WYKŁAD 10

Mobilne aplikacje XR

Programowanie aplikacji mobilnych

dr inż. Mateusz Pomianek

XR · ARCore · ARKit · OpenXR · WebXR · Spatial Computing

- Wprowadzenie do XR - AR, VR, MR
- Sprzęt XR - HMD, sensory, tracking
- ARCore i ARKit - platforma mobilna
- OpenXR - standard API
- WebXR i przyszłość

© Katedra Informatyki i Automatyki - Programowanie aplikacji mobilnych XR - v1.46

Plan wykładu

1. Klasyfikacja i hardware: AR, VR, MR, ER
2. ARCore (Android), ARKit (iOS), OpenXR, WebXR
3. Śledzenie i SLAM
4. Urządzenia wejścia
5. Optymalizacja i wydajność XR
6. Bezpieczeństwo i prywatność w XR

Historia i klasyfikacja XR: od Sensorama do Apple Vision Pro

1962 **Sensorama**
Pierwsze urządzenie immersyjne. Multisensoryczne: obraz 3D, dźwięk, wibr., zapach. Morton Heilig.

1968 **Sword of Damocles**
Ivan Sutherland. Pierwsze HMD z śledzeniem głowy. Wireframe. CRT na głowie.

1994 **CAVE**
Cave Automatic Virtual Environment. Projekcja na ściany. Marka Skali CAD/MED.

2013 **Google Glass**
AR na żądanie dla konsumentów. IMU + mikrokamera + display pryzmatyczny. Porażka rynkowa → przemysł.

2016 **Hololens 1**
Microsoft. Standalone MR HMD. HPU (Holographic Proc. Unit). 4 kamery + IMU + ToF. See-through hologram.

2019 **Quest 1**
Meta. 6DoF standalone VR. Snapdragon 835. Inside-out tracking. $399 - masowy rynek.

2023 **Vision Pro**
Apple. M2+R1 chip. 23M pikselowych mikrodisplayów per oko. EyeSight. $3499 developer kit.

2025 **AI-Powered XR**
On-device AI 80 TOPS. Semantic scene understanding. Foundation models + spatial computing.

Sprzęt XR: HMD, sensory i śledzenie

Tethered HMD

Podłączony do PC/konsoli. Nvidia 4090 → full RT. Valve Index, PlayStation VR2. Kabel DisplayPort/USB-C.

Standalone HMD

Własny SoC (Snapdragon XR2+). Meta Quest 3, Vision Pro M2. 6DoF inside-out tracking. Autonomia 2–4h.

Smartfon + holder

Google Cardboard / Gear VR. DOF: 3DoF (obrot). Brak tracking pozycji. Low cost. Edukacja.

AR Glasses

See-through waveguide. Ostry kąt widzenia ~50°. HoloLens 2, Magic Leap 2. B2B focus.

sensory i tracking

IMU (6-axis)

Akcelerometr + żyroskop. 1000Hz+ sampling. Rotacja głowy < 1ms latencja.

Camera-based tracking

SLAM + feature points. ORB-SLAM3. 60fps stereo.

ToF / Structured light

Głębokość 3D: Apple LiDAR, Azure Kinect. < 1% błąd na 3m.

Eye-tracking

PSVR2: IR LED + sensor 120Hz. Apple VP Iris. Foveated rendering.

Hand Tracking

MediaPipe Hands 21 punktów. HoloLens 2: artykuł. Quest 3 no-controller.

GPS + Compass

Outdoor AR (Niantic). Dokładność 3m. RTK: ±10cm. UWB: < 10cm.

Śledzenie przestrzenne i stopnie swobody (DoF)

3 DoF: tylko orientacja głowy

Pitch (pochylenie), **Yaw** (obrót), **Roll** (przechylenie)

TYLKO rotacja głowy; brak translacji

Urządzenia: Google Cardboard, Oculus Go

Tylko na IMU: tanie, niska latencja

Problem: teleportacja zamiast chodzenia

6 DoF: pełny ruch w 3D

3DoF rotacja + X/Y/Z translacja (ruch w przestrzeni)

Inside-out: kamery na HMD śledzą otoczenie

Outside-in: bazy (Base Stations) śledzą HMD

Urządzenia: Quest 3, Valve Index, HoloLens

Wymaga SLAM lub lighthouse tracking

Inside-out tracking = kamera na HMD kalkuluje pozycję względem środowiska (SLAM).
Outside-in = stacje bazowe emitują sygnał, HMD odbiera. Room-scale wymagany dla 6DoF.

ARCore (Android): Fundamenty platformy AR

Kotlin

```
fun configure(session: Session) { session.configure(Config(session).apply
{ planeFindingMode =
Config.PlaneFindingMode.HORIZONTAL_AND_VERTICAL depthMode = if
(session.isDepthModeSupported(Config.DepthMode.AUTOMATIC))
Config.DepthMode.AUTOMATIC else Config.DepthMode.DISABLED
lightEstimationMode =
Config.LightEstimationMode.ENVIRONMENTAL_HDR cloudAnchorMode =
Config.CloudAnchorMode.ENABLED })}

fun place(frame: Frame, tap: MotionEvent) { val hit =
frame.hitTest(tap).firstOrNull { val p = it.trackable as? Plane ?:
return@firstOrNull false p.trackingState == TrackingState.TRACKING &&
p.isPoseInPolygon(it.hitPose) } ?: return


val anchorNode = AnchorNode(engine, hit.createAnchor())
anchorNode.addChildNode(ModelNode( modelInstance =
modelLoader.createModelInstance("models/robot.glb"), scaleToUnits =
0.5f )) arSceneView.scene.addChildNode(anchorNode)}
```

Motion Tracking (VIO)
Visual-Inertial Odometry: kamera + IMU. 60Hz pose estimation. Feature points ORB.

Plane Detection
Horizontal i Vertical planes. Mesh triangulation. Environmental HDR lighting.

Depth API
ARCore + ToF/stereo: depth map 180°. Okkluzja cyfrowych obiektów przez realne.

Cloud Anchors
Kotwice zapisane w chmurze Google. Multiplayer AR. Persistent AR od nowa.

Geospatial API
Google Street View + ARCore = AR outdoor z decymetrową dokładnością GPS+VPS.

ARCore: semantyka sceny, Lighting i RecordingDataset

Scene Semantics API

ARCore 1.34+: segmentacja semantyczna przez ML. Klasy: niebo, podłoga, ściana, roślina, okno, stół, krzesło. 256×192 @ 30fps. Pozwala przykleić efekty do klasy sceny.

Raw Camera Access + Image API

AugmentedImageDatabase: baza obrazów do śledzenia (QR-like). Każdy obraz ma pose 6DoF. Raw CPU image: dostęp do YUV/JPEG dla własnego ML pipeline (object detection etc.).

Environmental HDR Lighting

Kamera skierowana w tył → sferyczna mapa środowiska. Spherical Harmonics (SH) 9 współczynników + directional light. PBR shading realistyczny dzięki prawdziwemu HDR otoczenia.

AR Foundation (Unity/Unreal)

Abstrakcja nad ARCore (Android) i ARKit (iOS). Jeden kod, dwie platformy. SubSystem: XR Planes, XR Anchors, XR Environment Probes. Zalecane dla multi-platform XR.

Recording i Playback (Dataset)

ARCoreRecorder: zapis sesji AR do MP4+metadata. Odtwarzanie w emulatorze lub innym urządzeniu. CI/CD dla AR: reprodukowalne testy automatyczne scen AR bez fizycznego urządzenia.

ARKit (iOS): RealityKit, LiDAR i People Occlusion

```
import SwiftUIimport RealityKit

struct ARContentView: View { @State private var anchor =
AnchorEntity(.plane( .horizontal, classification: .floor, minimumBounds: [0.2,
0.2] ))

var body: some View { RealityView { content in if let model = try? await
Entity(named: "robot.usdz") { model.scale = .init(repeating: 0.01)
anchor.addChild(model) } content.add(anchor) } update: { _ in // Aktualizacja
sceny AR. } }}
```

Swift

LiDAR Scene Reconstruction
iPhone 12 Pro+: mesh 3D sceny. Plane detection 10× szybciej. People Occlusion: ludzie zakrywają AR.

RealityKit 2.0
Silnik AR Apple. Physically Based Rendering, Custom Shaders (Metal). Entity-Component.

Object Scanning API
Skanowanie przedmiotów 3D. ARObjectAnchor: śledzenie znanych obiektów.

Body Tracking
ARBodyTrackingConfiguration: 91 jointów ciała. Skeleton pose. Armature AR characters.

Collaborative Session
MultipeerConnectivity + ARKit. Każde urządzenie dodaje kotwice. Sync. peer-to-peer.

Reality Composer Pro, USDZ i visionOS

Reality Composer Pro
Narzędzie Xcode do tworzenia scen AR. Drag & drop obiektów USDZ. Behaviors: animacja, physics, audio, custom Swift Actions. Particle systems. Import FBX/OBJ/USDZ.

USDZ - Universal Scene Description
Apple + Pixar format 3D. Jedno-plikowe archiwum: geometria + materiały + animacje. AR Quick Look: podgląd AR z Safari/Mail/Messages bez instalacji aplikacji.

visionOS i Spatial Computing
Apple Vision Pro OS. WindowGroup, ImmersiveSpace, Volume. SwiftUI + RealityKit native. Passthrough HDR. EyeSight avatar. Hand gestures bez kontrolerów.

ShaderGraph (Metal Shaders)
Reality Composer Pro: node-based shader editor. Custom PBR materials: emission, metallic, roughness, normal maps. Dostęp do ARKit depth w shader (okkluzja custom).

ARKit 6 - Image Tracking 4K
ARImageTrackingConfiguration: śledzenie obrazów 4K. Mapy miast (ARGeoTrackingConfig). Coaching overlay guidance. Automatic environmental meshing iPhone 15 Pro.

OpenXR: Standardowy API dla XR

Architektura OpenXR

Aplikacja (XR App / Game Engine)

↓ *xrCreateInstance() / xrCreateSession()*

OpenXR API (Loader / Runtime)

↓ *Extensions: XR_KHR_vulkan_enable2, XR_EXT_eye_gaze*

Vendor Runtime (Meta/Valve/PICO/Pico)

↓ *HAL / Driver*

Sprzęt HMD / GPU

Kluczowe rozszerzenia OpenXR

XR_KHR_vulkan_enable2
Vulkan rendering. Zero-copy swapchain. Direct timewarp.

XR_EXT_eye_gaze
Eye tracking: gaze direction + pose. Foveated rendering.

XR_FB_hand_tracking
Meta rozszerzenie: 26 jointów na ręka. Hand mesh.

XR_KHR_composition_layer
Multiple layers: quad, cylinder, cube, equirect.

XR_EXT_performance_settings
CPU/GPU level hint dla runtime. Thermal throttle prevention

XR_MSFT_scene_understanding
Microsoft: planes, meshes, objects z depth sensor.

OpenXR: Inicjalizacja, Session Loop i Rendering

C++

```
// 1. Instance creation
XrInstanceCreateInfo info{};
info.type = XR_TYPE_INSTANCE_CREATE_INFO;
info.applicationInfo.apiVersion = XR_CURRENT_API_VERSION;
strcpy(info.applicationInfo.applicationName, "MyXRApp");
xrCreateInstance(&info, &instance);

// 2. System (HMD form factor)
XrSystemGetInfo sysInfo{};
sysInfo.formFactor = XR_FORM_FACTOR_HEAD_MOUNTED_DISPLAY;
xrGetSystem(instance, &sysInfo, &systemId);

// 3. Session z Vulkan
XrGraphicsBindingVulkan2KHR vkBinding{};
vkBinding.instance = vkInstance;
vkBinding.device = vkDevice;
XrSessionCreateInfo sessionInfo{};
sessionInfo.next = &vkBinding;
xrCreateSession(instance, &sessionInfo, &session);

// 4. Reference space (STAGE = room scale)
XrReferenceSpaceCreateInfo spaceInfo{};
spaceInfo.referenceSpaceType = XR_REFERENCE_SPACE_TYPE_STAGE;
xrCreateReferenceSpace(session, &spaceInfo, &space);
```

```
// 5. Frame loop (simplified)
while (running) {
    XrEventDataBuffer event{};
    while (xrPollEvent(instance, &event) == XR_SUCCESS) {
        handleEvent(event);
    }
    XrFrameWaitInfo waitInfo{};
    XrFrameState frameState{};
    xrWaitFrame(session, &waitInfo, &frameState);

    xrBeginFrame(session, nullptr);

    // Pobierz pose obu oczu
    XrViewLocateInfo locateInfo{};
    locateInfo.viewConfigurationType =
        XR_VIEW_CONFIGURATION_TYPE_PRIMARY_STEREO;
    locateInfo.displayTime = frameState.predictedDisplayTime;
    locateInfo.space = space;
    std::vector<XrView> views(2);
    xrLocateViews(session, &locateInfo, ..., views.data());

    // Render do swapchain per oko
    renderEyes(views, frameState.predictedDisplayTime);
    xrEndFrame(session, &frameEndInfo);
}
```

Rendering XR: Reprojection, Foveated Rendering i latencja

Asynchronous Timewarp (ATW)

Reprojection ostatniej klatki gdy GPU nie nadąża. Obraca obraz proporcjonalnie do ruchu głowy. Eliminuje 'judder'. Wymagane: < 20ms MTP (Motion-to-Photon) dla brak choroby lokomocyjnej.

Foveated Rendering (FR)

Eye-tracking → centralny obszar full res, peryferyjny 1/4 res. Fixed FR: stały obszar. Variable FR (VFR): dynamiczny wg gaze. Oszczędność GPU 40–50%. Quest 3 + PSVR2 hardware VFR.

Multiview Rendering

GPU renderuje oba oczy w jednym draw call. Geometry Shader lub Vertex Amplification replikuje wierzchołki dla View L/R. 1.5–1.8× szybciej vs dwa osobne draw calle.

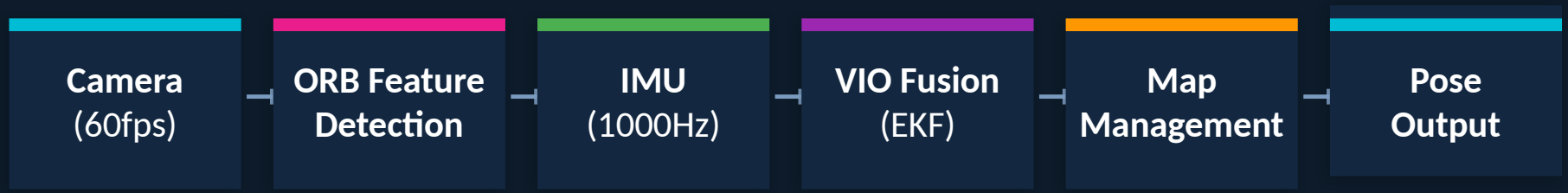
Supersampling (MSAA / DLSS)

VR wymaga anti-aliasing - piksele są duże przez soczewki. MSAA 4× standard. DLSS/FSR: upscaling z niższej rozdzielczości + temporal accumulation = lepsza jakość / niższy koszt.

MTP Latency i Motion Sickness

Motion-to-Photon < 20ms: ruch głowy → obraz. > 20ms = choroba lokomocyjna. Breakdown: Sensor 1ms + Tracking 2ms + App CPU 3ms + GPU 8ms + Compositor 2ms + Display 4ms.

Tracking i SLAM: Visual-Inertial Odometry w praktyce



ORB Feature Detection

Oriented FAST + Rotated BRIEF. 500-2000 punktów/klatkę. Niezmiennicze na rotację i skalę. 10× szybsze od SIFT. Używane w ORB-SLAM3, ARCore, ARKit.

Extended Kalman Filter (EKF)

Połączenie IMU (predykcja 1000Hz) + kamera (korekta 30–60Hz). State vector: [pos, vel, orient, IMU bias]. Noise covariance tuning kluczowe dla stabilności.

Loop Closure Detection

DBoW2: bag-of-words podobieństwo klatek. Rozpoznanie poprzedniej lokalizacji = korekta dryfu. Global map optimization: g2o graph-based bundle adjustment.

Plane Extraction

RANSAC na depth/pointcloud → płaskie obszary. Normal estimation: PCA na sąsiedztwie punktów. Merge: kąt <15° + odległość <5cm = ten sam plane.

Wejście XR: Kontrolery, Hand Tracking i Eye Gaze

6DoF Kontrolery

Trigger, grip, joystick, A/B/X/Y. Haptic feedback (LRA vibration 100–3000Hz). Touch capacitive: palm presence. Tracking przez IR LED constellation lub inside-out CV. OpenXR: XrAction binding.

Hand Tracking (bez kontrolerów)

MediaPipe Hands: 21 punktów per ręka, 30fps. Meta Quest: model ML na kamerach monochromatycznych. Pinch, grab, point gestures. Latencja ~20ms. Problem: occlusion (ręce zasłaniają się).

Eye Gaze Tracking

IR diody + sensor 120Hz. Gaze ray: origin + direction w world space. Zastosowanie: foveated rendering, attention analytics, eye-based UI control (dwell click). PSVR2, Vision Pro (Iris scan).

Voice Input + NLP

On-device STT: Whisper 1B lub Google STT. Intenty głosowe: 'Hey Siri', komedy. visionOS: głos + gaze + pinch = naturalne UI. Offline intent classification dla prywatności.

Haptyka zaawansowana

ThermoReal: ciepło/zimno na dłoniach. OWO: elektryczna stymulacja mięśni (EMS) pełne ciało. bHaptics TactSuit: 40 ERM silniczków na kamizelce. Force Feedback Gloves: HaptX.

OpenXR Input System: Action Bindings i Interaction Profiles

C++ / OpenXR

```
#define XR_CHECK(x) if (XR_FAILED(x)) throw std::runtime_error(#x)

struct Input { XrActionSet set; XrAction trigger; XrAction handPose; };

Input createInput(XrInstance instance, XrSession session) { Input in{};

XrActionSetCreateInfo set{XR_TYPE_ACTION_SET_CREATE_INFO}; strcpy(set.actionSetName, "gameplay");
strcpy(set.localizedActionSetName, "Gameplay"); XR_CHECK(xrCreateActionSet(instance, &set, &in.set));

XrActionCreateInfo a{XR_TYPE_ACTION_CREATE_INFO}; a.actionType = XR_ACTION_TYPE_FLOAT_INPUT; strcpy(a.actionName,
"trigger_pull"); strcpy(a.localizedActionName, "Trigger Pull"); XR_CHECK(xrCreateAction(in.set, &a, &in.trigger));

a = {XR_TYPE_ACTION_CREATE_INFO}; a.actionType = XR_ACTION_TYPE_POSE_INPUT; strcpy(a.actionName, "hand_pose");
strcpy(a.localizedActionName, "Hand Pose"); XR_CHECK(xrCreateAction(in.set, &a, &in.handPose));

XrPath triggerPath, posePath, profile; xrStringToPath(instance, "/user/hand/right/input/trigger/value", &triggerPath); xrStringToPath(instance,
"/user/hand/right/input/grip/pose", &posePath); xrStringToPath(instance, "/interaction_profiles/oculus/touch_controller", &profile);

XrActionSuggestedBinding bindings[] = {{in.trigger, triggerPath}, {in.handPose, posePath}}; XrInteractionProfileSuggestedBinding
suggested{XR_TYPE_INTERACTION_PROFILE_SUGGESTED_BINDING}; suggested.interactionProfile = profile;
suggested.countSuggestedBindings = 2; suggested.suggestedBindings = bindings; XR_CHECK(xrSuggestInteractionProfileBindings(instance,
&suggested));

XrSessionActionSetsAttachInfo attach{XR_TYPE_SESSION_ACTION_SETS_ATTACH_INFO}; attach.countActionSets = 1; attach.actionSets
= &in.set; XR_CHECK(xrAttachSessionActionSets(session, &attach)); return in;}

void updateInput(XrSession session, const Input& in) { XrActiveActionSet active{in.set, XR_NULL_PATH}; XrActionsSyncInfo
sync{XR_TYPE_ACTIONS_SYNC_INFO}; sync.countActiveActionSets = 1; sync.activeActionSets = &active;
XR_CHECK(xrSyncActions(session, &sync));

XrActionStateGetInfo get{XR_TYPE_ACTION_STATE_GET_INFO}; get.action = in.trigger; XrActionStateFloat
state{XR_TYPE_ACTION_STATE_FLOAT}; XR_CHECK(xrGetActionStateFloat(session, &get, &state)); if (state.isActive && state.currentState
> 0.5f) fireWeapon();}
```

WebXR: Immersive Web i przeglądarkowy AR/VR

```javascript
// WebXR Device API - immersive-ar mode
async function startAR() {
  if (!navigator.xr) return; // sprawdź wsparcie
  const supported = await navigator.xr.isSessionSupported('immersive-ar');
  if (!supported) { showFallback(); return; }

  const session = await navigator.xr.requestSession('immersive-ar', {
    requiredFeatures: ['hit-test', 'local-floor'],
    optionalFeatures: ['dom-overlay', 'depth-sensing'],
    domOverlay: { root: document.getElementById('overlay') }
  });

  const gl = canvas.getContext('webgl2', { xrCompatible: true });
  await gl.makeXRCompatible();
  session.updateRenderState({
    baseLayer: new XRWebGLLayer(session, gl)
  });

  // Hit-test source dla plane detection
  const hitTestSource = await session.requestHitTestSource({
    space: viewerSpace  // promień z centrum widoku
  });

  session.requestAnimationFrame(function onXRFrame(t, frame) {
    const hits = frame.getHitTestResults(hitTestSource);
    if (hits.length > 0) {
      const pose = hits[0].getPose(refSpace);
      placeReticle(pose.transform.matrix); // wskaźnik AR
    }
    session.requestAnimationFrame(onXRFrame);
  });
}
```

Tryby sesji
inline (w stronie), immersive-vr (pełny VR), immersive-ar (passthrough AR). Chrome Android: immersive-ar przez ARCore.

Three.js / Babylon.js
Three.js WebXRManager: session.requestAnimationFrame, XRReferenceSpace. A-Frame: deklaratywny HTML. Model Viewer: <model-viewer ar>.

WebXR Layers API
XRWebGLLayer (std), XRCubeLayer, XRCylinderLayer, XRQuadLayer. Warstwy renderowane przez runtime = niższa latencja niż w WebGL.

Depth Sensing API
XRDepthInformation: bufor głębokości. Dostępna w Chrome 120+ Android. Okkluzja AR bez ARCore native. Privacy: permission required.

Anchors i Hit-Testing
XRAnchor: trwałe zakotwiczenie. Hit-test: rzutowanie promienia na wykryte płaszczyzny. Persistent Anchors: IndexedDB storage.

Tworzenie aplikacji AR na Androida: ARCore + SceneView + Kotlin

Kotlin

```kotlin
// build.gradle.kts - zależności
implementation("io.github.sceneview:arsceneview:2.2.1")

// ARSceneView w Compose - najprostsza forma AR
@Composable
fun ARScreen() {
    val engine = rememberEngine()
    val modelLoader = rememberModelLoader(engine)
    val nodes = rememberNodes()

    ARScene(
        modifier = Modifier.fillMaxSize(),
        engine = engine,
        modelLoader = modelLoader,
        nodes = nodes,
        planeRenderer = true,  // rysuj wykryte płaszczyzny
        onSessionUpdated = { session, frame ->
            // Sprawdź trackingState
            if (frame.camera.trackingState == TrackingState.TRACKING) {
                handleTracking(frame)
            }
        },
        onGestureListener = rememberOnGestureListener(
            onSingleTapConfirmed = { motionEvent, node ->
                if (node == null) {  // trafiono w pusty teren
                    placeObjectOnPlane(motionEvent, session, nodes)
                }
            }
        )
    )
}
```

setup i wymagania

1. AndroidManifest.xml
CAMERA permission + meta-data ar.google.arcore.minApkVersion: 190700

2. Sprawdź kompatybilność
ArCoreApk.getInstance().checkAvailability() → SUPPORTED_INSTALLED

3. Prośba o pozwolenia
ActivityResultContracts.RequestPermission dla CAMERA

4. Konfiguruj sesję
Config z planeFindingMode, depthMode, lightEstimation

5. Dodaj model GLTF/GLB
assets/models/robot.glb - Binary GLTF z PBR materiałami

6. Testuj na urządzeniu
adb install + logcat filter: ARCore

Tworzenie Aplikacji AR na iOS: ARKit + RealityKit + SwiftUI

Swift

```
// ARKit 6 + RealityKit 3 + SwiftUI
import SwiftUI
import RealityKit
import ARKit

struct ContentView: View {
    @State private var arView = ARView(frame: .zero)
    var body: some View {
        ARViewContainer(arView: $arView)
            .ignoresSafeArea()
            .overlay(alignment: .bottom) { ControlPanel() }
    }
}

struct ARViewContainer: UIViewRepresentable {
    @Binding var arView: ARView
    func makeUIView(context: Context) -> ARView {
        let config = ARWorldTrackingConfiguration()
        config.planeDetection = [.horizontal, .vertical]
        config.sceneReconstruction = .meshWithClassification
        config.frameSemantics = .personSegmentationWithDepth
        arView.session.run(config)
        arView.debugOptions = [.showSceneUnderstanding]
        // Gest - tap to place
        let tap = UITapGestureRecognizer(target: context.coordinator,
            action: #selector(Coordinator.handleTap))
        arView.addGestureRecognizer(tap)
        return arView
    }
}
```

People occlusion
LiDAR + AI: ludzie zasłaniają obiekty AR. ARConfiguration.frameSemantics.personSegmentation

Scene geometry (Mesh)
sceneReconstruction.meshWithClassification: mesh 3D sceny + etykiety (ściana, sufit, podłoga).

Object & image anchors
ARObjectAnchor dla zeskanowanych obiektów. ARImageAnchor dla QR-like image targets.

Spatial audio (RealityKit)
AudioFileResource + Reverb preset. 3D dźwięk pozycjonowany do Entity w scenie AR.

Reality composer Pro Flow
1. Stwórz scenę .usda → 2. Dodaj behaviors → 3. Export .usdz → 4. Załaduj w ARView.loadAsync.

Optymalizacja i wydajność XR: frame pacing, bateria i termika

72–120	<20	<8	45°C
Target FPS	MTP Latencja	GPU Frame	Termika max

Frame Pacing i VSync

Android GameActivity + Swappy: synchronizacja do 72/90/120Hz. Bez Swappy: tearing/judder. Frame pacing variance <2ms dla VR comfort.

LOD i Mesh Optimization

Draco compression: mesh 90% mniejszy. LOD0/1/2: zmniejsz poly gdy dalej. Billboards: 2D dla odległych obiektów. Texture atlasy: minimalizacja draw calls.

Batching i Instancing

GPU Instancing: tysiące identycznych obiektów w 1 draw call (drzewa, kamienie). Dynamic batching: małe meshe. SRP Batcher (Unity): UBO per material.

Zarządzanie baterią AR

ARCore: setFocusMode(FIXED) = wyłącz AF ciągły. Ogranicz planeFindingMode gdy planes już znalezione. Zmniejsz camera FPS z 60 do 30 gdy statyczna scena.

Thermal Throttling Prevention

Android Thermal API: PowerManager.getThermalHeadroom(). Gdy < 1.0: zredukuj FPS, wyłącz efekty. CpuHeadroom > 0.5: pełna jakość. DTPM callback.

Bezpieczeństwo i prywatność w XR: dane przestrzenne i GDPR

Dane Przestrzenne (Spatial Data)

Mapy 3D pomieszczeń = wrażliwe dane osobowe (RODO art.4). ARCore Cloud Anchors: szyfrowanie AES-256 w tranzycie. Apple Vision Pro: Room maps lokalnie na urządzeniu, niesynchronizowane z iCloud.

Eye-Tracking Privacy

Dane ruchów oczu ujawniają: stan emocjonalny, zainteresowania, choroby (Alzheimer, ALS). Meta: dane gaze przechowywane on-device. Illinois BIPA: biometria wymaga zgody. GDPR = sensitive data.

Deepfake i Prezencja Cyfrowa

Photorealistic avatars + voice cloning = tożsamość cyfrowa. EEA: Digital Identity Regulation. Watermarking C2PA dla treści AI. visionOS EyeSight: twarz nie jest w pełni cloneable w SDK.

XR Phishing i Social Engineering

Overlay AR na screen: bankowe UI fałszywe w AR. Typosquatting w Metaverse domenach. FIDO2/WebAuthn dla auth w XR. Biometric spoofing prevention: liveness detection.

Regulacje i Certyfikacja

FCC SAR dla nadawania RF w HMD. CE Red Directive. HIPAA dla medical XR (FDA SaMD). CE/MDR dla XR terapeutycznego. ISO/IEC 30160: standardy bezp. XR (draft 2024).

Studium przypadku: Meta Quest 3 i architektura MR headset

Snapdragon XR2 Gen2
SoC

8 GB LPDDR5
RAM

2064×2208 per oko
Display

90/120 Hz
Refresh

Mixed Reality Passthrough
4 monochrome + 2 RGB kamery. 18ppd passthrough (pixel per degree). Depth estimation dla okkluzji. Color passthrough 1st w Quest.

Inside-Out 6DoF Tracking
Bez stacji bazowych. 6DoF głowa + 6DoF kontrolery. Hand Tracking 2.0: 26 jointów. Camera-to-IMU kalibr. na urządzeniu.

Pancake Optics
Fresnel → Pancake: 40% cieńszy headset. 3× odbicia via BS + polarizer. Ciemniejszy obraz (-30% transmisja). Eye-relief 0–7mm.

Snapdragon XR2 Gen2 specifics
4nm TSMC. CPU Kryo 780. Adreno 740. Hexagon DSP NPU ~15 TOPS. WiFi 6E (6GHz) Air Link 2.1 Gbps streaming z PC.

Meta Horizon OS (Android base)
Android 12 AOSP fork. Meta XR SDK. OpenXR 1.0 conform. Spatial Anchors API. Presence Platform: Scene Understanding, Voice SDK.

Studium przypadku: Apple Vision Pro - Spatial Computer

M2 + R1	23M pix per oko	12 ms R1→display	12 szt. + 5 sensor.
Chip	Display	Latencja	Kamery

R1 Chip - Real-Time Processor

Dedykowany koprocesor. Przetwarza: 12 kamer + 5 sensorów → display w 12ms. Streaming 4K per oko 90fps. Niższy od reaction time człowieka (250ms).

visionOS Programming Model

WindowGroup (2D app), ImmersiveSpace (AR overlay), Full Immersive Space (VR). SwiftUI native. RealityView 3D. ARKit visionOS: Plane/Image/Object tracking.

Micro-OLED Display

Sony custom. 3391ppi. Tandem OLED stacked. 1600 nit HDR. Pancake lenses. Optical persona: renderuje awatar twarzy na EyeSight display.

Enterprise i Przyszłość

$3499 developer. DiagonalXR Enterprise: surgical guidance. Stryker HoloSuite. iPad Pro 11 jako companion. Rumorowany Vision Pro 2 @ $2000 w 2026.

EyeSight i Social Cues

Widoczna twarz użytkownika innym. LEDs za szkłem: Immersed = ciemny, Present = twarz widoczna. Spatial Personas w FaceTime: hologram rozmówcy.

Ekosystem Narzędzi XR: Unity, Unreal, Godot i SDK

Unity XR Toolkit

XR Interaction Toolkit 3.0. AR Foundation (ARCore+ARKit). XR Plugin Management. OpenXR native. Universal RP + URP: mobile-optimized rendering. Asset Store: 1000+ XR assets.

Unreal Engine 5

Lumen GI + Nanite dla high-end VR/PC. MetaXR SDK plugin. Niagara particles. Pixel Streaming XR. C++/ Blueprint. Wymagania GPU znacznie wyższe niż Unity.

Godot 4 + GDExtension

Open-source. XR Tools addon (official). OpenXR support built-in. GDNative dla ARCore. Lżejszy niż Unity/UE. Rosnące wsparcie community. WebXR out-of-box.

Snap Lens Studio

AR lenses na Snapchat 300M+ DAU. Body tracking, face effects, world lenses. JavaScript + TypeScript. Connected Lens: multiplayer AR. Distribution: Camera Kit SDK.

Meta Presence Platform

Scene Understanding, Voice SDK, Interaction SDK, Body Tracking SDK, Avatar SDK. Horizon Worlds: social VR creator tool. OpenXR konformalny. Android Studio workflow.

8th Wall / Niantic Lightship

WebAR (WebXR): dowolna przeglądarka, AR bez instalacji aplikacji. Lightship ARDK: city-scale AR maps, shared AR, VPS. Pokémon GO infrastructure. Unity plugin.

Spatial Audio w XR: HRTF, Ambisonics i implementacja

Kotlin / ARCore Audio

```kotlin
// Google Resonance Audio SDK - spatial 3D audio
// build.gradle: implementation
'com.google.resonance.audio:sdk:1.0.0'
import com.google.vr.sdk.audio.GvrAudioEngine

class ARSoundManager(context: Context) {
    private val audioEngine = GvrAudioEngine(context,
        GvrAudioEngine.RenderingMode.BINAURAL_HIGH_QUALITY)
    fun placeSound(glbAnchorPose: Pose): Int {
        val soundId =
audioEngine.createSoundObject("footsteps.ogg")
        // Pozycja dźwięku = pozycja obiektu AR w przestrzeni
        audioEngine.setSoundObjectPosition(
            soundId,
            glbAnchorPose.tx(),  // X
            glbAnchorPose.ty(),  // Y
            glbAnchorPose.tz()   // Z w układzie sceny AR
        )
        audioEngine.setSoundObjectDistanceRolloffModel(
            return soundId)
        }
    fun updateListenerPose(cameraPose: Pose) {
        audioEngine.setHeadPosition(
            cameraPose.tx(), cameraPose.ty(), cameraPose.tz())
        audioEngine.setHeadRotation(
            cameraPose.qx(), cameraPose.qy(),
            cameraPose.qz(), cameraPose.qw())
        audioEngine.update() // wywołaj co klatkę
    }
```

HRTF: Head Related Transfer Function

Filtr binauralny symulujący małżowiny uszne. Spersonalizowany HRTF = najlepszy przestrzenny. Apple AirPods Pro: pomiar ucha iPhone kamerą.

Ambisonics (HOA)

Format dźwięku przestrzennego 360°. B-format 1st order: 4 kanały W/X/Y/Z. HOA 3rd order: 16 kanałów. YouTube 360 audio.

Occlusion i Reverb

Resonance Audio: Room model (shoe-box model). Ray casting audio: ściana blokuje dźwięk. Materiał (beton = odbicie, tkanina = pochłanianie).

Steam Audio (Valve)

Open-source. Convolution reverb z IR. GPU compute propagation. Unity + Unreal plugin. Geometry-aware reverb na podstawie mesh sceny.

AI i Machine Learning w XR: Scene Understanding i Generative AI

Semantic Scene Segmentation
ARCore Scene Semantics API: 8 klas per piksel @30fps. DepthLab: depth + semantics → obiektowa okkluzja. Zastosowanie: 'umieść roślinę tylko na podłodze', 'nie zakrywaj okna'.

Object Detection 3D
YOLO3D + ARCore depth: bounding box 3D per obiektu. MediaPipe ObjectDetector na klatce z ARKit. Zastosowanie: AR etykietki na meblach, automatyczne anchory.

Generative AI w Scenie
Stable Diffusion XL → texture inpainting AR surface. Spójne oświetlenie z LEA (Lighting Estimation). 3D object generation: text→GLTF (Shap-E, Point-E, Meshy AI).

NeRF i Gaussian Splatting
Neural Radiance Fields: fotografuj pomieszczenie → 3D reprezentacja neural. 3D Gaussian Splatting 2023: 100× szybszy render vs NeRF. Polycam, KIRI Engine apps.

On-Device LLM w XR Asystencie
Llama 3.2 3B (INT4, ~800MB) na Snapdragon XR2 Gen2 (~60 tok/s). Spatial assistant: 'co to jest?' (obraz → vision LLM → odpowiedź). visionOS Siri AI integration.

Wieloosobowe XR: Cloud Anchors, Networking i Metaverse

Kotlin

```kotlin
// ARCore Cloud Anchors - multiplayer AR
// Krok 1: hostuj anchor w chmurze Google
val hostFuture = session.hostCloudAnchorAsync(
    localAnchor,
    ttlDays = 365,  // czas życia anchor
    { cloudAnchorId, cloudState ->
        if (cloudState == Anchor.CloudAnchorState.SUCCESS) {
            // Wyślij cloudAnchorId innym przez Firestore/RTDB
            firestore.collection("shared_anchors")
                .add(mapOf("anchorId" to cloudAnchorId,
                           "roomId" to currentRoomId))
        }
    }
)

// Krok 2: resolwuj anchor u innych uczestników
fun resolveSharedAnchor(cloudAnchorId: String) {
    session.resolveCloudAnchorAsync(cloudAnchorId)
    { anchor, cloudState ->
        if (cloudState == Anchor.CloudAnchorState.SUCCESS) {
            // Anchor zresolvowany w tej samej lokalizacji fizycznej
            val sharedAnchorNode = AnchorNode(engine, anchor)
            // Obaj użytkownicy widzą ten sam obiekt w tym samym miejscu
            arSceneView.scene.addChild(sharedAnchorNode)
        }
    }
}

// Krok 3: synchronizacja stanu obiektów - Photon/Mirror/Normcore
// Photon Fusion: RPCs + State Synchronization per NetworkObject
```

Photon Fusion 2
Dedicated servers, relay servers. Client-State Prediction + Server Reconciliation. Up to 100 graczy per session. WebGL + Android/iOS.

Normcore (Normal Studio)
Abstracted networking: automatyczny state sync per GameObject. Rooms: join/leave. Voice: Normcore Voice. Prosty API dla XR.

Geospatial Multiplayer
ARCore Geospatial API: GPS + VPS (Visual Positioning System z Street View). Outdoor city-scale shared AR. Pokémon GO backend.

Metaverse i Standardy
Open Metaverse Standard (Khronos): OpenXR + OpenUSD + glTF + OpenVDB. Interoperability: przenoś awatary między światami.

Testowanie i CI/CD dla Aplikacji XR: ARCore Playback, XCTest

ARCore Dataset Recording

ARCoreApk.createRecordingConfig(outputPath). Nagrywaj sesje AR do pliku .mp4 + metadata. Odtwarzaj na emulatorze lub innym telefonie. Reprodukowalne testy bez fizycznego środowiska.

ARKit RecordReplayKit

ARWorldTrackingConfiguration + ARCapture. Eksportuj .arexperience. Testowanie w Xcode Simulator (ograniczone). Apple ARKit Testing: XCTest + ARSCNView mockowanie.

Unity Test Framework XR

Unity Test Runner: Edit Mode + Play Mode tests. XR Simulation (Unity 2023): wirtualne środowisko AR. XR Interaction Toolkit Test Helpers. GitHub Actions: Ubuntu + GPU worker.

Firebase Test Lab XR

Google: fizyczne urządzenia w chmurze. Espresso testy na urządzeniach z ARCore. Remote testing Quest: Sideload + monkey test. Robo test: automatyczna eksploracja UI.

XR Accessibility Testing

W3C XAUR (XR Accessibility User Requirements). Kontrast, wielkość elementów interaktywnych, alternatywy dla hand tracking. VR Motion Comfort Rating (komfort ruchu).

```
fun startRecording(session: Session, path: String)
{ session.startRecording(RecordingConfig(session).apply
{ mp4DatasetFilePath = path autoStopOnPause = true })}
fun stopRecording(session: Session) = session.stopRecording()
```

```
import XCTestimport ARKit
final class ARTests: XCTestCase { func testPlaneDetectionConfig() throws
{ guard ARWorldTrackingConfiguration.isSupported else { throw
XCTSkip() } let config = ARWorldTrackingConfiguration()
config.planeDetection = [.horizontal, .vertical]
XCTAssertTrue(config.planeDetection.contains(.horizontal))
XCTAssertTrue(config.planeDetection.contains(.vertical)) }}
```

Zastosowania XR: medycyna, edukacja, przemysł i rozrywka

Medycyna

Chirurgia wspomagana AR (Proximie). Symulacje 3D anatomii (Complete Anatomy). Rehabilitacja VR bólu (SnowWorld). MRI AR overlay.

Edukacja

Wirtualne laboratoria (Labster). AR podręczniki (Mondly). Google Expeditions: 900+ wirtualnych wycieczek. Mixed Reality Lab.

Przemysł 4.0

Boeing: montaż kabli drut wg AR overlay - 25% szybciej. Microsoft HoloLens Dynamics 365 Guides. Remote Assist z ekspertem.

Rozrywka

Beat Saber 4M+ egz. Pokémon GO $6B revenue. Half-Life: Alyx - AAA VR. PSVR2 biblioteka tytułów Sony exclusive.

e-Commerce

IKEA Place: meble AR w domu. Amazon AR View. Sephora Virtual Artist: make-up AR. Ocado: magazyn VR szkolenia.

Urbanistyka

Niantic Lightship ARDK: city-scale AR maps. Digital twins miast (NVIDIA Omniverse). AR Navigation (Google Maps Live View).

Wojskowość

Microsoft IVAS (HoloLens) US Army $22B. Symulatory pilotów VR F-35. AR HUD w hełmach. Tactical battlefield overlay.

? Maintenance

PTC Vuforia: AR instrukcje serwisu maszyn. Scope AR Remote: zdalne wsparcie z AR overlay. GE Aviation: AR turbine repair.

Przyszłość XR 2025–30: AI Phone XR, Neural Interface i Spatial Web

Snapdragon XR2 Gen3 (2025)

100+ TOPS NPU, WiFi 7, 5G mmWave. Standalone 8K per eye VR. AI-powered scene understanding on-device. Meta Quest 4 / PICO 5.

Neural Interface

Neuralink N1 chip: 1024 elektrody, BCI. Meta: EMG opaska rąk (sygnały mięśniowe) → gesture input. Myoelectric gloves mainstream 2027.

Light Field Display

Koniec z vergence-accommodation conflict. Focal Planes variable: 2 warstwy OLED + ML wypełnienie. Google CTRL Labs lensless display 2026.

Spatial Web (3D Internet)

Open Metaverse Foundation: WebXR + OpenUSD + glTF. Każda strona www = przestrzeń 3D. Spatial commerce: try before buy. Apple Vision Pro Safari.

AI Avatar + Holograms

Photorealistic 3D hologram rozmówcy w AR glasses. Codec Avatars (Meta): neural rendering z 170 kamer. Real-time deepfake detector w HMD.

Smart Contact Lens (2028+)

Mojo Vision: AR w soczewce kontaktowej. 14,000ppi micro-LED. Zasilanie bezprzewodowe. Sanity check: regulatory FDA Class III. Samsung patent.

Podsumowanie

ARCore + ARKit

Natywne platformy AR: plane detection, anchors, depth, lighting. Google i Apple - dwie ścieżki, AR Foundation jako unifikacja.

OpenXR Standard

Jeden API dla wszystkich platform XR. Action system, extensions, runtime. Przyszłość przemysłu: fragmentacja kończy się.

Rendering XR

ATW, foveated rendering, multiview jest kluczowe dla płynności. MTP < 20ms obowiązkowe. Frame pacing ważniejszy niż peak FPS.

AI + XR synnergia

Scene semantics, on-device LLM, Gaussian Splatting. AI przestaje być dodatkiem, a staje się fundamentem spatial computing.

Kluczowe wnioski

- XR = heterogeneous stack: sensor fusion + SLAM + rendering + networking + AI - każda warstwa ma znaczenie dla programisty
- ARCore (Android) i ARKit (iOS) to dojrzałe, produkcyjne platformy - pierwsze kroki AR wymagają < 50 linii kodu
- OpenXR unifikuje przemysł - nauka jednego API = wsparcie Quest, Valve, PSVR, HoloLens i kolejnych
- On-device AI (NPU 80+ TOPS) zmienia XR: scene understanding, AI avatary i LLM-powered asystenci spatial w 2025