POLITECHNIKA RZESZOWSKA
im. IGNACEGO ŁUKASIEWICZA

KATEDRA INFORMATYKI I AUTOMATYKI
POLITECHNIKI RZESZOWSKIEJ

Renderowanie i wydajność aplikacji XR

WiRR · Wykład 06 · Automatyka i Robotyka · 3 ECTS · 2026/2027 · v1.3.8

dr inż. Mateusz Pomianek

Katedra Informatyki i Automatyki, Politechnika Rzeszowska

27.10.2026

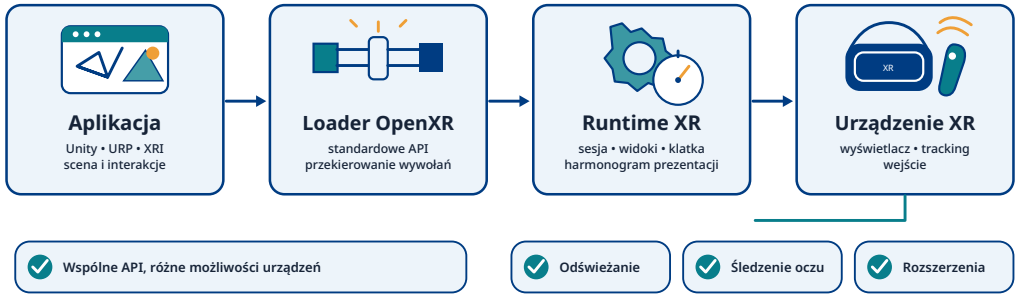
Cel: znaleźć koszt klatki przed obniżaniem jakości



Porównaj **tę samą scenę XR** przed i po jednej zmianie. Nie obniżaj jakości, dopóki nie rozpoznasz wąskiego gardła.

- Czy termin klatki przekracza CPU, GPU czy synchronizacja?
- Jak wyglądają rozkłady czasów po rozgrzewce?
- Czy użytkownik nadal odczytuje etykiety i wykonuje zadanie?

OpenXR: wspólne API



OpenXR standaryzuje interfejs aplikacji z runtime XR, ale nie ujednolica możliwości urządzeń.

`Aplikacja → OpenXR → Runtime → Zestaw XR`

Sprawdzaj osobno: obsługiwane rozszerzenia, tryb renderowania, dostępność śledzenia oraz docelową częstotliwość. Wspólne API nie oznacza jednakowego FPS.

Unity, XRI i AR Foundation



- **Unity + URP:** logika sceny, shadery i rendering.
- **XR Plug-in Management + OpenXR:** integracja z runtime i widokami XR.
- **XR Interaction Toolkit:** wskazywanie, wybór, chwyt.
- **AR Foundation:** płaszczyzny, kotwice i śledzenie, jeżeli obsługuje je provider.

Gdy spada FPS, najpierw profiluj rendering, a nie wymieniaj komponentów interakcji.

Aplikacja, runtime i kompozytor: kto odpowiada za obraz?

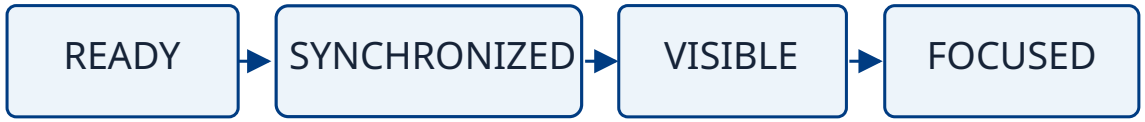


Reprojekcja nie generuje nowego stanu sceny

Aplikacja renderuje nowe widoki sceny. **Runtime** planuje pracę z urządzeniem. **Kompozytor** prezentuje warstwy, niekiedy korygując je na podstawie nowszej pozy.

Stabilna prezentacja po reprojekcji **nie oznacza**, że aplikacja wygenerowała nową klatkę. Rozróżniaj licznik klatek aplikacji i zdarzeń prezentacji.

Sesja OpenXR: działanie nie oznacza widoczności

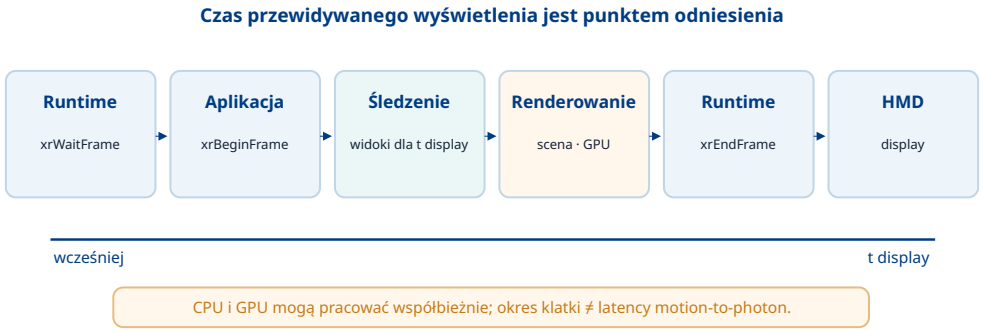


STOPPING: zakończenie sesji

Sesja `SYNCHRONIZED` może nie być jeszcze widoczna. `VISIBLE` oznacza możliwość prezentacji, a `FOCUSED` również fokus wejścia.

Przejścia stanów zależą od zdarzeń runtime. **Obsługa wejścia wymaga odpowiedniego stanu i fokusu**; samo utworzenie sesji nie wystarcza.

Pętla klatki odnosi się do czasu wyświetlenia



OpenXR przekazuje `predictedDisplayTime` i flagę `shouldRender` w strukturze `XrFrameState`.

`xrWaitFrame` → `xrBeginFrame` → `xrLocateViews` → `render` → `xrEndFrame`

Pozy widoków wyznaczamy dla przewidywanego czasu prezentacji. Gdy `shouldRender == XR_FALSE`, pomijamy ciężkie renderowanie i przekazujemy puste warstwy; **nie pomijamy synchronizacji pętli**. Unity zarządza tą sekwencją przez provider.

Motion-to-photon: FPS nie wyczerpuje opisu opóźnienia



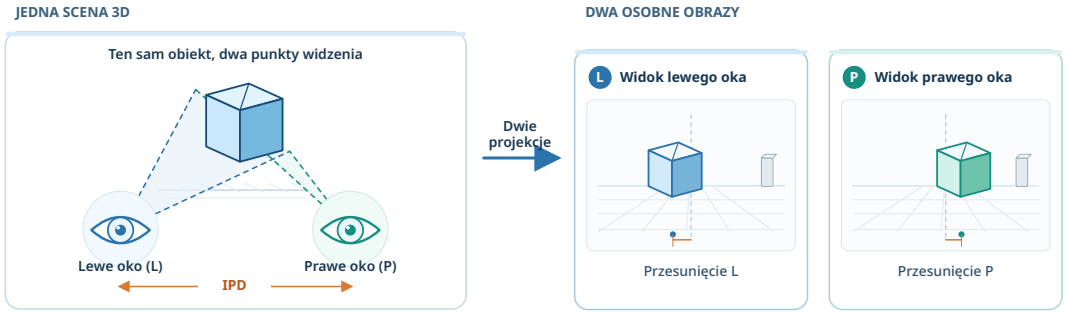
Cały łańcuch ≠ 1000 / FPS

Motion-to-photon obejmuje drogę od ruchu głowy do zmiany światła docierającego z wyświetlacza.

`Sensor → estymacja pozy → aplikacja → GPU → kompozytor → wyświetlacz`

`90 Hz = 11,11 ms` oznacza **okres odświeżania**, a nie całkowitą latencję MTP. Reprojekcja ogranicza niektóre skutki opóźnienia, lecz nie dostarcza brakujących nowych zmian sceny.

Rendering stereoskopowy



Dwa oczy wymagają **różnych pozycji widoku i projekcji**. Prawidłowa geometria stereoskopowa jest warunkiem jakości i komfortu.

Rendering obu oczu nie musi kosztować dokładnie dwa razy tyle co pojedynczy widok: część pracy może być współdzielona przez silnik i GPU.

Single Pass Instanced i multiview: mniej powtarzanej pracy

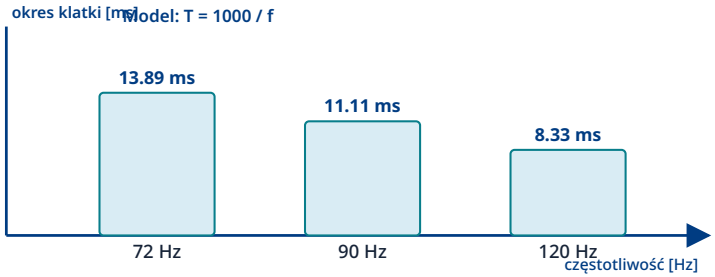


Multi-pass: osobne przejścia dla dwóch widoków. **Single Pass Instanced / multiview:** ograniczenie części powtarzalnej pracy związanej z przygotowaniem renderowania.

Oszczędność zależy od platformy, API grafiki i shaderów; **oba obrazy nadal wymagają przetwarzania pikseli**.

Unity: `Project Settings → XR Plug-in Management → OpenXR → Render Mode`, jeżeli provider udostępnia tę opcję.

Budżet klatki



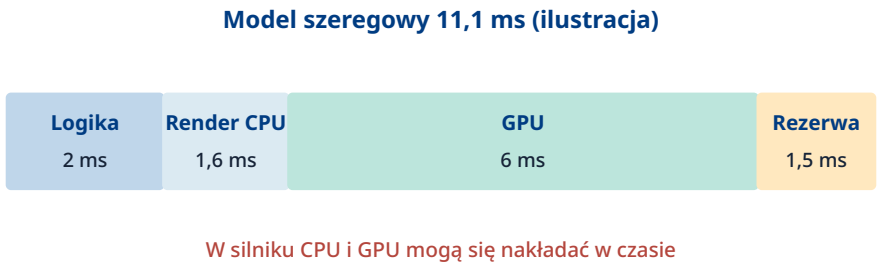
Czas jednego okresu wyświetlania wynosi

$$T = \frac{1}{f}, \qquad T_{\mathrm{ms}} = \frac{1000}{f_{\mathrm{Hz}}}.$$

- **72 Hz:** 13,89 ms
- **90 Hz:** 11,11 ms
- **120 Hz:** 8,33 ms

To **termin dostarczenia klatki**, nie wynik pomiaru CPU/GPU ani motion-to-photon.

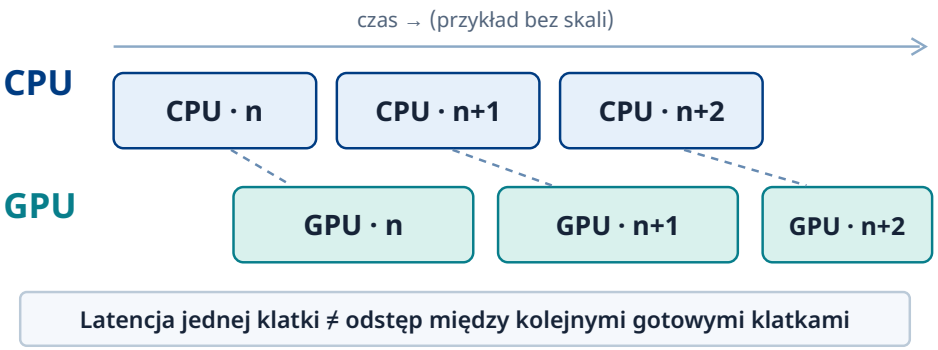
Jak interpretować budżet klatki



Przykład modelowy szeregowy dla 90 Hz: logika 2,0 ms + przygotowanie renderowania 1,6 ms + GPU 6,0 ms + rezerwa 1,5 ms = **11,1 ms**.

W silniku CPU i GPU pracują częściowo równolegle. **Nie sumuj czasów liczników pochodzących z równoległych kolejek**; kontroluj zakres i jednostkę każdego licznika.

CPU i GPU: równoległość zmienia rachunek

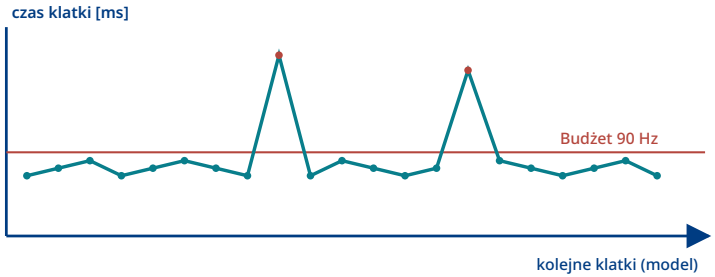


CPU może przygotowywać klatkę `n+1`, gdy GPU renderuje klatkę `n`. **Oba etapy nakładają się w czasie**, choć obsługują różne klatki.

Przepustowość mierzy tempo gotowych klatek, a **latencja** drogę jednej klatki przez potok. Czasów pracy dwóch równoległych kolejek nie sumujemy.

Gdy CPU czeka na GPU, badaj zależności i synchronizację, zamiast automatycznie obwiniać skrypty.

Jedna liczba FPS ukrywa niestabilność

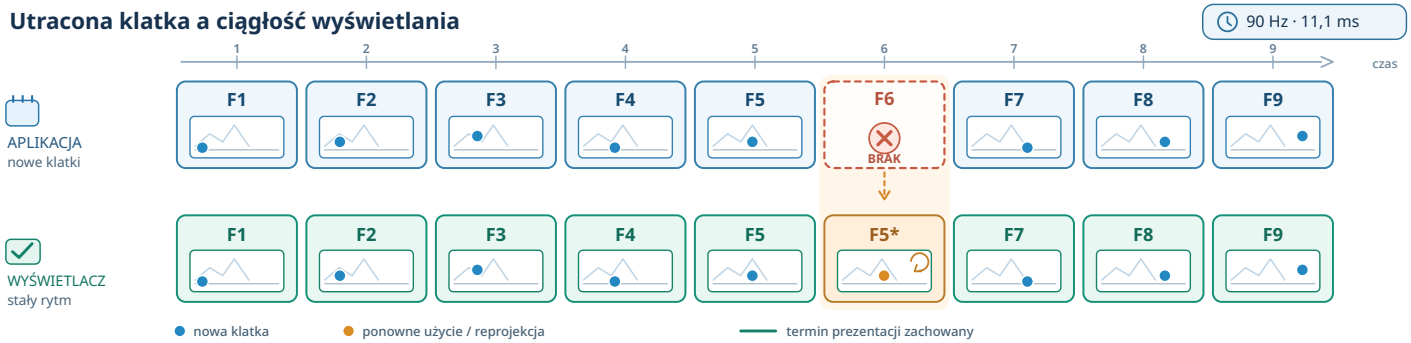


Dwie serie mogą mieć identyczną medianę, ale różne **p95** i liczbę spóźnionych klatek.

- **p95** to wartość, poniżej której mieści się około 95% próbek, a nie „95% FPS“.
- Podawaj **medianę, p95, n oraz odsetek przekroczeń** ustalonego progu.
- Oddziel licznik CPU/GPU od metryki **faktycznie pominiętych klatek** runtime.

Porównuj rozkłady dla **tej samej konfiguracji**, przy stałej częstotliwości i po rozgrzewce.

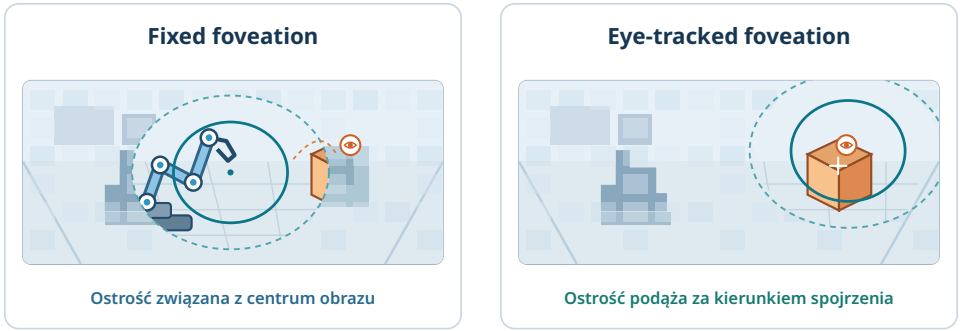
Jitter, zacięcia i pominięte klatki



Jitter oznacza nieregularność odstępów między klatkami; **stutter** to widoczne zacięcie związane z takim zakłóceniem.

Rozróżniaj: spóźnioną klatkę aplikacji, brak nowej klatki oraz prezentację obrazu skorygowanego przez runtime. Średni FPS nie identyfikuje tych zjawisk.

Foveated rendering



Fixed foveated rendering: obszar najwyższej jakości jest nieruchomy względem obrazu.

Eye-tracked foveated rendering: obszar ten przemieszcza się wraz z oszacowaną pozycją spojrzenia.

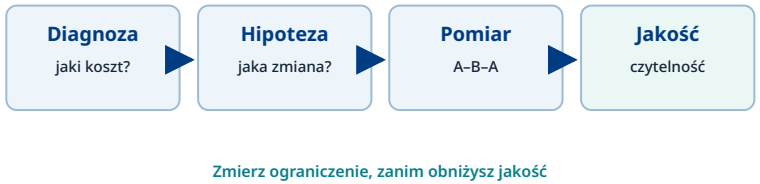
Porównuj artefakty w ruchu gałek ocznych i głowy; w trybie gaze-contingent dochodzą błędy kalibracji oraz opóźnienie śledzenia. **Oszczędności GPU trzeba zmierzyć**.

5 minut przerwy

Pytania do samodzielnego powtórzenia

1. Dlaczego przy 90 Hz dostępne jest 11,11 ms, lecz motion-to-photon może być inne?
2. Czym różnią się multi-pass, single-pass instanced i multiview?
3. Kiedy obniżenie rozdzielczości nie poprawi wydajności?
4. Jak przygotować wiarygodne porównanie A–B–A po optymalizacji?

Optymalizacja



Reguła eksperymentu: diagnoza → hipoteza → jedna zmiana → pomiar → kontrola jakości.

Przykładowa hipoteza: *przezroczyste nakładki zwiększają koszt pracy na pikselach*. Wtedy testuj ich usunięcie, zamiast redukować trójkąty w całej scenie.

Unity, warianty jednej sceny:

```
public void ShowBaseline() => Apply(true, true);
public void ShowLowDetail() => Apply(false, true);
```

Pełny demonstrator C#.

Dynamic resolution: kiedy obniżanie rozdzielczości pomaga?



Zmniejszenie liczby fragmentów może odciążyć GPU. Czas pracy CPU i zysk FPS zależą od wąskiego gardła.

Zmniejszenie obu wymiarów obrazu do **80%** oznacza **64% pierwotnej liczby pikseli**:

$$N'/N = 0,8^2 = 0,64.$$

Regulacja ręczna w URP: `XRSettings.renderViewportScale` (odczyt: `XRSettings.appliedRenderViewportScale`).

Regulacja automatyczna: kompatybilny runtime OpenXR i URP mogą dostosowywać skalę do obciążenia GPU.

Kontrola jakości: czytelność etykiet podczas ruchu.

LOD zmienia geometrię, nie znaczenie zadania



LOD: wybór uproszczonej siatki wraz z odległością lub wielkością projekcji.

Zmniejszenie liczby trójkątów pomaga tylko wtedy, gdy odciąża **rzeczywisty koszt ograniczający**. Element krytyczny dla zadania, np. dźwignia czy przycisk, musi pozostać rozpoznawalny.

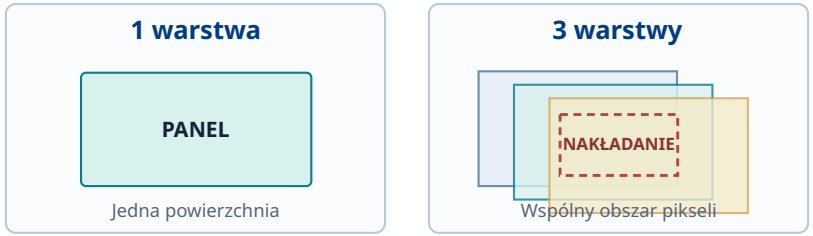
Culling i instancing rozwiązują inne problemy



- **Frustum culling:** nie renderuj obiektu poza bryłą widzenia.
- **Occlusion culling:** pomijaj zasłonięte obiekty.
- **GPU instancing:** ogranicz powtarzaną pracę przy zgodnych instancjach siatki i materiału.

W URP wynik instancingu zależy od shaderów i SRP Batcher. **Mierz liczbę draw calls i koszt CPU**, zamiast zakładać przyspieszenie.

Fill rate, overdraw i koszt przezroczystości



Overdraw to wielokrotne przetwarzanie fragmentów tego samego obszaru obrazu; na schemacie porównano jeden i trzy pokrywające się panele.

Przezroczysty UI może mieć mało trójkątów, a jednocześnie duży **koszt pikselowy**. Istotne są zasięg na ekranie, shader, blendowanie i odrzucanie fragmentów.

Eksperyment: zmieniaj wyłącznie liczbę przezroczystych nakładek i porównuj koszt GPU, jeśli platforma udostępnia odpowiedni licznik.

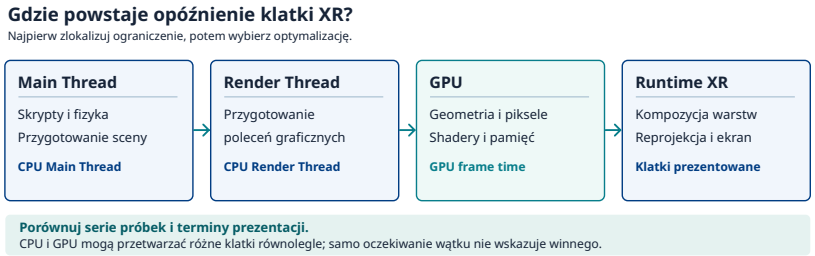
Kod: interpretacja przekroczenia budżetu

Okres 90 Hz: `11,11 ms` (przykład dla ustalonego trybu odświeżania).

```
float budgetMs = 1000f / targetHz;
bool cpuStageTooLong = mainThreadMs > budgetMs;
```

`mainThreadMs` mierzy **główny wątek CPU**. Warunek sygnalizuje przekroczenie progu przez ten etap, **nie dowodzi pominięcia klatki**. Zobacz rejestrator Unity.

Profiler: diagnoza przed zmianą

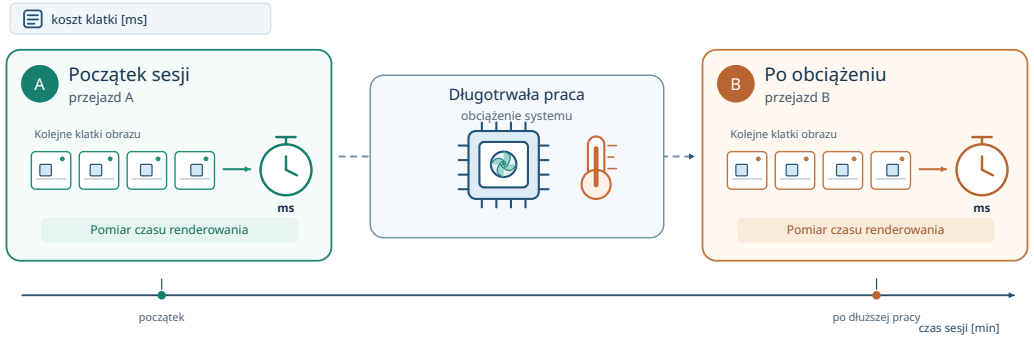


Diagnostyka: znajdź przekroczenia budżetu → Main Thread → Render Thread → GPU (jeśli dostępny) → oczekiwanie i prezentacja w runtime.

```
// Nanosekundy z ProfilerRecorder -> ms
double mainMs = recorder.LastValue / 1_000_000.0;
```

Wątek może czekać na GPU lub synchronizację. **Profiluj build na docelowym headsetcie** i porównuj liczniki aplikacji z metrykami runtime.

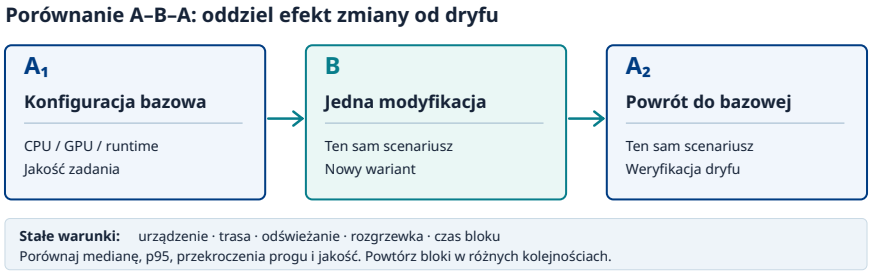
Termika i stabilność po dłuższej pracy



Pomiar tuż po uruchomieniu ≠ pomiar po długim obciążeniu. Sprzęt może ograniczać częstotliwości pracy z powodów termicznych lub energetycznych.

Porównaj identyczny przejazd na początku i później. Rejestruj warunki środowiska, czas testu, stan baterii i dostępne metryki termiczne. Bez nich nie przypisuj zmiany wydajności do temperatury.

Kontrolowane porównanie wydajności



A–B–A: bazowy → jedna modyfikacja → powrót do bazowego. Powrót pomaga wykryć dryf termiczny lub zmianę warunków.

- Ten sam build, urządzenie, trasa, częstotliwość, rozgrzewka.
- Porównywalne okna i **powtórzone bloki**, nie pojedynczy przebieg.
- Mediana, p95, przekroczenia progu, **metryki runtime** i jakość zadania.

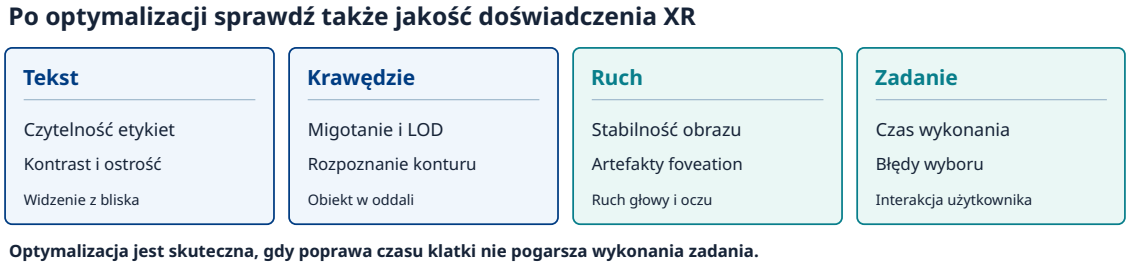
Samo A–B–A nie eliminuje efektów kolejności: w razie potrzeby zmieniaj kolejność bloków.

Wydajność i doświadczenie użytkownika

Przyspieszenie GPU o 2 ms ma wartość użytkową tylko wtedy, gdy **nie pogarsza wykonania zadania**.

Rozdziel wskaźniki **techniczne** (ms, p95, spóźnione klatki) oraz **funkcjonalne** (czas zadania, błędy wyboru, czytelność etykiet, komfort). Zmiana techniczna nie gwarantuje poprawy doświadczenia XR.

Jakość percepcyjna: co warto sprawdzać po optymalizacji?

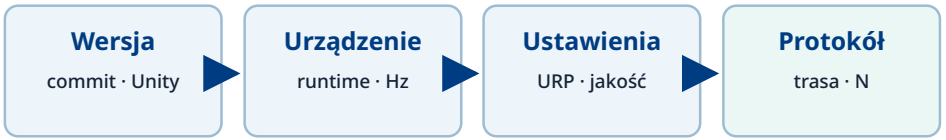


Po zmianie LOD, foveation lub rozdzielczości sprawdź w **ruchu**, a nie tylko na jednym zrzucie:

- Czy drobne etykiety są czytelne?
- Czy sylwetka i krawędzie istotne dla zadania nie migoczą?
- Czy elementy interaktywne pozostają identyfikowalne?
- Czy jakość jest porównywalna między oczami?

Zysk czasu bez jakości funkcjonalnej nie jest pełnym sukcesem.

Reprodukowanie projektu



Same FPS bez kontekstu nie są porównywalne

Wynik zależy od **urządzenia, wersji Unity, URP, OpenXR, assetów, shaderów, jakości i trasy**.

Zapisuj commit, platformę, konfigurację renderowania, liczbę próbek, warunki i definicję każdego licznika. Bez tego dwóch wyników „90 FPS“ nie można rzetelnie porównać.

Podsumowanie

1. Ustal częstotliwość i **budżet** okresu klatki.
2. Zidentyfikuj ograniczenie **CPU / GPU / synchronizacji**.
3. Dobierz jedną technikę: LOD, culling, instancing, foveation lub rozdzielczość.
4. Porównaj **medianę, p95 i błędy**, a następnie **jakość zadania**.

Zasada: optymalizujemy ograniczający koszt, nie liczbę trójkątów dla samej liczby.

Mini-eksperyment: czy optymalizacja rzeczywiście pomogła?

Stałe warunki: to samo urządzenie, scena, trasa i docelowe 90 Hz.

- **A:** zbierz bazowe czasy CPU/GPU i wskaźniki jakości zadania.
- **B:** zmień wyłącznie rozdzielczość renderowania i powtórz pomiar.
- **A ponownie:** przywróć ustawienia, aby ocenić wpływ zmian warunków.
- Porównaj medianę, p95, liczbę przekroczeń budżetu i widoczne artefakty.

Gdy ogranicza CPU, mniejsza liczba pikseli nie musi zwiększyć FPS.

Bibliografia

- Khronos, OpenXR 1.1 – specyfikacja.
- Unity 6000.6, ProfilerRecorder – czasy klatek.
- Unity, Renderowanie stereo.
- Unity, Skalowanie rozdzielczości XR.
- Unity 6000.6, CPU Usage Profiler.
- Patney i in. (2016), Foveated rendering w VR.
- Warburton i in. (2023), Pomiar motion-to-photon latency.