanizmów silnika Unity do wytworzenia obiektów potrzebnych do stworzenia symulacji medycznej zakładnia szwów w technologii VR.

Przedstawienie i uzasadnienie rozdziałów występujących w pracy
- Omówienie praktyk medycznych stosowanych przy zszywaniu ran. – Rozdział traktujący o wiedzy medycznej, podłożu merytorycznym aplikacji.
- Tworzenie aplikacji w Unity – Omówienie aplikacji z punktu widzenia środowiska w jakim jest wytwarzana. Elementy kodu, wykorzystane IDE, zastosowane biblioteki, omówienie state machine, algorytmów.
- Technologia VR – Konfiguracja i eksploatacja aplikacji na urządzeniu do VR.

Rozdziały i podrozdziały

Rozdział I. Omówienie praktyk medycznych stosowanych przy zszywaniu ran.
1.1. Wprowadzenie i uzasadnienie wyboru.
1.2. Metody i przyrządy stosowane w medycynie.
1.3. Uzasadnienie wybranych metod

Rozdział II. Tworzenie aplikacji w Unity – Uzasadnienie wyboru
2.1. Scenariusz aplikacji.
2.2. Przedstawienie obiektów reprezentujących narzędzia i powierzchnie(reprezentacja skóry) wykorzystane do przeprowadzenia symulacji.
2.3. Omówienie mechanizmów i systemów odpowiedzialnych za zachowania odpowiadające obiektom.

Rozdział III. Technologia VR – Omówienie wybranego sprzętu.
3.1. Przegląd okularów VR pod kątem ich przydatności w przeprowadzaniu symulacji medycznych.
3.2. Etap konfiguracji: konfiguracja wybranego sprzętu w technologii VR z silnikiem Unity.
3.3. Ocena sprawności i celów założonej aplikacji.

Rozdział IV. Osiągnięte funkcjonalności.
4.1. Aspekt użytkownika i wpływ na użytkownika – skrócenie czasu nauki dla studentów i poczatkujących

Rozdział V. Zastosowania dla innych branż.
5.1. Jakie inne zastosowania w przemyśle i inne branże: elektronika – łączenie elementów, nauka robienia węzłów, medycyna, branża szkoleniowa

Zakończenie

Ocena powodzenia bądź niepowodzenia przeprowadzonej próby stworzenia aplikacji.
Niepowodzenie: Ewentualne uzasadnienia niepowodzenia i propozycje innych rozwiązań.
Powodzenie: Wskazanie możliwości rozwoju i przyszłych zastosowań opracowanych funkcjonalności.

Bibliografia
Nazwisko Pełne Imię/ Nazwisko Imię skrót – konsekwentnie

Rozdział III. Technologia VR – Omówienie wybranego sprzętu.
3.1. Przegląd okularów VR pod kątem ich przydatności w przeprowadzaniu symulacji medycznych.

W tym rozdziale chciałabym przedstawić obecnie dostępne na rynku Google VR potrzebne do przeprowadzania symulacji medycznych w wirtualnej rzeczywistości. Skupię się na ich zróżnicowaniu pod kątem wymagań sprzętowych niezbędnych do tworzenia aplikacji, rozwiązań systemowych wpływających na imersje oraz aspekty mogące wpływać na poprawę poprawności wykonywanych zadań.

Na rynku możemy wymienić czterech głównych dostawców Googli VR i ich najnowsze modele, m.in.

- Apple – Apple Vision Pro; przeznaczone do rozwijania aplikacji na sprzęcie z systemem operacyjnych macOS i pracujące na swoim wewnętrznym systemie operacyjnym visionOS. Rozdzielczość wynosi 23 miliony pikseli, a częstotliwość odświeżania obrazu dostępna jest w wariantach 90 Hz, 96 Hz i 100 Hz. Pole widzenia wynosi od 100 do 110 stopni. Przekładają się na wysoką rozdzielczość, płynność obrazu i naturalne postrzeganie przestrzeni. Urządzenie przetwarza informację pobrane z sensorów obserwujących użytkownika lub z przycisków i dedykowanych do tego przekaźników uruchamianych świadomie przez niego. Oto przykłady technologii zastosowanych w opisywanym urządzeniu:

(1)Digital Crown – pokrętło służące do ustalania poziomu głośności, modyfikowania poziomu imersji, centrowania wirtualnego środowiska i poruszania się po menu. System reaguje również na wydarzenia takie jak poruszanie się zbyt szybko, podchodzenie zbyt blisko fizycznych obiektów, przekraczanie granicy zdefiniowanej do aktywacji środowiska VR.

(2)Eyes – sterowanie wzrokiem ułatwiające nawigowanie po interfejsie aplikacji. Elementy interfejsu na które użytkownik kieruje swój wzrok zostają podświetlone, jest to tzw. hover effect który zwraca użytkownikowi wizualną odpowiedź gotowości do interakcji z wybrany elementem używając do tego gestów pośrednich. W niektórych przypadkach samo skierowanie wzroku może aktywować element.

(3)Focus effect – wyostrzanie ostrości obiektu który został wybrany przez użytkownika korzystającego z urządzenia zewnętrznego, na przykład kontroler lub klawiatura. Obiekt z zaprogramowanym efektem focusu może zostać również zostać powiększony, przeniesiony na pierwszy plan, oświetlony bądź wzbogacony o animację.

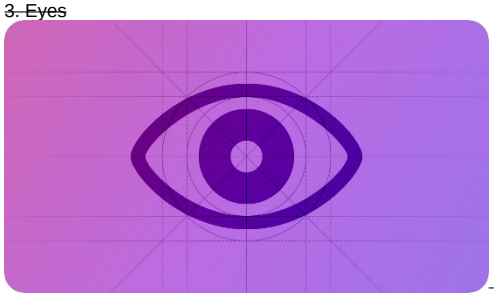
(4)Fizyczne kontrolery – umożliwiają wprowadzanie danych wejściowych zawsze, niezależnie od ich położenia i widoczności. W przypadku środowiska wirtualnego stanowią alternatywę dla hand trackingu i według dobrych praktyk tworzenia aplikacji powinny być zawsze uwzględniane jako jeden ze sposobów sterowania.

(5)

3.2. Etap konfiguracji: konfiguracja wybranego sprzętu w technologii VR z silnikiem Unity.
3.3. Ocena sprawności i celów założonej aplikacji.

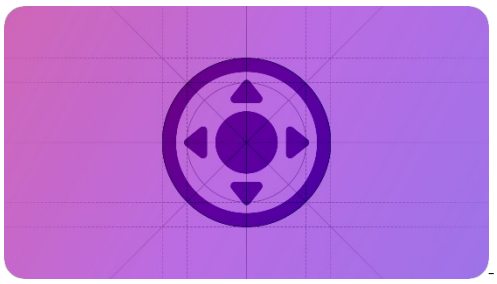
~~**Apple Vision Pro**~~
~~https://developer.apple.com/design/human-interface-guidelines/touch-bar~~
~~How to setup Project~~
~~Equipment: App development: MAC/Windows; Tests & debugging: MAC or Windows with Virtual Machine and MAC on it or on Windows through cloud services with MAC on it.~~
~~XR.sdk.visonOS - instaled in Unity~~
~~XR plug-manager in Unity for visionOS~~
~~Packages: com.unity.polyspatial, com.unity.polyspatial.visionos, com.unity.polyspatial.xr~~

~~3. Eyes~~



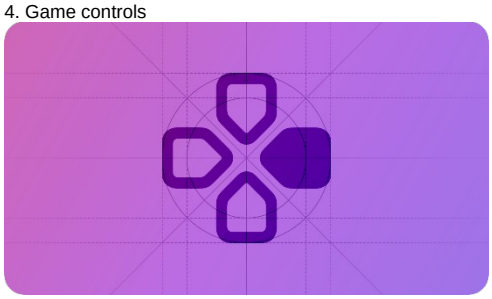
~~When people look at an interactive element, visionOS highlights it as a confirmation of their choice. **The hover effect** provides visual feedback, indicating that it is possible to use an indirect gesture to interact with the element. Indirect gestures involve looking at something and manipulating it with hand gestures. In some cases, using a gesture may not be necessary, as certain objects can reveal their content on their own.~~

~~3. Focus effect – provides navigation through elements in the FOV by using inputs like remote, controller, keyboard etc.~~



~~It's not related to hover effect.~~

4. Game controls



~~Input that every player can use and which is familiar to most of the users.~~

5. Gestures & hand tracking

Data from hand tracking: anchor, skeleton, live data about the position of a person's hands and hand joints, location and orientation of a hand in world space, joints on a hand, state of tracking.

Gesture – physical motion that is affecting an object in an app. We can differentiate gestures on a touchscreen, in the air, on a trackpad, remote or game controller that includes a touch surface. **Direct gestures** serve to physically touch an interactive object. Direct interaction would be using visionOS keyboard, or virtual mouse.
Advice: people tent to get tired ones they need to use direct gestures all the time. So, direct gestures are best for infrequent use.
Object should provide interaction through direct and indirect gestures.

Standard direct gestures

Direct gesture	**Common use**
Touch	Directly select or activate an object.
Touch and hold	Open a contextual menu.
Touch and drag	Move an object to a new location.
Double touch	Preview an object or file; select a word in an editing context.
Swipe	Reveal actions and controls; dismiss views; scroll.
With two hands, pinch and drag together or apart	Zoom in or out.
With two hands, pinch and drag in a circular motion	Rotate an object.

Standard gestures (indirect)

Gesture	**Supported in**	**Common action**
Tap	iOS, iPadOS, macOS, tvOS, **visionOS**, watchOS	Activate a control; select an item.
Swipe	iOS, iPadOS, macOS, tvOS**, visionOS**, watchOS	Reveal actions and controls; dismiss views; scroll.
Drag	iOS, iPadOS, macOS, tvOS, **visionOS**, watchOS	Move a UI element.
Touch (or pinch) and hold	iOS, iPadOS, tvOS**, visionOS**, watchOS	Reveal additional controls or functionality.
Double tap	iOS, iPadOS, macOS, tvOS, **visionOS**, watchOS	Zoom in; zoom out if already zoomed in; perform a primary action on Apple Watch Series

		9 and Apple Watch Ultra 2.
Zoom	iOS, iPadOS, macOS, tvOS, **visionOS**	Zoom a view; magnify content.
Rotate	iOS, iPadOS, macOS, tvOS, **visionOS**	Rotate a selected item.

Custom gestures - if there is no standard gesture that responds to certain behaviour that we want to perform in an app it's possible to create custom gesture.
Basic requirements:

- App running in Full Space
- Request for people's permission to access information about their hands

Advice:

- Avoid custom gesture just for specific hand
- Diminish complexity of custom gestures, it should require usage of both hands and multiple fingers
- Consider the time and number of repetitions of gestures, not to make it too tiring

System overlay – layer concerning basic system short cuts for user like Home and Control Centre.

Deferring System overlay behaviour – depending on background or being in Full Space System overlay can be modified and require gestures to reveal features.

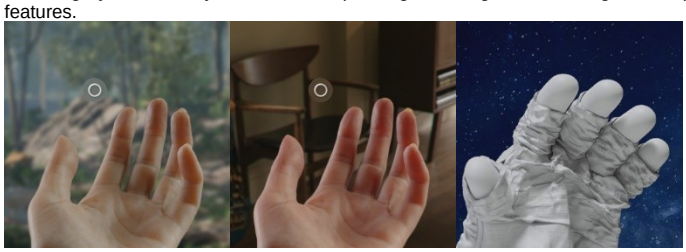
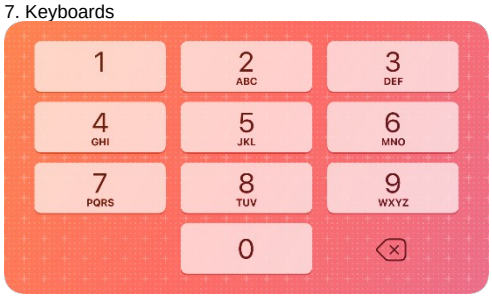


Image a) Shared Space, b) Full Space, c) deferred behavior in a Full Space

6. Gyroscope and accelerometer
On-device gyroscopes and accelerometer supply data about a device's movement in physical world within raw or processed values. Processed values eliminate forms of bias that might affect usage of the data. For example, accelerometer raw data will consider gravity, while processed data only user's movement.

7. Keyboards



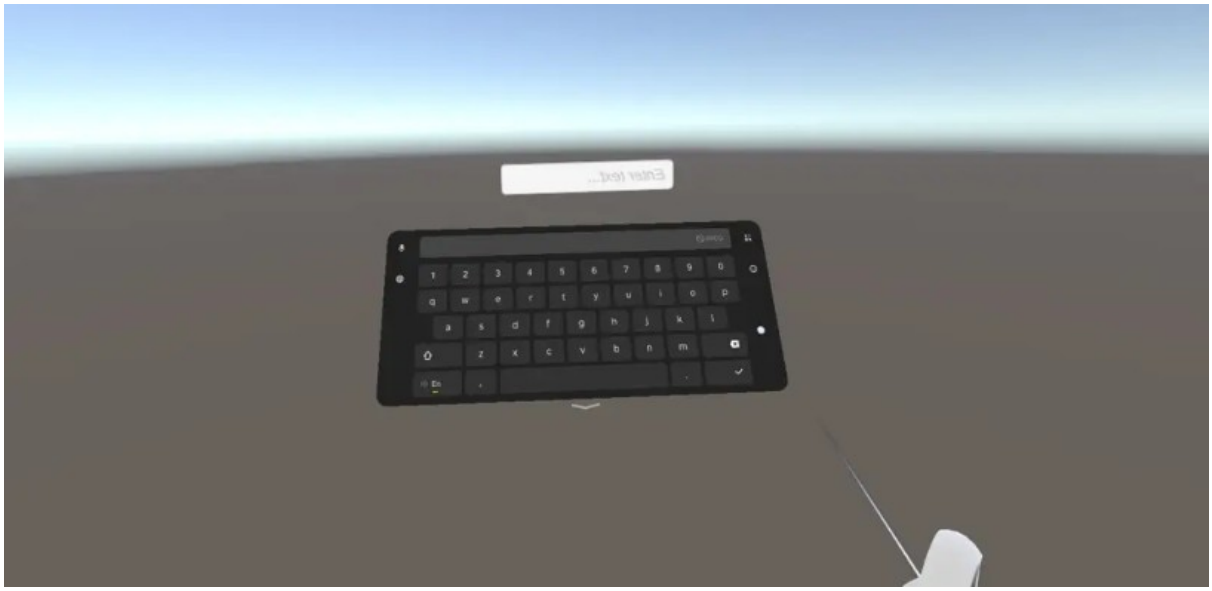
Virtual keyboards come with variety of types accordingly to the needs of a user. For example, to enter an email keyboard can contain straight buttons like ".com"
Should behave like physical keyboard in the matter of sound. System should support direct and indirect gestures and should appear in separate window witch user can move wherever one wants.

Pico
Controller & HMD input mapping | PICO Developer (picoxr.com)
Equipment: App development: MAC/Windows; Tests & debugging: Windows or MAC through Android Studio

Technicals: 72Hz/90Hz, 105°

1. System Keyboard
2. Eye Tracking
3. Hand Tracking
4. Face Tracking
5. Body Tracking
6. Object Tracking
7. Field of recognition

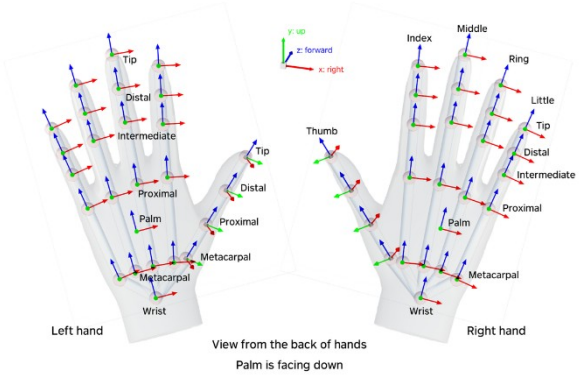
1. System keyboard



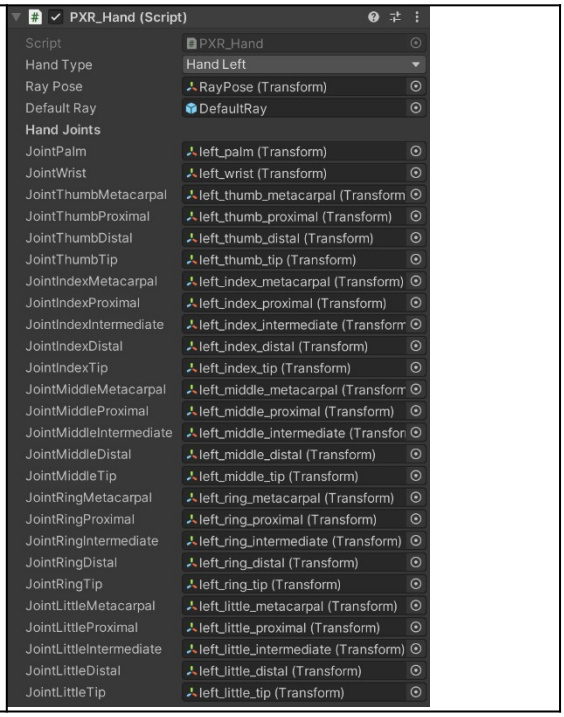
Virtual keyboard enabled when typing input in app.

2. Eye Tracking
Sensor technology that is tracking users gaze in real time. Eye movement is converted into data streams containing pupil distance, gaze vector, gaze point, openness, blink as a device's input. The device then decodes the data to display what the eyes are capturing in real time.
Optimalisation: Eye tracking can be dynamically started/stopped – optimized solution

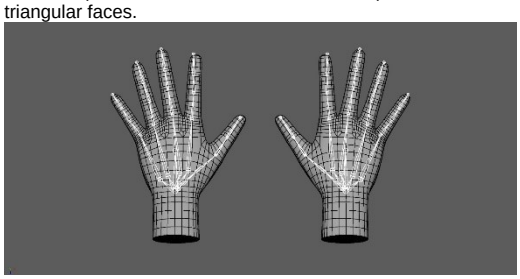
3. Hand Tracking
PICO SDK's hand tracking feature follows the hand joint conventions outlined by OpenXR and supports the 26 hand joints, using it as the primary input source. Some hand poses can trigger different events like, pinch enables the ray pointer which can be used to click and drag objects.
Hand tracking data
Call GetSettingState to get the status of hand tracking for your app. The request returns a bool value, true indicates "enabled" and false indicates "disabled".
Call GetActiveInputDevice to get the current active input device, which can be the HMD, controllers, or hands.
Call GetAimState to get the status of hand interaction. The response returns the pose of the ray, whether the ray is displayed and touched.
Call GetJointLocations to get the overall level (high/low) of confidence for hand tracking as well as hand pose data which includes the number of joints tracked, the scale of hand, the locations and orientations of joints, the radius of joints.



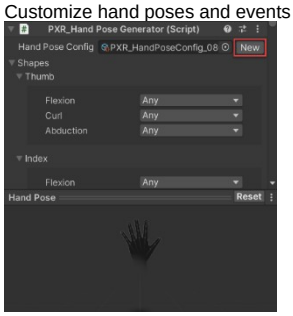
```
// Provided by XR_EXT_hand_tracking
typedef enum XrHandJointEXT {
  XR_HAND_JOINT_PALM_EXT = 0,
  XR_HAND_JOINT_WRIST_EXT = 1,
  XR_HAND_JOINT_THUMB_METACARPAL_EXT = 2,
  XR_HAND_JOINT_THUMB_PROXIMAL_EXT = 3,
  XR_HAND_JOINT_THUMB_DISTAL_EXT = 4,
  XR_HAND_JOINT_THUMB_TIP_EXT = 5,
  XR_HAND_JOINT_INDEX_METACARPAL_EXT = 6,
  XR_HAND_JOINT_INDEX_PROXIMAL_EXT = 7,
  XR_HAND_JOINT_INDEX_INTERMEDIATE_EXT = 8,
  XR_HAND_JOINT_INDEX_DISTAL_EXT = 9,
  XR_HAND_JOINT_INDEX_TIP_EXT = 10,
  XR_HAND_JOINT_MIDDLE_METACARPAL_EXT = 11,
  XR_HAND_JOINT_MIDDLE_PROXIMAL_EXT = 12,
  XR_HAND_JOINT_MIDDLE_INTERMEDIATE_EXT = 13,
  XR_HAND_JOINT_MIDDLE_DISTAL_EXT = 14,
  XR_HAND_JOINT_MIDDLE_TIP_EXT = 15,
  XR_HAND_JOINT_RING_METACARPAL_EXT = 16,
  XR_HAND_JOINT_RING_PROXIMAL_EXT = 17,
  XR_HAND_JOINT_RING_INTERMEDIATE_EXT = 18,
  XR_HAND_JOINT_RING_DISTAL_EXT = 19,
  XR_HAND_JOINT_RING_TIP_EXT = 20,
  XR_HAND_JOINT_LITTLE_METACARPAL_EXT = 21,
  XR_HAND_JOINT_LITTLE_PROXIMAL_EXT = 22,
  XR_HAND_JOINT_LITTLE_INTERMEDIATE_EXT = 23,
  XR_HAND_JOINT_LITTLE_DISTAL_EXT = 24,
  XR_HAND_JOINT_LITTLE_TIP_EXT = 25,
  XR_HAND_JOINT_MAX_ENUM_EXT = 0x7FFFFFFF
} XrHandJointEXT;
```



PICO hand model prefabs
The SDK provides two standard hand model prefabs: HandLeft and HandRight. Each model consists of 1209 vertices, 1198 quadrilateral faces, and 2414 triangular faces.



Customize hand poses and events



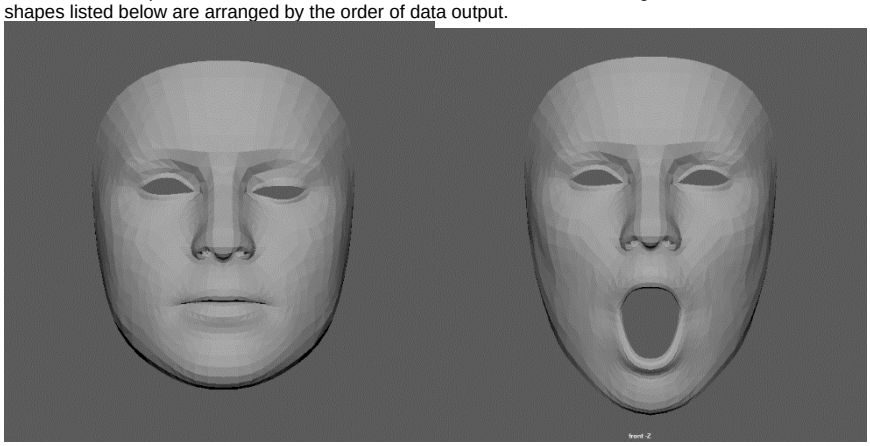
Hand gestures customisation – gives possibility to create margins for smooth transitions and smooth activation avoiding jittering between two states. It provides Hold duration with state that maintains before switching. Also, it provides the shapes of the fingers like flexion, curl and abduction.
Hand pose script – it serves for tracking the hand events: Start(), Update(), End(). The track type is: Any/Left/Right Hand.

Self-adaptive hand models

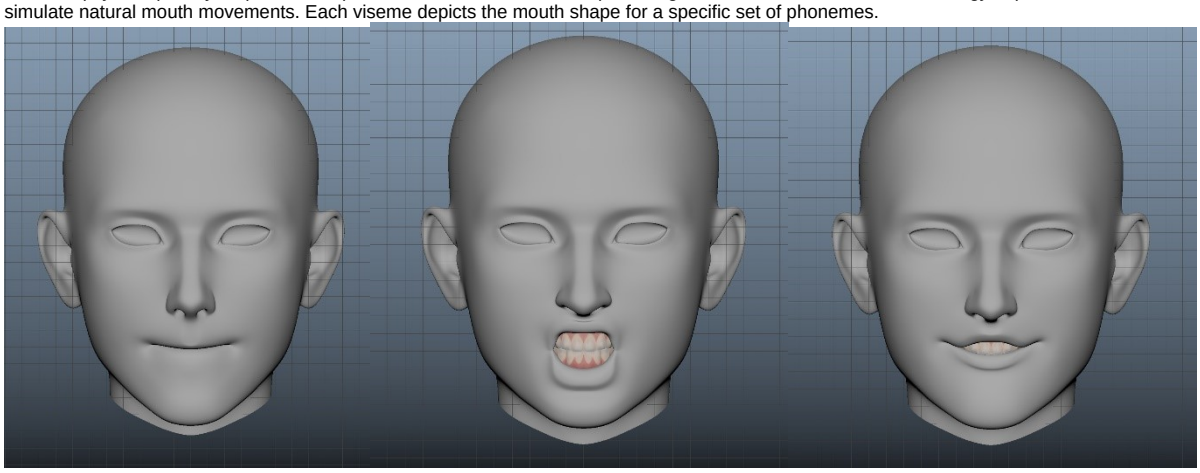
PICO's official hand models are self-adaptive, where its size can dynamically change based on the change in the size of the user's real hands. – with custom models GetHandScale needs to be called
https://developer.picoxr.com/document/unity/about-the-pxr-hand-pose-generator-script/https://developer.picoxr.com/document/unity/about-the-pxr-hand-pose-generator-script/

4. Face tracking
PICO Neo3 and PICO 4 series - Lipsync Only
PICO 4 Pro and PICO 4 Enterprise – Face Only / Hybrid (Viseme) / Hybrid (Blendshape) - API converts data into 52 blendshapes and 20 visemes, which can be retrieved in array od length 72 (52+20)
Dynamically start/stop face tracking by calling Call StartFaceTracking and Call StopFaceTracking.

Blend shapes
The 52 blend shapes describe the movements of facial features. The following table describes the BlendShapeIndex enums numbered 0 to 51. The blend shapes listed below are arranged by the order of data output.



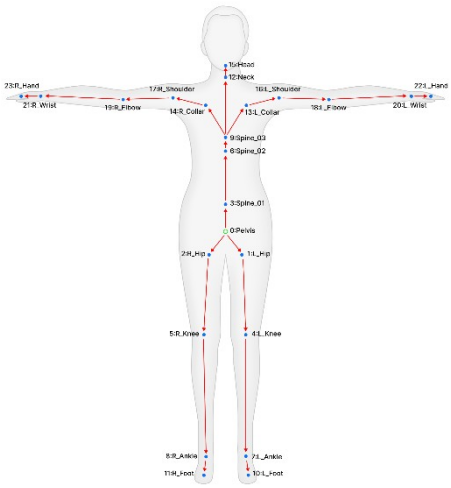
Visemes
PICO's Lipsync capability maps human speech to a set of 20 mouth shapes using visemes. Visemes are visual analogy of phonemes that are used to simulate natural mouth movements. Each viseme depicts the mouth shape for a specific set of phonemes.



5. Body tracking
Body trackers - Data that we can retrieve from body trackers: position, orientation data.
Stepping recognition – maximum 4 steps per second, small amplitude steps is also unadvised, as well as half-toe stepping.
Can be use as object tracking. Maximum trackers usage – 3

PICO SDK's Body Tracking feature supports tracking 24 human body joints as shown below.

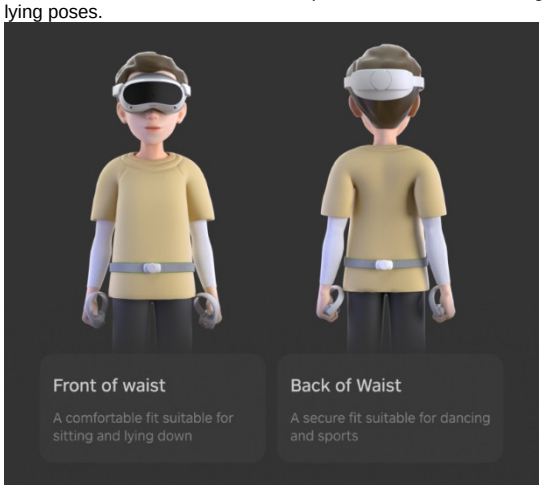
```
public enum BodyTrackerRole
 {
    Pelvis = 0,
    LEFT_HIP = 1,
    RIGHT_HIP = 2,
    SPINE1 = 3,
    LEFT_KNEE = 4,
    RIGHT_KNEE = 5,
    SPINE2 = 6,
    LEFT_ANKLE = 7,
    RIGHT_ANKLE = 8,
    SPINE3 = 9,
    LEFT_FOOT = 10,
    RIGHT_FOOT = 11,
    NECK = 12,
    LEFT_COLLAR = 13,
    RIGHT_COLLAR = 14,
    HEAD = 15,
    LEFT_SHOULDER = 16,
    RIGHT_SHOULDER = 17,
    LEFT_ELBOW = 18,
    RIGHT_ELBOW = 19,
    LEFT_WRIST = 20,
    RIGHT_WRIST = 21,
    LEFT_HAND = 22,
    RIGHT_HAND = 23
 }
```



Below are the descriptions of related concepts:

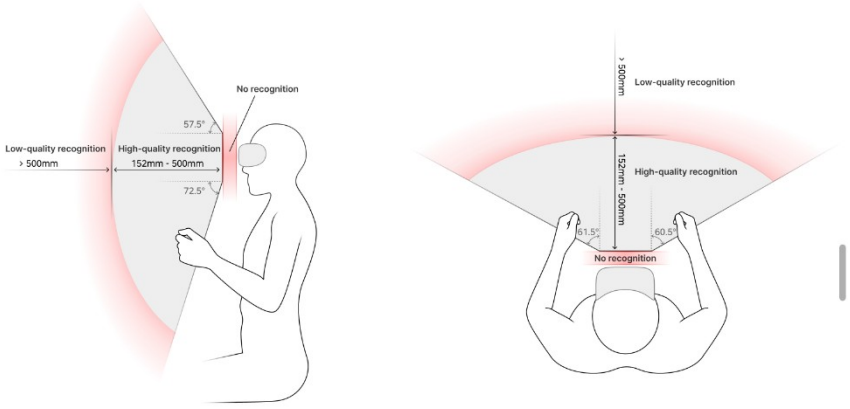
Concept	Description
Coordinate	Body joint data uses the same global coordinate system as the HMD data
Root joint	0 (Pelvis)
Parent/child joint	Joints numbered from 1 to 23. Parent joints are located near the root joint, while the child joints are located near the end of the limbs.
Bone	A bone is a rigid part between two joints, and its pose is stored in the parent joint which is located near the root joint. For example, the pose of the bone of the lower leg is stored in the knee joint. More examples: Joint 4 (LEFT_KNEE): It stores the location information of the left knee joint and the pose of the bone of the left lower leg. Joint 7 (LEFT_ANKLE): It stores the location information of the left ankle joint and the pose of the bone of the left foot.

Waist-worn tracker
Waist-worn trackers can refine the capture of actions like bending over and twisting the waist and can also improve the stability of tracking in sitting and lying poses.



6. Object recognition
Information of external devices
Type-C interface of PICO Sense tracker provides connection to external devices and exchanges data with PICO headset. That can allow to access information such as: batteries level, button operations, vibration commands and key values of external device to be passed.
7. Field of recognition

Depth & Orientation	Recognition Spec	Remarks
Depth	152mm ~ 500mm	If the hands are too far from the device or too close to the body, the device can not recognize hand poses.
Upwards	57.5°	If the hands are lifted too high or put too low, the device can not recognize hand poses.
Downwards	72.5°	
Leftwards	61.5°	When the angle between the hand and the device is too large, the device can not recognize hand poses.
Rightwards	60.5°	



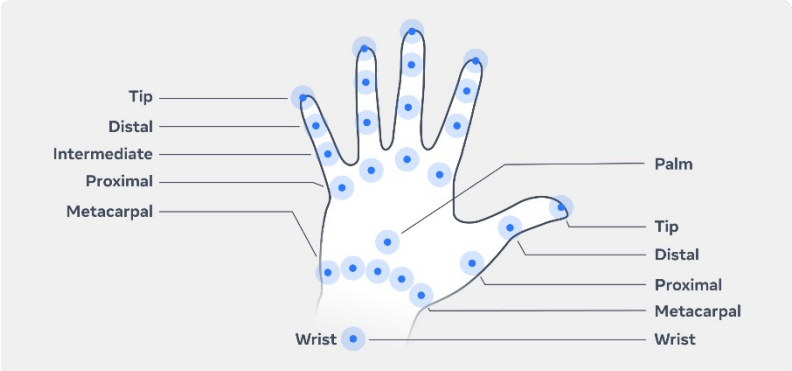
Meta
https://developers.meta.com/horizon/documentation/unity/unity-ovrinput
Equipment: App development: Windows/MAC; Tests & debugging: Windows or MAC with Meta Integration package.
Technical: 110° horizontal / 96° vericaly / 72Hz – 120Hz

1. Hand tracking
2. Keyboard
3. Body tracking
4. Face tracking
5. Eye tracking
6. Voice

Controller Animation – useful to provide indication for some actions. Shows short animation over an interactable object and input that needs to be provided while this action. – is there hand animations or is just a normal animation?

2. Hand tracking
Interactions SDK – provides standardized gestures and interactions facilitating tasks like tool usage, communication and tactile exploration. HD – Hand data is captured by HMD – head mounted display, and processed by software algorithms, involving machine learning and computer vision to recognize hand position and movement. Software interprets actions allowing user interaction with environment

Terminology



Joints - Joints in the hand provide flexibility and mobility. In XR these are referenced in code for determining interactions. See image above.
Wrist - The joint connecting the hand with the forearm, often tracked in code for hand position.
Fingers & thumb - The five digits of the human hand, consisting of four fingers and one opposable thumb, are essential for performing tactile interactions.

3D hand model – is generated from OVR Skeleton and OVR Mesh data. OVR Skeleton returns: bind pose, bone hierarchy and capsule collider data.

Root Pose – position of the hand in the tracking space. To adjust that we use – OVRHandPrefab. Depending on preferences if hand position depends on cameraRig position, we leave it unchecked, if hand position should be independent, we select checkbox – Update Root Pose.
Root Scale – Update Root Scale provides scaling of user hand model. Base scale is 100%.
Physic Capsules – is the volume of users bones use for collisions and triggering events. We use OVR Skeleton to adjust that by selecting – Enable Physic Capsules.
Customization – materials, shadows and probes.

Differences between OpenXR Hand Skeleton (New type in OpenXR, standard which gives compatibility across different platforms) and OVR Hand Skeleton

	OpenXR Hand Skeleton	OVR Hand Skeleton
Forearm	Not Included	Included
Palm	Included	Not Included
Thumb	Metacarpal, Proximal, Distal, Tip	Trapezium, Metacarpal, Proximal, Distal, Tip
Index	Metacarpal, Proximal, Intermediate, Distal, Tip	Proximal, Intermediate, Distal, Tip
Middle	Metacarpal, Proximal, Intermediate, Distal, Tip	Proximal, Intermediate, Distal, Tip
Ring	Metacarpal, Proximal, Intermediate, Distal, Tip	Proximal, Intermediate, Distal, Tip
Pinky	Metacarpal, Proximal, Intermediate, Distal, Tip	Metacarpal, Proximal, Intermediate, Distal, Tip
Total	26 Joints	24 Joints
Alignment	Joints aligned z-forward, y-up for both hands	Joints aligned x-forwards, y-up for the right hand, and joints are mirrored for the left hand

Information provided from OVR Skeleton and OVR Hand:

- Bone information
- Hand and finger position and rotation
- Pinch strength
- Pointer pose for UI raycasts
- Tracking confidence
- Hand size
- System gesture for opening universal menu

OVR Skeleton contains a full list of bone IDs, methods to implement interactions and detect gestures, calculate gesture confidence, target a particular bone, or trigger a collision event in the physics system. Examples:

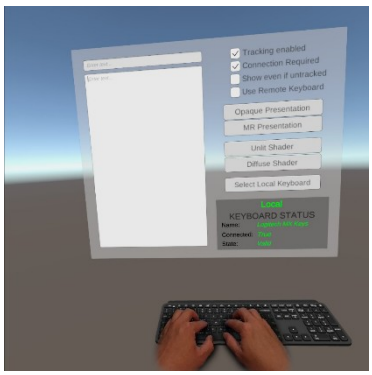
- GetCurrentStartBoneId() / GetCurrentEndBoneId() – used to iterate through subset of bones
- GetCurrentNumBones() / GetCurrentNumSinnableBones() – return number of bones in the skeleton that are skinnable. Skinnable – bones with anchors for the fingertips.

OVR Hand provides pointer pose and pinch gesture
Pinch can provide other information's like: is hand currently pinching, pinch strength and confidence level

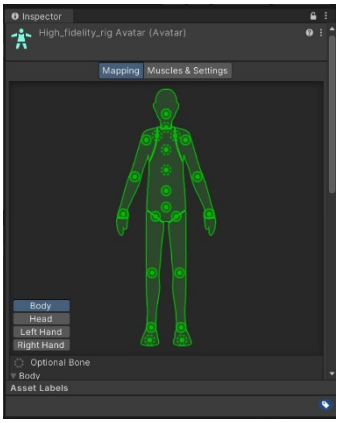
- GetFingerIsPinching() – returns bool
- GetFingerPinchStrength() – returns float 0-1
- GetFingerConfidence()

Pointer pose – consistent pose across Meta Quest apps. It indicates starting point position of the ray in the tracking space. Mostly use for UI indications.
Hand Confidence – returns low or high ?
Hand Scale – gets scale of the user hand relatively to the hand model, returns float as a scale for example: 1.05 which means that hand is 5% bigger than model.
Check System Gestures – allows to transition to universal menu through palm gesture facing user and pinch. With non-dominant hand it turns on Button – Start event.

Features:
WMM – Wide Motion Mode – allows to track hand and display plausible hand poses even when hands are out of the headset FOV. Its thanks to IOBT. Requires Body tracking to estimate hand position
Capsense – animations provided to create indication for user holding controllers. Natural – when user is holding controllers which are not displayed, Controller hand pose – when user is holding controllers and virtually animated hand models are visible.
Multimodal - provides simultaneous tracking for both controllers and hands. Multimodal overrides others transition methods.
Fast Motion Mode – high frequency hand tracking, advised due to fast hands movement.
3. Keyboard
Provides users with their physical keyboard inside VR environment. Keyboard is from inside Meta XR All in One SDK which is expecting from user to have one of the keyboards models from the list.



4. Body tracking

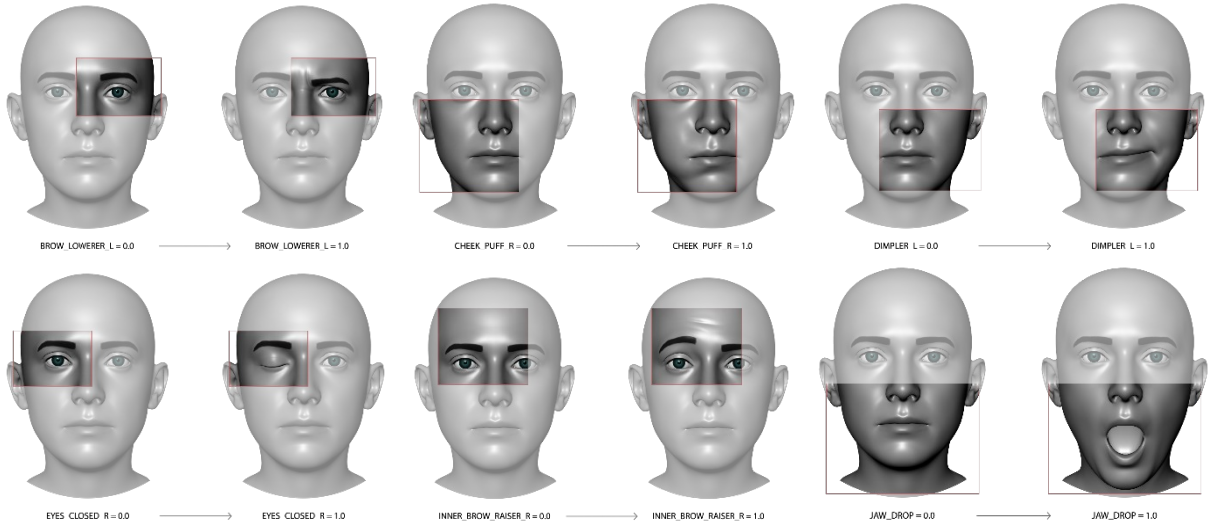


Body tracking is defined as OpenXR Extension. For Unity there are scripts like OVRBody() that abstract OpenXR interface.
API that uses hands/controllers and headset to infer the body pose which is after transformed into a body tracking skeleton. Body tracking API infer the movement of the person by repetitive calls for it. Body tracking allows to work with upper body part or full skeleton. Lower body is based on upper body is doing. It doesn't reflect accurately the legs.
Movement SDK provides access to the skeleton tracking and allows to animate with Unity Mecanim humanoid. Full raw data is in Avatar SDK.
Use cases:

- Use to analyze the movement of person,
- To animate the character to reflect human motions,
- To hit targets in gameplay by usage of body joints data
- Mapping body pose to non-playable characters.

5. Face tracking
The Face Tracking API converts the facial movements detected by the headset sensors into activations of expressions based on Facial Action Coding System (FACS). Cause people normally trigger couple of facial actions at once, the API returns a weight corresponding to the strength of the expression. This list of expressions along with their strength activate the blendshapes. Blendshapes can be restricted to creators needs, for example we can map only expressions like lip corner raiser. The Face Tracking API provides representation of most of the face like nose, jaw, eyebrows and close eyes area.

Which headset models support face tracking?
Natural Facial Expressions, which estimates facial movements based on inward facing cameras, is only available on Meta Quest Pro headsets. However, audio-driven face tracking is also available on Meta Quest 2 and Meta Quest 3 devices with the same API.



6. Eye tracking

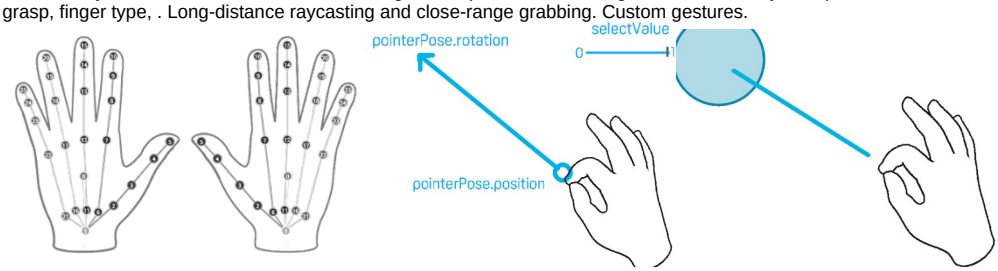
The Meta Quest Pro is the only device that supports this feature, utilizing the OVREyeGaze script. OVREyeGaze provides eye tracking or gaze information. Eye pose data is retrieved from OVRPlugin. When OVREyeGaze component added to a GameObject, it can simulate an eye based on actual human eye. Component can also select objects in a scene using raycasts. Currently, only the Quest Pro supports Eye Tracking.

7. Voice Tracking
The Voice SDK enables voice interaction in the app. Powered by Wit.ai Natural Language Understanding (NLU) can be used without prior AI/ML knowledge.
Pross: minimization usage of virtual keyboard while searching through nested menus; voice FAQ; usage as actions trigger during a gameplay.

VIVE – HTC Vive XR Elite
Unity: Overview | VIVE OpenXR - Developer Resources
Download: OpenXR for Unity - Developer Resources (vive.com)
Unity: Tutorials | VIVE OpenXR - Developer Resources

Equipment: App development: MAC/Windows; Tests & debugging: Both from Windows and MAC it needs to be tested and debugged through SteamVR and Steam accordingly.
Technical : 110° / 90Hz

1. Hand tracking
   Defines 26 joints with information about: tracking status, position, rotation, gesture info, hand joints, pose, confidence, scale, wrist velocity, grasp, finger type, . Long-distance raycasting and close-range grabbing. Custom gestures.

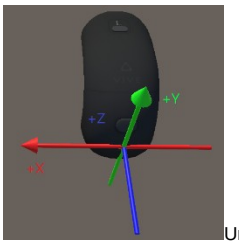


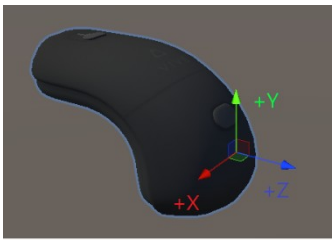
2. Hand Interaction – Realistic hand interaction
   Natural hand gestures for specific objects. Collider matching default hand model for more realistic effect.

3. Facial Tracking: jaw, mouth, cheek, eye, tongue
   It's a combination of Eye Expression and Lip Expression that we call Facial Tracking. Eyes and lips expressions are both ranges between 0 and 1 and it describes eye and mouth openness. Combination of this values then describe facial expressions.

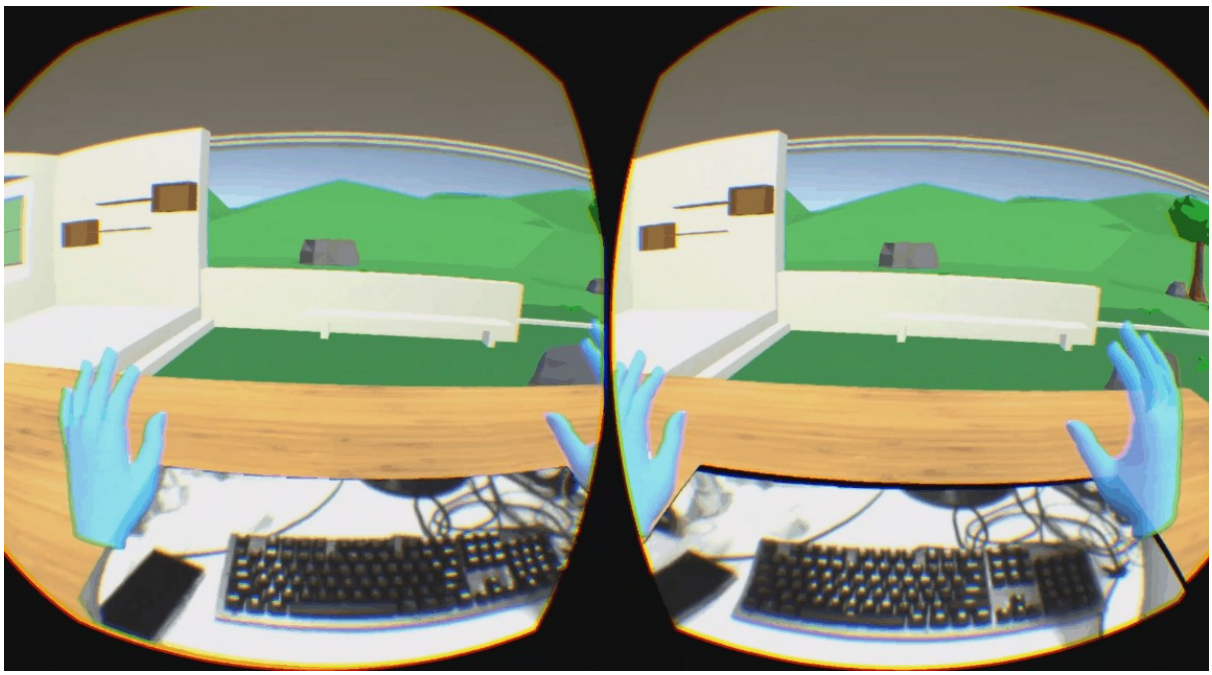
It also uses 52 blendshapes which cover: jaw, mouth, cheek, eye, tongue.

4. Wrist Tracker

OpenXR 

Unity 

   Data: buttons (primary, menu) input, device pose, state of tracking, state of being tracked
5. Eye gaze
   Only data that we can retrieve is eye pose,
6. Passthrough
   Planar Passthrough let user see and use real objects while being inside VR



7. Ultimate tracker
8. Anchor
9. Plane Detection
10. Display Refresh Rate
11. Foveation – rendering optimalization by reduction of pixel density in the peripheral vision.
12. Composition layer
    Results witch better and cleaner image thanks to layering of content. Content is then rendered by the headset not by Unity.

Next:
Face tracking – more information's, deeper research
Precision – passing one thing from one hand to another
Manipulation of small objects

Literatura:

(1) Digital Crown - https://developer.apple.com/design/human-interface-guidelines/digital-crown