LABORATORIUM 4
Rozumienie sceny i mieszanie rzeczywistości

Depth API • okluzja • light estimation • segmentacja • wiarygodność mieszania

Problem inżynierski

Jak zintegrować model wirtualny z rzeczywistością poprzez detektowanie głębi, okluzję i estymację oświetlenia? Jak zmierzyć, czy model wygląda wiarygodnie w scenie - czy jest prawidłowo zaciemniany, zokludowany i oświetlany?

Parametry laboratorium

Parametr	Wartość
Tryb pracy	pary; jeden telefon Android na parę, testy w pięciu warunkach oświetleniowych
Rezultat	aktywna Depth API, okluzja, pomiar błędu oświetlenia, scenariusz operacyjny
Oddanie	commit + evidence/lab04.md; bez tradycyjnego sprawozdania
Wersja	1.1.3 • 10 września 2026 r.

System oceniania

Ocena wynika z osiągnięcia kolejnych checkpointów. Każdy checkpoint weryfikuje prowadzący w ciągu maksymalnie 60 sekund.

Ocena	Główne wymagania
3.0	Depth API + okluzja + light estimation + wiarygodność
3.5	Segmentacja (osoba vs tło), okluzja selektywna
4.0	Pomiar błędu oświetlenia (5 warunków), analiza głębi
4.5	Porównanie: Depth API vs raycast, wpływ na wiarygodność
5.0	Dynamiczne teksturowanie, eksploracja granic Depth API

Walidacja: Numer indeksu i wariant diagnostyczny

Każdy student wpisuje numer indeksu. Wariant błędu wynika ze wzoru:

wariantDiagnostyczny = (numer_indeksu_1 + numer_indeksu_2) mod 5

Wynik: wariant A, B, C, D lub E (każda para ma inny błąd do naprawienia)

Wymagania wstępne

Przed zajęciami

- Zaliczone Laboratorium 3 albo działająca scena AR z raycastem i kotwicami
- Telefon Android z obsługą Depth API (ARCore 1.22+; lista: https://developers.google.com/ar/devices)
- Włączone opcje programistyczne, debugowanie USB
- Unity z AR Foundation + ARCore XR Plugin w wersji z Lab 3
- Brak czerwonych błędów w Console projektu
- Podstawy fotometrii: luminancja, temperatura barwowa (K), renderowanie fizycznie oparte

Pakiety wymagane w projekcie

Pakiet	Identyfikator	Rola
AR Foundation	com.unity.xr.arfoundation	Abstrakcja nad platformami AR
ARCore XR Plugin	com.unity.xr.arcore	Depth API, segmentacja, oświetlenie
XR Plug-in Management	com.unity.xr.management	Zarządzanie runtime'ami
XR Interaction Toolkit	com.unity.xr.interaction.toolkit	UI i interakcje
Input System	com.unity.inputsystem	Obsługa ekranu i gestów

UWAGA: Depth API wymaga specjalnych uprawnień na Androidzie (READ_EXTERNAL_STORAGE) i może nie być dostępna na wszystkich urządzeniach.

Bezpieczeństwo i organizacja testów

- Testy oświetleniowe w pięciu warunkach: słoneczne, sztuczne ciepłe, sztuczne zimne, mieszane, półcień. Oznacz każdy warunek datą i fotografią.
- Nie zapisuj obrazu z kamery ani danych osobowych. Wyniki pomiarowe - tylko numery par.
- Jeden pomiar = trzydzieści sekund przy stałym oświetleniu i pozycji urządzenia.
- Użytkownik patrzący na ekran - druga osoba kontroluje oświetlenie i zapisuje wyniki.
- Telefon powinien być ustawiony stabilnie (statyw albo rękaw na ramieniu).
- Po zajęciach właściciel telefonu może usunąć aplikację i dane testowe.

Podstawy teoretyczne

4.1. Depth API (ARCore)

Depth API estymuje głębię każdego piksela na podstawie obrazu z kamery (monocular depth estimation). Wynik to depth map o rozdzielczości 180×240 (niska precyzja, wysoka szybkość) lub wyższej, w zależności od urządzenia. ARDepthManager udostępnia dane w scenie ARKit/ARCore.

Zastosowania: okluzja wirtualnych obiektów za rzeczywistymi przeszkodami, segmentacja osób, analiza geometrii sceny.

4.2. Okluzja (Occlusion)

Model wirtualny powinien być zasłaniany (occluded) przez obiekty rzeczywiste, które są bliżej kamery. Bez okluzji model wygląda, jakby pływał ponad rzeczywistością. Depth API umożliwia takie renderowanie: pixel shader sprawdza depth map i decyduje, czy pixel należy do obiektu rzeczywistego czy wirtualnego.

4.3. Light Estimation

ARLightEstimate estymuje światło w scenie: intensywność, kolor (temperatura barwowa), kierunkowość. Model wirtualny powinien być oświetlony w taki sam sposób co rzeczywistość. Niedopasowanie oświetlenia powoduje, że model wygląda "sztucznie".

Parametry Light Estimation:

- Ambient Intensity: jasność otoczenia (0–1)
- Color Correction: multiplikator RGB (białość światła)
- Directional Light Estimate: kierunek i siła głównego światła
- Reflection Probes: tekstura otoczenia dla odbić

4.4. Wiarygodność mieszania (Credibility)

Użytkownik ocenia wiarygodność wizualną. Parametry:

1. Pozycja: czy model znajduje się w rzeczywistym miejscu? (zależy od lab3)

2. Okluzja: czy rzeczywiste obiekty zasłaniają model?
3. Oświetlenie: czy model jest oświetlony jak otoczenie?
4. Cień: czy model rzuca cień?
5. Kolory: czy kolory są realistyczne?

Wiarygodność = średnia z pięciu parametrów, ocena 1–5.

Plan pracy

Etap	Dowód ukończenia
Miniquiz i kontrola Depth API	Odpowiedzi + Depth API dostępna na urządzeniu
Repozytorium, konfiguracja ARDepthManager	Scena uruchamia się, depth map widoczna
Implementacja okluzji	Model jest okludowany za rzeczywistymi przeszkodami
Konfiguracja Light Estimation	Model zmienia oświetlenie razem ze sceną
Testy wiarygodności (5 warunków oświetleniowych)	Pięć pomiarów z różnymi warunkami
Diagnoza błędu	Objaw, hipoteza, test, przyczyna, commit
Oddanie i exit ticket	Evidence/lab04.md + odpowiedź

Checkpoint 3.0

Wymagania do zaliczenia:

✓ Aplikacja uruchamia się bez błędów krytycznych
✓ Depth API jest dostępna i zwraca depth map (zweryfikowano logiem)
✓ Model jest okluzowany: rzeczywiste przeszkody zasłaniają model wirtualny
✓ Light Estimation zmienia oświetlenie modelu razem ze sceną
✓ Przeprowadzono pięć testów wiarygodności w różnych warunkach oświetleniowych
✓ Dla każdego testu: zapisano warunek, pomiar wiarygodności (1–5), uwagi
✓ Diagnoza: objaw, dwie hipotezy, test rozstrzygający
✓ Evidence/lab04.md zawiera: numery indeksów, wyniki testów, wariant diagnostyczny

Część A. Uruchomienie projektu z Depth API

A.1. Miniquiz

1. Wyjaśnij, dlaczego Depth API jest estymacją, a nie rzeczywistym sensoremn
2. Czym różni się okluzja zależna od Depth API od okluzji opartej na sortowaniu Z?
3. Kiedy Light Estimation zwróci null i co powinien zrobić shader?
4. Podaj parametr, który NAJBARDZIEJ wpływa na wiarygodność mieszania: pozycja, okluzja czy oświetlenie?
5. Na jakim urządzeniu Depth API nie będzie dostępna?

A.2. Repozytorium i konfiguracja

1. git switch lab04-start i utwórz gałąź team-<nr>/lab04-<nazwiska>
2. Skopiuj scenę z Lab 3: Assets/Scenes/Lab04_SceneUnderstanding.unity
3. Dodaj ARDepthManager do sceny (jeśli nie istnieje)
4. Konfiguracja: Edit → Project Settings → ARCore → Enable Depth API

A.3. Weryfikacja Depth API

1. Skompiluj i uruchom na telefonie
2. Dodaj prosty shader, który wyświetla depth map w czarno-białym formacie

3. Jeśli depth map jest widoczna: Depth API działa
4. Log powinien zawierać: „ARDepthManager: depth data available"

PUNKT KONTROLNY A: Aplikacja uruchamia się, Depth API zwraca dane, depth map jest widoczna na ekranie.

Część B. Implementacja okluzji

B.1. Shader z okluzją

```
// Shader z okluzją na podstawie Depth APIShader "XR/OccludedModel"{
   SubShader
   {
      Pass
      {
         CGPROGRAM
         #pragma vertex vert
         #pragma fragment frag
                  sampler2D _DepthTexture;
         float4x4 _ViewProjectionMatrix;
                  float4 vert(float4 vertex : POSITION) : SV_POSITION
         {
            return UnityObjectToClipPos(vertex);
         }
                  fixed4 frag(float4 screenPos : VPOS) : SV_Target
         {
            // Pobierz głębię z mapy
            float2 uv = screenPos.xy / _ScreenParams.xy;
            float sceneDepth = tex2D(_DepthTexture, uv).r;
                        // Porównaj z głębią piksela modelu
            float fragmentDepth = screenPos.z;
                        // Jeśli scena jest bliżej: wyłącz pixel (okluzja)
            if (sceneDepth < fragmentDepth)
               discard;
                        return fixed4(1, 0, 0, 1); // Czerwony model
         }
         ENDCG
      }
   }
}
```

1. Zastosuj shader do materiału modelu
2. Uruchom: model powinien być okludowany przez rzeczywiste przeszkody
3. Test: przesłoń telefon dłonią - model powinien zniknąć

PUNKT KONTROLNY B: Model jest okludowany za rzeczywistymi przeszkodami. Przesłonięcie dłonią lub przedmiotem powoduje zniknięcie modelu.

Część C. Konfiguracja Light Estimation

C.1. Pobranie Light Estimate

```
using UnityEngine;using UnityEngine.XR.ARFoundation;
public class LightingController : MonoBehaviour {
   ARCameraManager cameraManager;
   Light directionalLight;
   int pairIndexSum = 0; // WPISZ: suma indeksów
      void Start() {
      cameraManager = GetComponent<ARCameraManager>();
      directionalLight = GetComponentInChildren<Light>();
   }
      void Update() {
      if (cameraManager.TryGetLatestFrame(
         ARCameraFrameEventArgs.LightEstimateUpdatedFlag,
         out ARCameraAcquisitionFrame frame)) {
                  var lightEstimate = frame.lightEstimate;
         if (lightEstimate != null) {
            // Intensywność
            directionalLight.intensity =
               lightEstimate.averageBrightness;
                     // Kolor
            directionalLight.color =
               lightEstimate.colorCorrection;
                     Debug.Log($"LIGHT brightness={lightEstimate.averageBrightness:F2} " +
                $"color={lightEstimate.colorCorrection} seed={pairIndexSum}");
         }
      }
   }
}
```

1. Dodaj skrypt do obiektu z directionalLight
2. Wpisz sumę numerów indeksów
3. Uruchom: oświetlenie modelu zmienia się wraz ze zmianą oświetlenia sceny
4. Test: przejdź między światłem słonecznym a sztucznym - model powinien się zaciemniać/rozjaśniać

PUNKT KONTROLNY C: Light Estimation zmienia intensywność i kolor światła na modelu. Zmiana oświetlenia sceny skutkuje zmianą oświetlenia modelu.

Część D. Testy wiarygodności (5 warunków oświetleniowych)

D.1. Protokół testowania (25 minut)

Pięć warunków oświetleniowych:

Warunek 1: Słoneczne (outdoor, promieniowanie bezpośrednie)
Warunek 2: Sztuczne ciepłe (biuro, lampa ~3000K)
Warunek 3: Sztuczne zimne (neon, lampa ~6500K)
Warunek 4: Mieszane (część słoneczna, część w cieniu)
Warunek 5: Półcień (brak bezpośredniego światła)

Dla każdego warunku:
1. Ustaw urządzenie stabilnie (statyw lub rękaw)
2. Czekaj 10 sekund, aby Light Estimation się ustabilizował
3. Przez 30 sekund obserwuj model - czy wygląda wiarygodnie?
4. Oceń każdy parametr (1–5): pozycja, okluzja, oświetlenie, cień, kolory
5. Średnia = wiarygodność całkowita dla warunku
6. Zapisz: warunek, temperatura K (jeśli znana), wiarygodność, uwagi

// Protokół testowania wiarygodności
Warunek | Temperatura | Pozycja | Okluzja | Oświetlenie | Cień | Kolory | ŚREDNIA
Słoneczne | 5500 K | _ | _ | _ | _ | _ | _
Sztuczne CW | 3000 K | _ | _ | _ | _ | _ | _
Sztuczne ZM | 6500 K | _ | _ | _ | _ | _ | _
Mieszane | 4500 K | _ | _ | _ | _ | _ | _
Półcień | 5500 K | _ | _ | _ | _ | _ | _

PUNKT KONTROLNY D: Pięć testów wiarygodności przeprowadzonych, każdy z oceną każdego parametru, mediana obliczona.

Część E. Zadanie diagnostyczne

Wyznaczenie wariantu:

$$\text{wariant} = (\text{numer_indeksu_1} + \text{numer_indeksu_2}) \bmod 5$$

Wariant	Objaw	Hipotezy
A	Model zawsze jest całkowicie oświetlony (nie reaguje na zmianę światła)	Light Estimation nie działa | Shader nie czyta intensywności | debug.log nie pokazuje zmian
B	Model nie jest okludowany (prześwituje przez rzeczywiste obiekty)	Depth API nie zwraca danych | Shader nic nie sprawdza | depth texture nie przypisana
C	Model ma nieprawidłowy kolor (zawsze fioletowy zamiast szarego)	Błędna interpretacja colorCorrection | Shader mnożył zamiast zamieniać
D	Wiarygodność spada drastycznie w warunku mieszanym (odbicia nie działają)	Light Estimate zwraca tylko kierunkowe światło | Brak Environment Probe | błąd w estymacji odbić
E	Model dryfuje vertycznie (obniża się podczas testu)	XR Origin ma niejednostkową skalę | zmiana Layer Mask | błąd kotwicy sesji

1. Opisz objaw mierzalnie (np. „Wiarygodność w warunku 4 = 1.5, podczas gdy w warunku 1 = 4.2")
2. Sformułuj dwie hipotezy z różnymi przewidywaniami
3. Wykonaj test: sprawdź log, zmieniaj parametry, wizualizuj dane
4. Popraw jedną rzecz; powtórz test
5. git commit -m "fix(lab04): <przyczyna>"

PUNKT KONTROLNY E: Diagnoza zawiera objaw, dwie hipotezy, test, przyczynę, wynik. Commit opisuje przyczynę.

Checkpoint 3.5

Wymagania dodatkowe (oprócz checkpointu 3.0):

✓ Segmentacja: model powinien być okludowany wyłącznie przez osoby (face/body detection)
✓ Tło i inne obiekty nie okluzują modelu
✓ Dodatkowe testy wiarygodności: 5 warunków + osoba przeszłaniająca model
✓ Pomiar: czy wiarygodność spada, gdy osoba pojawia się przed modelem?
✓ Interpretacja: które elementy (pozycja, okluzja, oświetlenie) najbardziej wpływają na ogólną ocenę

Weryfikacja (prowadzący, max 60s):

✓ Checkpoint 3.0: wszystkie kryteria

✓ Segmentacja: model okludowany przez osobę, ale nie przez tło

✓ Pomiary wiarygodności z osobą i bez

✓ Analiza wpływu poszczególnych parametrów

Zaliczenie: 3/4 dodatkowe kryteria (+ Checkpoint 3.0)

Checkpoint 4.0

Wymagania dodatkowe (oprócz checkpointów 3.0 + 3.5):

✓ Pomiar błędu oświetlenia: różnica między estymowanym a obserwowanym oświetleniem (luminancja)
✓ Dla każdego z pięciu warunków: pomiar luksów (jeśli dostępny luksomierz)
✓ Tabela: warunek | luksomierz [lux] | Light Estimation [intensity] | błąd [%]
✓ Analiza dokładności Depth API: porównanie estymowanej głębi z mierzoną
✓ Korelacja: czy lepsze oświetlenie daje lepszą okluzję?

Weryfikacja (prowadzący, max 60s):

✓ Checkpointy 3.0 + 3.5: wszystkie kryteria

✓ Tabela błędu oświetlenia (5 warunków)

✓ Pomiar głębi: dokładność Depth API

✓ Korelacja oświetlenie ↔ okluzja

Zaliczenie: 3/4 dodatkowe kryteria (+ poprzednie). Ocena finalna: 4.0

Checkpoint 4.5

Wymagania dodatkowe (oprócz checkpointów 3.0 + 4.0):

✓ Porównanie dwóch metod okluzji: Depth API vs raycast (bez Depth API)
✓ Dla każdej metody: pomiary wiarygodności w 5 warunkach
✓ Analiza: która metoda bardziej poprawia wiarygodność?
✓ Wpływ Depth API na wydajność: pomiar FPS bez i z Depth API

✓ Propozycja optymalizacji: kiedy Depth API, kiedy raycast?

Weryfikacja (prowadzący, max 60s)

:✓ Checkpointy 3.0–4.0: wszystkie kryteria

✓ Porównanie: Depth API vs raycast (tabela wiarygodności)

✓ Pomiar wydajności: FPS dla obu metod

✓ Analiza trade-off: dokładność ↔ wydajność

Zaliczenie: 3/4 dodatkowe kryteria (+ poprzednie). Ocena finalna: 4.5

Checkpoint 5.0

Wymagania dodatkowe (oprócz checkpointów 3.0 + 4.0 + 4.5):

✓ Dynamiczne teksturowanie na podstawie Depth API: model zmienia teksturę w zależności od głębi otoczenia

✓ Eksploracja granic Depth API: testowanie w warunkach ekstremalnych (bardzo słabe oświetlenie, lustro, okno)

✓ Pomiary: gdzie Depth API zawodzi? Gdzie jest szczególnie dokładna?

✓ Zaproponowana architektura hybrydowa: Depth API + raycast + Light Estimation

✓ Wnioski: limitacje, przyszłe kierunki, porównanie z innymi platformami (iOS ARKit, Magic Leap)

Weryfikacja (prowadzący, max 60s):

✓ Checkpointy 3.0–4.5: wszystkie kryteria

✓ Dynamiczne teksturowanie: implementacja i efekt wizualny

✓ Testy graniczne: dane z ekstremalnych warunków

✓ Architektura hybrydowa: opis + porównanie metod

Zaliczenie: 3/4 dodatkowe kryteria (+ poprzednie). Ocena finalna: 5.0

Oddanie

```
# Laboratorium 4 - <para / nazwiska>
## Identyfikacja pary
- Numer indeksu student 1: _____
- Numer indeksu student 2: _____
- Suma numerów: _____
- Przypisany wariant diagnostyczny: ___ (A/B/C/D/E)

## Wersje i konfiguracja
- Unity / AR Foundation / ARCore: ___ / ___ / ___
- Depth API dostępna: TAK / NIE
- Model telefonu Android: _____

## Implementacja- Depth API aktywna: TAK / NIE
```

- Okluzja zaimplementowana: TAK / NIE
- Light Estimation zmienia oświetlenie: TAK / NIE
- Shader użyty: [nazwa i opis]

Testy wiarygodności (5 warunków)
Warunek	Temp. [K]	Pozycja	Okluzja	Oświetlenie	Cień	Kolory	ŚREDNIA
Słoneczne	5500	__	__	__	__	__	__
Sztuczne CW	3000	__	__	__	__	__	__
Sztuczne ZM	6500	__	__	__	__	__	__
Mieszane	4500	__	__	__	__	__	__
Półcień	5500	__	__	__	__	__	__
- Średnia wiarygodności: ___

Błąd oświetlenia (jeśli dostępny luksomierz)
| Warunek | Luksomierz [lux] | Light Intensity | Błąd [%] ||---|---|---|---|
| Słoneczne | __ | __ | __ |
| Sztuczne CW | __ | __ | __
|| ... | ... | ... | ... |

Diagnoza (wariant ___)
- Objaw: ___
- Hipoteza 1: ___
- Hipoteza 2: ___
- Test rozstrzygający: ___
- Przyczyna: ___
- Wynik przed / po: ___ / ___

Wnioski
1. ___
2. ___
3. ___
![Model z okluzją i oświetleniem](img/lab04-rendering.png)

Każda osoba w parze musi mieć co najmniej jeden merytoryczny commit.

Lista końcowa studenta - przed oddaniem

☐ Aplikacja uruchamia się na Android bez błędów krytycznych.
☐ Depth API jest dostępna i zwraca depth map (zweryfikowano logiem).
☐ Shader implementuje okluzję: rzeczywiste przeszkody zasłaniają model.
☐ Light Estimation zmienia oświetlenie modelu razem ze sceną.

☐ Przeprowadzono pięć testów wiarygodności w różnych warunkach oświetleniowych.
☐ Każdy test zawiera ocenę pięciu parametrów (1–5 dla każdego).
☐ Mediana wiarygodności obliczona.
☐ Diagnoza zawiera objaw, dwie hipotezy, test, przyczynę, wynik.
☐ Evidence/lab04.md i zrzut znajdują się w repozytorium.
☐ Każda osoba w parze wykonała co najmniej jeden merytoryczny commit.
☐ W skrypcie LightingController wpisałem poprawnie sumę numerów indeksów pary.

Pytania kontrolne

1. Dlaczego Depth API jest estymacją, a nie rzeczywistym sensorem
2. Czym różni się okluzja zależna od Depth API od okluzji opartej na sortowaniu Z?
3. Kiedy Light Estimation zwróci null i co powinien zrobić shader?
4. Który parametr NAJBARDZIEJ wpływa na wiarygodność mieszania?
5. Na którym Androidzie Depth API nie będzie dostępna?
6. Jak shader wiedz, czy pixel piksela należy do obiektu rzeczywistego czy wirtualnego?
7. Czy Light Estimation zwiększa czy zmniejsza wydajność aplikacji?

Exit ticket

Jedno zdanie: Które ograniczenie Depth API (dokładność, opóźnienie, dostępność urządzenia) będzie najpoważniejszą przeszkodą w Twoim przyszłym projekcie AR i jak byś to chciał rozwiązać?