PRZEGLĄD TECHNOLOGII I ARCHITEKTUR BADAWCZYCH

Silniki i platformy dla przemysłowego XR

dr inż. Mateusz Pomianek, wersja 1.1.0

Streszczenie

Projektowanie środowisk łączących rzeczywistość rozszerzoną i wirtualną (*extended reality*, XR), robotykę, symulację fizyczną oraz renderowanie fotorealistyczne wymaga pogodzenia celów, które rzadko optymalizuje się tym samym silnikiem. Aplikacja XR potrzebuje stabilnego czasu generacji klatki i interakcji w czasie rzeczywistym; symulator robotyczny wymaga określonej dynamiki kontaktowej, sensorów i łączności z systemem sterowania; narzędzie do analizy naukowej często priorytetyzuje skalowalność obliczeń, powtarzalność i precyzję danych. Raport porządkuje te wymagania, porównuje reprezentatywne środowiska i przedstawia architektury, w których poszczególne komponenty mogą być zastępowane bez przebudowy całej aplikacji.

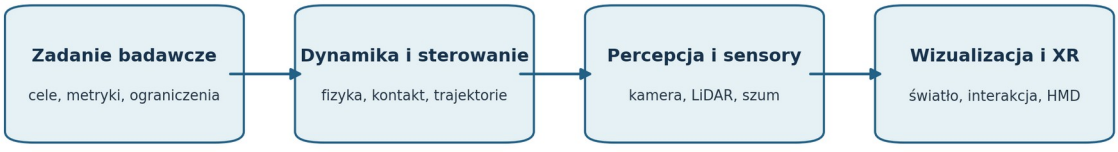
Analiza obejmuje Unreal Engine, Unity, Godot, Open 3D Engine (O3DE), NVIDIA Isaac Sim, Gazebo, MuJoCo, Drake, CARLA, VTK, ParaView, DualSPHysics oraz wybrane platformy specjalistyczne. Oceny mają charakter jakościowej syntezy dokumentacji technicznej,, a nie niezależnego benchmarku. Kluczowym wynikiem jest wskazanie architektury modułowej: odrębna definicja eksperymentu, właściwy solver lub symulator, ustandaryzowana wymiana danych oraz niezależna warstwa wizualizacji i XR. Rekomendacje podano warunkowo, zależnie od dominującego zadania, ograniczeń obliczeniowych i ryzyka integracyjnego.

Słowa kluczowe: XR, OpenXR, ROS 2, symulacja robotyczna, fotorealizm, MuJoCo, Gazebo, Isaac Sim, Unreal Engine, dane syntetyczne, sim-to-real, HPC, wizualizacja naukowa.

Najważniejsze ustalenia

Najbardziej przekonującym rozwiązaniem dla zróżnicowanych zastosowań jest **kompozycja wyspecjalizowanych narzędzi**,, a nie wybór jednego „zwycięskiego” silnika. Unreal Engine zapewnia rozbudowany warsztat wizualizacji i immersji, ale nie zastępuje zaawansowanego solvera dynamiki robotów. Gazebo upraszcza współpracę z ROS 2, MuJoCo zapewnia silne podstawy dynamiki kontaktowej i masowo równoległych symulacji, zaś Isaac Sim łączy fotorealistyczne sensory ze środowiskiem robotycznym kosztem znaczących wymagań sprzętowych. VTK i ParaView pełnią odmienną funkcję: pozwalają analizować wyniki obliczeń, nie zastępując interaktywnego symulatora [1,2,3,4,5,6,7].

Drugim wnioskiem jest konieczność rozdzielenia **wierności obrazu, wierności fizyki i wiarygodności sygnału sensorycznego**. Fotorealistyczna kamera może przekazywać obrazy o dużej jakości percepcyjnej, mimo że geometria kolizji, charakterystyka szumu lub opóźnienia komunikacji nie odpowiadają rzeczywistości. Z tego względu każda platforma powinna być oceniana według jawnego protokołu walidacyjnego,, a nie na podstawie efektownych materiałów demonstracyjnych.



Rys. 1. Wielowarstwowy model systemu XR i symulacji.

1. Cel, zakres i sposób oceny

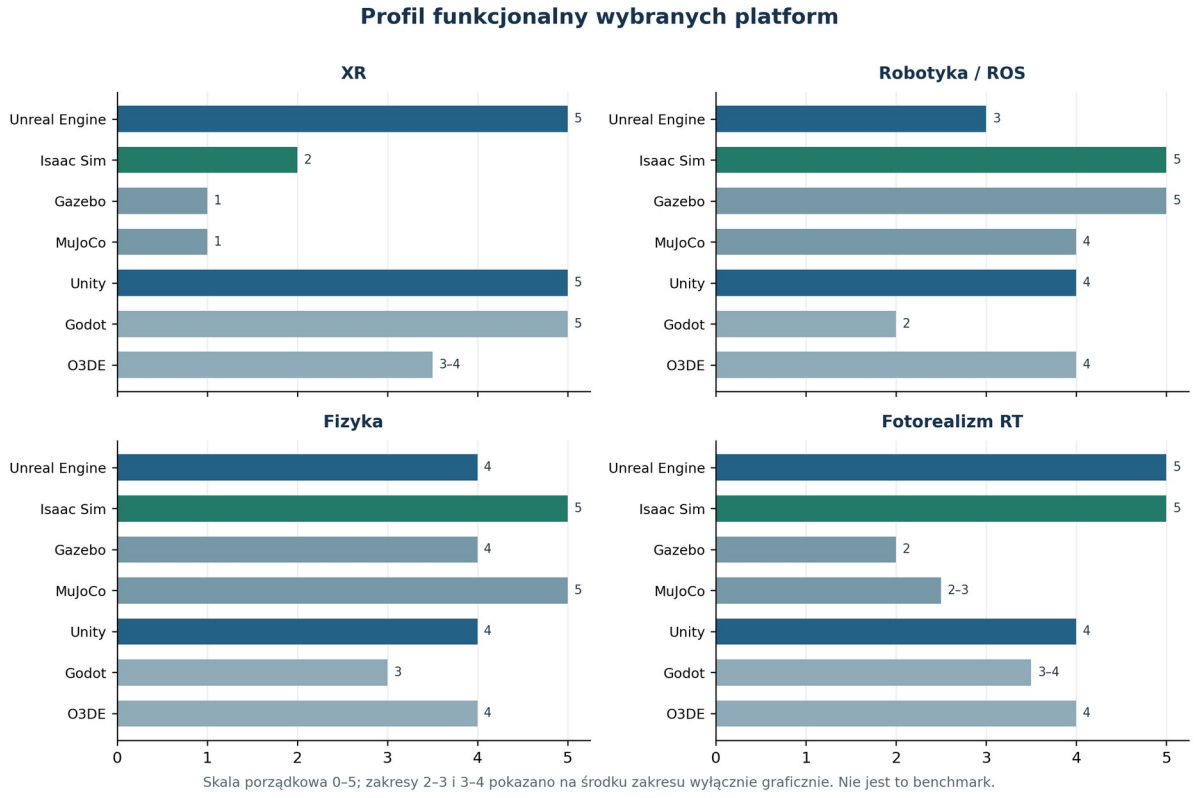
W praktyce porównania silników 3D łatwo prowadzą do pozornego rankingu: nazwy narzędzi zestawia się w jednej tabeli,, a następnie przypisuje im jedną ocenę końcową. Takie postępowanie miesza odmienne kategorie produktu. Silnik XR odpowiada za interakcję i renderowanie, solver CFD za równania przepływu,

środowisko robotyczne za modele układów wieloczłonowych i sensory, natomiast aplikacja wizualizacji naukowej za eksplorację danych. Zestawienie jest wartościowe dopiero wtedy, gdy cechy porównuje się względem konkretnego przypadku zastosowania. Podstawę stanowi dokumentacja producentów i projektów, której znaczenie zostało rozdzielone na trzy grupy: właściwości deklarowane i dokumentowane, wnioski architektoniczne oraz jakościowe oceny eksperckie. Tam, gdzie wersje lub warunki licencji zmieniają się w czasie, należy traktować podane informacje jako stan na listopad 2026 r. i sprawdzić je przed wdrożeniem.

1.1. Kryteria funkcjonalne

Wybór technologii prowadzono według sześciu wymiarów: (1) wsparcia XR i możliwości interakcji, (2) integracji z ROS 2 oraz dostępności modeli robotycznych, (3) wierności i szybkości dynamiki, (4) jakości renderowania czasu rzeczywistego, (5) renderowania offline i przetwarzania naukowego oraz (6) przenośności, kosztu utrzymania i ryzyka zależności od dostawcy. Ostatni wymiar nie jest własnością renderera: odzwierciedla zależności API, formatów scen, licencji, sprzętu i kompetencji zespołu.

Oceny 0–5 stosowane w grafice porównawczej mają charakter **porządkowy**: 5 oznacza centralną funkcję platformy, 4 bardzo dojrzałe wsparcie, 3 funkcję użyteczną, 2 rozwiązanie wymagające znacznej integracji, 1 marginalną przydatność i 0 obszar wykraczający poza przeznaczenie narzędzia. Nie wolno traktować różnicy między stopniami jako stałej jednostki efektywności,, a sumy punktów jako zmierzonej przewagi. Kategorie pokrewne mogą być oceniane odmiennie w zależności od klasy obciążeń, sceny i sprzętu.



Rys. 2. Eksperckie profile funkcjonalne (skala 0–5)

1.2. Typy źródeł i granice wnioskowania

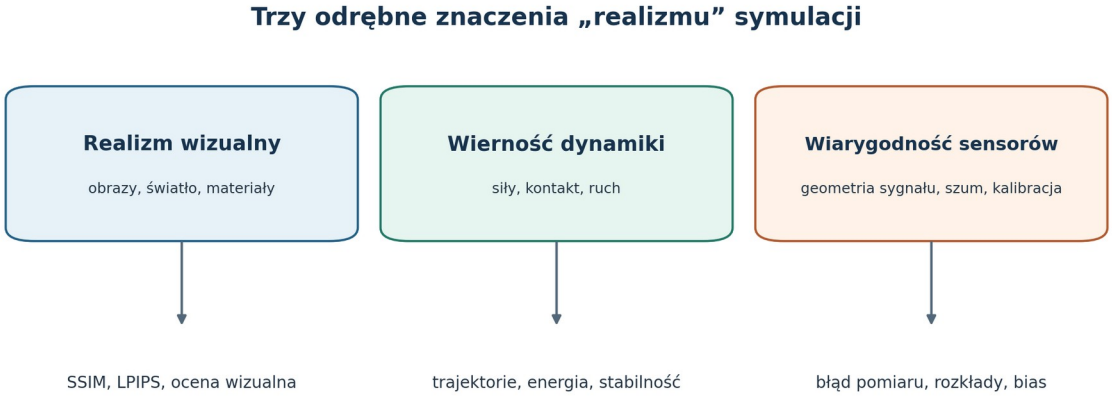
Dokumentacja techniczna wiarygodnie opisuje dostępność funkcji, formatów lub oficjalnych punktów integracji, lecz nie dowodzi przewagi w konkretnym eksperymencie. Z kolei przykłady dostarczane przez producenta zwykle nie kontrolują wszystkich zmiennych, takich jak złożoność sceny, sterowniki, obciążenie GPU i liczba sensorów. Raport nie przedstawia zatem niezmierzonych wartości liczby klatek na sekundę, opóźnień, przepustowości lub dokładności fizycznej. Parametry tego rodzaju muszą zostać ustalone we wspólnym teście na porównywalnych warunkach.

2. Dlaczego jeden silnik nie rozwiązuje wszystkich problemów?

2.1. Obraz, dynamika i dane nie są równoważne

W komputerowej grafice 3D podstawowym zadaniem jest wyznaczenie obrazu z geometrii, materiałów, źródeł światła i parametrów kamery. W symulacji fizycznej wynik ma inne znaczenie: jest stanem modelu matematycznego opisującego ruch, siły, przepływy lub kontakt. Moduł sensorów przekształca natomiast stan środowiska w obserwację podlegającą ograniczeniom rozdzielczości, charakterystyce przyrządu i błędom pomiarowym. Te trzy odwzorowania mogą działać na tej samej scenie, ale podlegają odrębnym kryteriom poprawności.

Dobrym przykładem jest robot mobilny. Wysokiej jakości render obrazu z kamery nie przesądza, czy tarcie kół, zderzenia, kalibracja LiDAR-u i opóźnienia między wiadomościami ROS odpowiadają rzeczywistemu układowi. Z drugiej strony bardzo szybki symulator sterowania może przedstawiać świat uproszczonym rendererem, ponieważ obraz jest wtórny wobec poprawnej trajektorii. Zwiększenie fotorealizmu bez równoczesnej walidacji innych warstw może zatem stworzyć złudne poczucie wiarygodności.



Rys. 3. Trzy komplementarne, lecz niezamienne aspekty wierności symulacji

2.2. Ograniczenie czasu klatki w XR

W XR obserwator reaguje na opóźnienie pomiędzy ruchem głowy lub kontrolera, a aktualizacją obrazu. Wymagany okres pojedynczej klatki wyraża zależność: $T_k = 1000/f$ [ms], gdzie f jest częstotliwością odświeżania wyrażoną w hercach. Dla 90 Hz daje to około 11,1 ms całkowitego budżetu na klatkę, który musi uwzględnić rendering stereoskopowy, logikę, obliczenia i narzut systemowy. Nie oznacza to jednak, że cały budżet jest dostępny silnikowi fizycznemu lub shaderom.

To wyjaśnia, dlaczego technologie pozwalające uzyskać realistyczny obraz przy umiarkowanej liczbie klatek na sekundę nie muszą nadawać się do immersyjnego VR. Dokumentacja Unreal opisuje Lumen jako zaawansowany system oświetlenia globalnego, ale wsparcie Nanite i Lumen w XR ma status eksperymentalny i jest ograniczone do wybranych konfiguracji PC; w docelowym projekcie należy uwzględniać koszty dwóch widoków oraz wymagania urządzenia [2, 8, 9]. Praktyczna jakość XR jest funkcją stabilności czasu klatki,, a nie tylko jej średniego tempa.

2.2.1. Nieciągłości, deterministyczność i częstotliwość aktualizacji

Obliczenia symulacyjne mogą wykorzystywać stały albo adaptacyjny krok czasowy. Krok stały ułatwia odtwarzanie eksperymentu, ale zbyt duża wartość może powodować niestabilność przy gwałtownym kontakcie, sprężystości lub szybkiej zmianie sił. Krok adaptacyjny ogranicza część takich problemów, lecz komplikuje synchronizację z zewnętrznym sterownikiem oraz porównywanie przebiegów. W modelu robotycznym trzeba więc niezależnie wyznaczyć częstotliwość integracji stanu, odświeżania sterowania, publikacji sensorów i renderowania obrazu. Te cztery wielkości nie muszą być równe.

Przykładowo, silnik może integrować dynamikę kilkakrotnie między kolejnymi klatkami XR,, a kamera syntetyczna może generować obraz z częstotliwością niższą niż pętla sterowania. Błędem jest odtwarzanie

wszystkich zdarzeń na podstawie jednego zegara interfejsu graficznego. Równie ważne są strategie synchronizacji: uruchamianie całego systemu w krokach deterministycznych, wymuszanie czasu rzeczywistego albo odtwarzanie danych z uprzednio zapisanego przebiegu. Każda z nich odpowiada innej klasie eksperymentu.

2.3. Rzeczywisty czas, czas symulowany i szybkość eksperymentu

Robotyka wykorzystuje także symulację szybszą od czasu rzeczywistego (*faster-than-real-time simulation*), szczególnie w uczeniu ze wzmocnieniem. Miarą jest współczynnik $R = \Delta t_{sim}/\Delta t_{r}zecz$. Gdy $R > 1$, symulator przetwarza więcej czasu modelowanego niż upływa czasu zegarowego; wartości R zależą jednak od liczby równoległych instancji, częstości synchronizacji, dynamiki i sprzętu. Wymaganie wysokiego R może uzasadniać zastosowanie MuJoCo/MJX zamiast rozbudowanego silnika wizualnego [4].

W obliczeniach CFD i SPH czas fizyczny jest ponadto niezależny od czasu renderowania. Bardziej racjonalne staje się wykonanie solvera w trybie bezgłowym (*headless*), zapisanie stanu i dopiero późniejsze tworzenie obrazów lub animacji w VTK i ParaView [6, 7, 10].

3. Technologie wizualizacji immersyjnej i silniki ogólnego przeznaczenia

3.1. Unreal Engine: zaawansowany renderer i rozbudowane środowisko XR

Unreal Engine łączy rozwinięty edytor scen 3D, materiałowy pipeline, systemy oświetlenia, programowanie w C++ i Blueprints oraz obsługę OpenXR. Z perspektywy badań naukowych jego główną zaletą jest możliwość tworzenia realistycznych i interaktywnych środowisk,, a także integracji danych geograficznych i krajobrazowych przez narzędzia takie jak Cesium. CARLA wykorzystuje Unreal jako część specjalistycznego systemu symulacji jazdy autonomicznej, w którym ważne są realistyczne warunki obserwacji i sensory [2, 11, 12].

Nie oznacza to natywnej pełnej zgodności z ROS 2. Integracja robotyczna zależy od adapterów i wtyczek, które należy testować pod kątem czasu transmisji, jakości synchronizacji i zgodności wersji. Unreal jest więc naturalnym wyborem dla XR, demonstratorów wizualnych i fotorealistycznej prezentacji wyników, ale nie stanowi samodzielnego zastępstwa Gazebo, MuJoCo czy Drake w zadaniach wymagających szczegółowej kontroli dynamiki. Wersje renderera powinny być odrębnie dobierane dla ekranu, XR i zadań offline.

3.2. Unity: pomost pomiędzy XR, robotyką, a uczeniem maszynowym

Unity zapewnia dojrzały interaktywny przepływ pracy, ekosystem C#, OpenXR i rozbudowane biblioteki zasobów. Robotics Hub obejmuje ROS-TCP-Connector oraz URDF Importer, który przenosi strukturę robota do elementów dynamiki środowiska. ML-Agents umożliwia użycie scen jako środowisk uczenia ze wzmocnieniem lub uczenia przez naśladowanie, z warstwą treningową Python/PyTorch [13, 14].

Zaletą Unity jest skrócenie drogi od eksperymentu 3D do aplikacji z interfejsem użytkownika. W robotyce cena tej wygody obejmuje jednak utrzymywanie łącznika komunikacyjnego, warstwy fizycznej i zgodności z aktualną dystrybucją ROS. Dodatkową zmienną jest licencja: poza zastosowaniami rozrywkowymi zasady Unity Industry zależą od profilu działalności i progu finansowego; nie należy przyjmować, że typowa licencja edukacyjna lub Unity Personal obejmuje każdy projekt instytucjonalny [15].

3.3. Godot: otwarte środowisko XR i kontrola nad kodem

Godot, udostępniany na licencji MIT, oferuje otwarty kod źródłowy, obsługę OpenXR oraz własny zestaw technologii renderowania. Może być dobrym punktem wyjścia dla mobilnych demonstratorów, środowisk edukacyjnych, prototypów cyfrowych bliźniaków (*digital twins*) i aplikacji z nietypowym interfejsem. GDScript, C# i rozszerzenia C++ sprzyjają szybkiemu tworzeniu narzędzi, w których integracja ze specjalistyczną symulacją odbywa się poza samym silnikiem [16, 17].

Najważniejsze ograniczenie wynika nie z braku grafiki 3D, ale z mniejszej dostępności gotowych komponentów robotycznych, sensorów i systemów danych syntetycznych. W poważniejszym zastosowaniu ROS-2

niezbędna będzie własna warstwa komunikacji i uzgodnienie znaczenia danych. Zaletę otwartej licencji należy więc rozpatrywać łącznie z kosztem utrzymania brakujących adapterów.

3.4. Open 3D Engine i pozostałe środowiska lekkie

Open 3D Engine (O3DE) zajmuje szczególne miejsce między komercyjnymi silnikami wysokiej jakości grafiki, a narzędziami o otwartym kodzie. Renderer Atom i moduł ROS 2 Gem pozwalają rozważać budowę otwartego symulatora z bogatą warstwą sceny 3D. Jednocześnie koszt kompilacji, integracji i utrzymania rozwiniętego stosu C++ bywa większy niż przy prostszych platformach. Dlatego O3DE można rozważyć szczególnie tam, gdzie możliwość zmian wewnątrz renderera lub warstwy robotycznej jest wymaganiem projektu [18].

Panda3D stanowi wygodny framework Python/C++ dla niestandardowych wizualizacji i symulacji sterowanych kodem. Cocos Creator koncentruje się na lekkich aplikacjach mobilnych i internetowych,, a Defold na stosunkowo niewielkich aplikacjach 2D/3D. Żaden z tych produktów nie zastępuje wyspecjalizowanej infrastruktury ROS, fizyki kontaktowej czy rendererów klasy Unreal. Ich sensowność rośnie wtedy, gdy podstawowym wymaganiem jest minimalna złożoność klienta i krótki czas przygotowania interfejsu [19, 20].

4. Robotyka, sensory i symulacja dynamiki

4.1. Gazebo: modelowanie robotów w ekosystemie ROS 2

Gazebo jest modularnym środowiskiem symulacji robotycznej, w którym fizyka, renderowanie, sensory i wymiana wiadomości stanowią wyraźnie zdefiniowane komponenty. Zastosowanie formatów SDF/URDF oraz narzędzi ros_gz czyni go rozwiązaniem szczególnie naturalnym dla robotów mobilnych, nawigacji, lokalizacji, manipulatorów i eksperymentów nad fuzją sensorów. Modułowość zwiększa możliwość budowania powtarzalnego środowiska testowego i uruchamiania symulacji bez GUI [3].

Ważna jest jednak konkretna kombinacja dystrybucji Gazebo i ROS 2. Zgodność między wydaniami jest utrzymywana i opisywana w dokumentacji, ale nie należy zakładać zgodności wszystkich wersji. Wybór należy więc zapisać w specyfikacji projektu razem z wymaganym systemem operacyjnym, formatami modeli, pluginami i kryteriami testu integracyjnego [3].

4.2. MuJoCo: wydajność dynamiki kontaktowej i uczenia sterowania

MuJoCo specjalizuje się w szybkiej dynamice układów wieloczłonowych, modelowaniu kontaktów i zadaniach sterowania. Oprogramowanie jest dostępne na licencji Apache 2.0, ma interfejsy C/Python i obsługuje m.in. MJCF oraz URDF. W zastosowaniach polegających na wielokrotnym wykonywaniu trajektorii, optymalizacji polityk lub identyfikacji parametrów dynamiki może osiągnąć korzystniejszy kompromis między wydajnością,, a złożonością niż rozbudowany silnik graficzny [4].

Rodzina MJX obejmuje implementacje wykorzystujące JAX i Warp. Dokumentacja rozróżnia MJX-JAX oraz MJX-Warp, które mogą mieć inne właściwości wydajnościowe i inne możliwości różniczkowania. Nie należy więc traktować skrótu „GPU MuJoCo” jako jednej bezwarunkowo szybszej implementacji: uzyskana przepustowość zależy od charakterystyki kontaktów, liczby światów i kosztu transmisji danych [4]. Wysoki realizm wizualny pozostaje w tym stosie funkcją dodatkową.

4.3. Drake: modele matematyczne, optymalizacja i sterowanie

Drake to zestaw bibliotek C++/Python służących modelowaniu dynamiki, projektowaniu trajektorii oraz obliczeniom optymalizacyjnym. Największą wartość przynosi tam, gdzie model jest częścią rozumowania o sterowalności, ograniczeniach i właściwościach układu. Wizualizacja ma charakter wspomagający,, a nie podstawowy. Typowy scenariusz polega na użyciu Drake jako komponentu referencyjnego i osobnej warstwy graficznej [21].

4.4. NVIDIA Isaac Sim: sensory RTX, dane syntetyczne i integracja robotyczna

Isaac Sim łączy modelowanie robotów z renderowaniem wysokiej jakości, symulowanymi sensorami i narzędziami generowania danych syntetycznych (*synthetic data generation*). Integracja ROS 2, modelami URDF/MJCF oraz formatem USD upraszcza budowanie środowisk do badań nad percepcją i przenoszeniem algorytmów z symulacji na sprzęt (*simulation-to-reality transfer*, sim-to-real) [5].

Takie połączenie zwiększa jednak wymagania sprzętowe i zależność od urządzeń NVIDIA. Wymagania Isaac Sim 6.2 wskazują minimum 32 GB pamięci RAM oraz GeForce RTX 4080 z 16 GB VRAM; konfiguracje z rozbudowanymi sensorami mogą wymagać większego zapasu [22]. Warunki licencyjne odróżniają kod źródłowy od komponentów Omniverse Kit i zasobów na odrębnych licencjach. Wykorzystanie na potrzeby wewnętrznych prac R&D nie jest równoznaczne z prawem do redystrybucji całej platformy jako usługi lub produktu [23].

4.4.1. Dane syntetyczne i przeniesienie modelu do rzeczywistości

Środowisko zapewniające fotorealistyczne sensory może pomóc w tworzeniu dużej liczby kontrolowanych scen treningowych: od zmian oświetlenia i materiałów po warianty trajektorii, zasłonięć czy konfiguracji kamer. Jednak syntetyczne zróżnicowanie danych (*domain randomization*) nie jest zamiennikiem kalibracji przyrządu. Jeżeli symulator nie odtwarza charakterystyki obiektywu, szumu matrycy, przebiegu skanowania LiDAR-u, interferencji i opóźnień, model percepcji może nauczyć się zależności charakterystycznych tylko dla świata wirtualnego.

Ocena różnicy domen (*sim-to-real gap*) wymaga dlatego oddzielnych miar: skuteczności algorytmu na scenach syntetycznych, skuteczności na niezależnych danych rzeczywistych oraz zmian jakości po transferze. Trzeba też kontrolować, czy zbiór testowy nie był wykorzystywany do kalibracji modelu lub generatora scen. W badaniach nad rozpoznawaniem obiektów sama zgodność estetyczna obrazów jest niewystarczająca, ponieważ o błędzie często decydują drobne własności krawędzi, widma, rozdzielczości lub rozkładu przypadków rzadkich.

4.5. CARLA i symulacja domenowa

CARLA jest wyspecjalizowaną platformą służącą eksperymentom nad pojazdami autonomicznymi. Zawiera modele kamer, LiDAR-u, radaru, GPS/GNSS i IMU oraz interfejsy używane w integracji z robotyką. Jej przewaga polega na gotowym modelu problemu motoryzacyjnego i bogatym środowisku drogowym. Nie należy przenosić tej przewagi na roboty manipulacyjne, humanoidalne lub zadania hydrodynamiczne, dla których bardziej adekwatne są inne solvery [11].

5. Wizualizacja naukowa, przetwarzanie wsadowe i solvery domenowe

5.1. VTK i ParaView

Visualization Toolkit (VTK) jest biblioteką do wizualizacji i przetwarzania danych, w tym pól skalarnych i wektorowych, siatek, chmur punktów czy obrazowania objętościowego. ParaView wykorzystuje VTK w rozbudowanym środowisku analizy, w którym dostępne są techniki client–server, praca z dużymi zbiorami danych i wybrane metody renderowania oparte na śledzeniu promieni. W obu przypadkach centrum projektu stanowią dane naukowe,, a nie silnik interaktywnego świata [6, 7].

To rozróżnienie jest istotne, ponieważ oczekiwanie od ParaView natywnego sterowania robotem lub zestawu interakcji HMD prowadzi do niewłaściwej architektury. Jednocześnie pomijanie ParaView w projekcie CFD lub SPH zwiększa koszt budowy własnych narzędzi do przeglądania wyników. Słuszny podział polega na pozostawieniu symulacji i plików naukowych poza częścią prezentacyjną.

5.2. DualSPHysics i hydrodynamika cząsteczkowa

DualSPHysics rozwiązuje problemy hydrodynamiki metodą wygładzonych cząstek (*smoothed particle hydrodynamics*, SPH), wykorzystując wielordzeniowe CPU i odpowiednie urządzenia GPU. Jest to specjalistyczny

solver,, a nie odpowiednik MuJoCo lub silnika XR. W badaniach nad powierzchnią swobodną, falami i interakcją płynu ze strukturą sensowne jest zachowanie obliczeń w DualSPHysics, po czym przeprowadzenie analizy w ParaView lub VTK [10].



Rys. 4. Referencyjny przepływ danych w obliczeniach naukowych

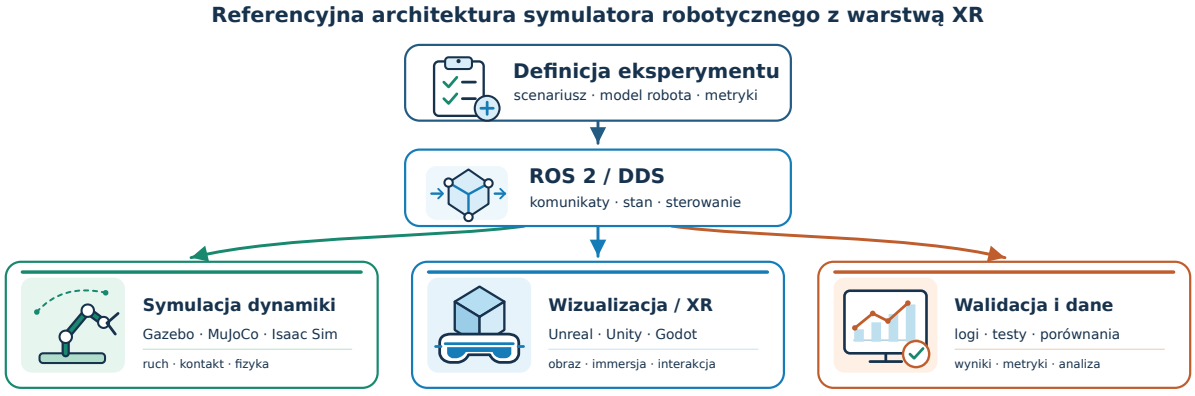
5.3. Technologie specjalistyczne i biblioteki renderujące

OpenSpace obsługuje wizualizację astronomiczną i prezentacje planetarne, SpaceEngine oferuje rozbudowaną prezentację kosmosu, lecz nie zapewnia standardowego, publicznego SDK do budowy dowolnego systemu symulacyjnego. Z perspektywy tworzenia niezależnej infrastruktury badawczej te rozwiązania należy traktować jako domenowe,, a nie ogólną alternatywę dla XR i robotyki [24, 25].

OpenSceneGraph pozostaje ważną technologią historycznych systemów *visual simulation*, ale jego rola w nowych projektach jest ograniczana przez status technologii wcześniejszej generacji. VulkanSceneGraph dostarcza niskopoziomowy graf scen oparty na Vulkan i C++, przeznaczony dla zespołów dysponujących kompetencjami w budowie własnego renderera. Nie zawiera on pełnego stosu fizyki, urządzeń XR, edytora i middleware: elementy te trzeba połączyć osobno [26, 27].

6. Architektura zintegrowanego systemu badawczego

Najbezpieczniejszą decyzją architektoniczną jest przechowywanie opisu eksperymentu i logiki sterowania poza środowiskiem graficznym. Rozróżnienie „źródła prawdy" i „reprezentacji wizualnej" pozwala wykorzystać ten sam model ruchu w różnym rendererze,, a także porównywać wynik symulacji z danymi z urządzenia rzeczywistego. Integracja może wykorzystywać ROS 2/DDS,, a przenośność scen i robotów opierać się na formatach URDF/SDF/MJCF, USD oraz glTF, zgodnie z właściwościami danego komponentu [3, 4, 5].



Rys. 5. Przykładowa architektura systemu łączącego symulację robotyczną, ROS 2, XR i walidację

6.0.1. Jakie dane powinny pozostać niezależne od silnika?

Dobrze zaprojektowana struktura danych powinna rozróżniać co najmniej cztery poziomy. Pierwszy to geometria i kinematyka: jednostki, układy współrzędnych, osie przegubów, masy oraz bezwładność. Drugi obejmuje parametry środowiska i kontaktu, takie jak grawitacja, współczynniki tarcia oraz rodzaj elementów kolizyjnych. Trzeci to protokół sensorów z parametrami kalibracji i częstotliwościami próbkowania. Czwarty zawiera opis próby: cel sterownika, stany początkowe, ziarna losowości, sposób oceny i warunki zatrzymania. Dopiero na tym fundamencie należy budować tekstury, animacje i szczegóły graficzne właściwe dla wybranego renderera.

W takim układzie przeniesienie pomiędzy dwiema platformami jest problemem jawnego odwzorowania specyfikacji,, a nie „wyeksportowania sceny". Część parametrów może nie mieć odpowiednika w docelowym silniku. Trzeba wtedy wybrać model przybliżony, oszacować jego błąd lub oznaczyć wynik jako nieporównywalny. Pozwala to uniknąć sytuacji, w której podobne obrazy sugerują zgodność zachowania robota, mimo że zmieniła się fizyka układu.

6.1. Co podlega migracji, a co pozostaje własnością silnika?

Eksport geometrii nie oznacza przeniesienia całego eksperymentu. Plik sceny może zachować położenie siatek i część materiałów,, a mimo to utracić konfigurację sensorów, złożone modele kolizji, zależności między przegubami, skrypty i logikę interakcji. Formaty wymiany, takie jak USD i glTF, rozwiązują różne części tego problemu. Należy więc stworzyć niezależną specyfikację: opis robota i parametrów fizycznych, konfigurację sensorów, definicje komunikatów, mapę układów współrzędnych, harmonogram próbkowania i oczekiwane rezultaty.

Z tego wynika praktyczny wymóg wersjonowania sceny, bibliotek, sterowników i kalibracji. Jeśli model przeniesiono między silnikami, nie wystarczy wizualne porównanie; trzeba sprawdzić zgodność mas, momentów bezwładności, kinematyki, ograniczeń ruchu i jednostek. Niezgodność systemu współrzędnych lub założeń o czasie symulowanym bywa bardziej kosztowna niż błąd w materiałach graficznych.

Warstwa	Typowy format / interfejs	Uwaga przy migracji
Model robota	URDF, MJCF, SDF	Sprawdzić dynamikę, jednostki i przeguby
Komunikacja	ROS 2 / DDS	Zweryfikować czas, QoS i wersje komunikatów
Scena i geometria	USD, glTF	Nie wszystkie rozszerzenia zachowują semantykę
Geodane	3D Tiles / GIS	Wymagana zgodność układów odniesienia
Wyniki naukowe	VTK i formaty domenowe	Zachować pola, siatki i metadane

6.2. Integracja niesynchronicznych komponentów

W rozdzielonym systemie źródłem problemów mogą być: częstotliwość kroków solvera, opóźnienia transportu sieciowego, niejednoznaczna kolejność wiadomości, interpolacja animacji i różnice między zegarem symulacji a zegarem ściennym. Należy jawnie określić, czy każdy odbiorca potrzebuje najnowszego stanu, kompletnej historii, czy stanu z określonym znacznikiem czasu. Dla eksperymentów odtwarzalnych istotne są identyfikatory przebiegów, zapis parametrów losowości oraz deterministyczny format archiwum.

7. Ocena kosztu, ograniczeń i ryzyka utrzymania

7.1. Koszt całkowity, to nie tylko cena licencji

Wybór środowiska powinien obejmować koszt całkowity użytkowania (*total cost of ownership*, TCO): licencję, sprzęt, integrację, utrzymanie API, testowanie po aktualizacjach oraz koszt kompetencji programistycznych. Otwarte narzędzie bywa droższe w eksploatacji, jeśli wymaga stworzenia własnych sterowników sensorów. Komercyjny silnik może zmniejszyć czas opracowania scen i interfejsu, lecz zwiększa zależność od licencji i producenta.

Godot i MuJoCo są licencjonowane odpowiednio MIT i Apache 2.0, Gazebo oraz O3DE również bazują na otwartych licencjach,, a Drake, VTK i ParaView mają modele BSD lub pokrewne [18,3,4,21,6,7,17]. Unreal korzysta z odrębnej umowy licencyjnej i zasad opłat zależnych od zastosowania [28]. Unity wymaga sprawdzenia poziomu planu oraz definicji klienta branżowego [15]. Isaac Sim ma licencję kodu źródłowego oraz odrębne warunki dotyczące wymaganych komponentów [23].

7.1.1. Ryzyko licencyjne, a możliwość publikowania wyników

W ocenie licencji należy rozróżnić prawa do korzystania z programu, dystrybucji jego kodu, udostępniania gotowej aplikacji oraz rozpowszechniania wytworzonych danych. Przykładowo, dopuszczalność wewnętrznych prac badawczo-rozwojowych w Isaac Sim nie jest automatyczną zgodą na udostępnienie pełnego środowiska

jako usługi. Podobnie w Unity zasady odnoszące się do zastosowań branżowych mogą różnić się od zasad standardowego tworzenia gier. Dokumentacja licencyjna powinna być weryfikowana w momencie wydawania aplikacji,, a nie tylko podczas projektowania [23,15].

Z naukowego punktu widzenia istotne jest również, czy odbiorca publikacji może samodzielnie odtworzyć eksperyment. Brak możliwości ponownego uruchomienia komercyjnego komponentu nie dyskwalifikuje badania, ale zwiększa znaczenie udostępnienia parametrów, danych wejściowych, wyników pośrednich i testów porównawczych. Odtwarzalność należy traktować jako właściwość całego procesu,, a nie samej licencji open source.

7.2. Sprzęt i architektura obliczeniowa

Wydajność środowiska zależy od tego, czy dominują operacje geometrii i shaderów, silnik dynamiki, sensory, transfer GPU–CPU czy obliczenia zewnętrzne. Isaac Sim wymaga stosunkowo silnej karty RTX i dużej ilości pamięci,, a wielosensorowe sceny zwiększają zapotrzebowanie na VRAM [22]. Gazebo można wykonywać bez interfejsu graficznego; MuJoCo dla pojedynczej symulacji może mieć znacznie mniejsze potrzeby sprzętowe, choć masowe treningi MJX także wykorzystują akceleratory [3,4]. ParaView umożliwia zdalną i równoległą wizualizację dużych wyników [7].

Pytania o konkretny komputer należy zastąpić protokołem obciążeniowym: liczba modeli i trójkątów, liczba aktywnych sensorów, rozdzielczość kamer, liczba kontaktów, liczba równoległych światów, maksymalny czas przetwarzania oraz docelowe środowisko uruchomieniowe. Dopiero te parametry pozwalają oszacować wymagania RAM/VRAM i dobrać architekturę serwerową lub lokalną.

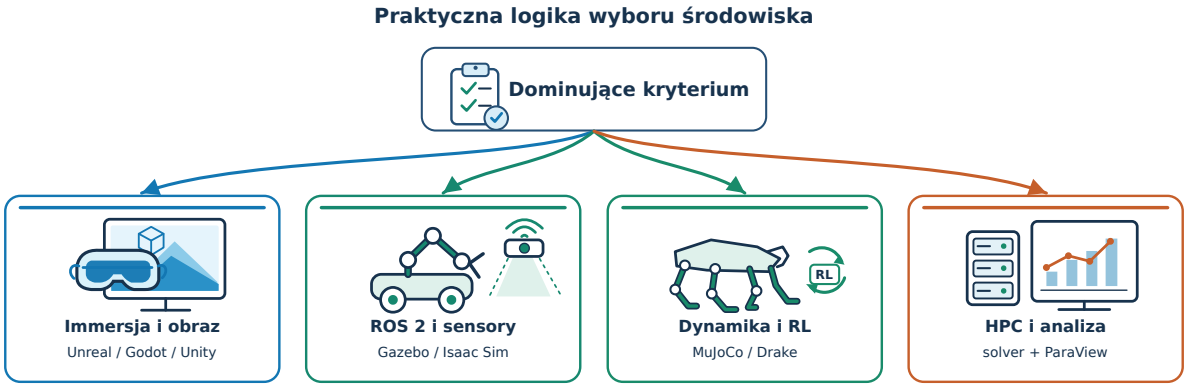
7.3. Reprodukowalność i ryzyko zależności od dostawcy

Otwartość kodu zmniejsza część ryzyk licencyjnych, ale nie eliminuje zależności od wyspecjalizowanego API lub formatu sceny. Ryzyko należy oceniać przez możliwość odtworzenia eksperymentu po migracji, dostępność długoterminowego wsparcia, liczbę niezależnych implementacji formatu oraz koszt zastąpienia komponentu. Z tej perspektywy utrzymywanie neutralnych opisów robotów i metryk poza rendererem jest jednym z najważniejszych środków ograniczania kosztów przyszłej migracji.

8. Porównanie scenariuszowe i procedura wyboru

Kryteria platformy powinny wynikać z problemu badawczego. Ocenianie „najlepszości” silnika bez wskazania zadania prowadzi do błędnych wniosków: otwarty silnik XR może być optymalny dla badania interakcji, natomiast dedykowany solver będzie właściwszy dla dynamiki. Poniżej zestawiono rekomendacje warunkowe, które można wykorzystać jako punkt wyjścia do eksperymentu porównawczego.

Dominujący cel	Rozwiązania pierwszego wyboru	Główne ryzyko
Immersyjny XR i wysoka jakość obrazu	Unreal Engine; Godot i Unity jako alternatywy	Budżet klatki, wsparcie urządzeń
Robot mobilny, ROS 2 i nawigacja	Gazebo; Isaac Sim dla sensorów RTX	Zgodność wersji ROS/Gazebo
Uczenie sterowania i kontakt	MuJoCo/MJX; Drake dla optymalizacji	Fidelity modelu i stabilność kontaktu
Dane syntetyczne / sim-to-real	Isaac Sim; CARLA w autonomicznej jeździe	Model sensora, różnica domen
CFD/SPH i naukowa analiza	DualSPHysics + VTK/ParaView	Powtarzalność solvera i skali danych
Przenośność, modyfikowalność kodu	Godot + Gazebo/MuJoCo; O3DE	Koszt własnych adapterów



Rys. 6. Uproszczona logika wyboru narzędzi zorientowana na dominujące wymaganie projektu

8.1. Protokół testu porównawczego

Właściwy test powinien używać tej samej sceny referencyjnej i wspólnego opisu działania robota, o ile da się zapewnić równoważne modele. W eksperymencie XR należy mierzyć medianę i 95. percentyl czasu klatki, opóźnienie interakcji, stabilność po zmianie sceny oraz czytelność obrazu. W robotyce należy oddzielnie mierzyć błąd trajektorii, zgodność modelu kontaktowego, kompletność publikowanych informacji ROS, jakość sensorów i zachowanie przy utracie wiadomości. W HPC ważne są przepustowość, pamięć szczytowa oraz odtwarzalność danych wynikowych.

Wyniki należy raportować w parach: *jakość–koszt*, np. błąd pozycji względem czasu kroku, opóźnienie względem liczby sensorów lub stabilność klatek względem rozdzielczości. Porównanie wyłącznie średnich FPS lub czasu uruchomienia pomija zjawiska istotne dla bezpieczeństwa i wartości naukowej. Jeśli scena nie może być odwzorowana równoważnie między silnikami, należy wskazać zakres nieporównywalności zamiast sztucznie wymuszać wspólny ranking.

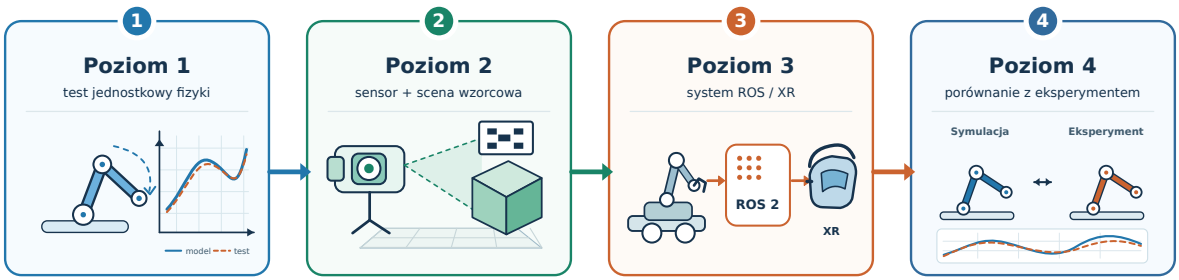
8.1.1. Porównywanie kosztu i jakości bez mylących rankingów

Przed wykonaniem oceny wielokryterialnej trzeba wybrać warunki nieprzekraczalne (*hard constraints*) i kryteria preferencyjne. Do pierwszej grupy mogą należeć konkretna wersja ROS 2, obsługa wskazanego urządzenia XR, określony format modelu robota oraz limit czasu klatki. Narzędzie, które nie spełnia warunku koniecznego, nie powinno otrzymywać rekomendacji tylko dlatego, że dobrze wypada pod względem fotorealizmu. Pozostałe kryteria można rozpatrywać na zasadzie dominacji: platforma jest preferowana, gdy nie jest gorsza we wszystkich istotnych wymiarach i lepsza co najmniej w jednym.

Przy ocenie eksperckiej opartej na skali porządkowej należy zachować ostrożność wobec wyliczania średnich. Różnica pomiędzy ocenami „3” i „4” nie musi odpowiadać różnicy między „4” i „5”,, a oszacowania dwóch specjalistów mogą mieć odmienną kalibrację. Bardziej przejrzyste jest podanie uzasadnienia dla każdej cechy, identyfikacja kryterium rozstrzygającego i powtórzenie analizy po zmianie priorytetów. Dzięki temu rekomendacja pozostaje zrozumiała również dla odbiorcy, który ma inne wymagania.

8.2. Weryfikacja i walidacja

Weryfikacja odpowiada na pytanie, czy oprogramowanie prawidłowo rozwiązuje zaimplementowany model; walidacja dotyczy tego, czy model odpowiada badanemu zjawisku z dokładnością wymaganą w zadaniu. Test jednostkowy integratora nie mówi nic o tym, czy współczynnik tarcia jest realistyczny; z kolei wizualna zgodność trajektorii nie potwierdza poprawnego zachowania całego systemu sterowania. Dlatego należy przechodzić od prostych przypadków analitycznych przez testy sensorów i całego middleware aż do kontrolowanego porównania z pomiarem rzeczywistym.



Rys. 7. Cztery poziomy oceny systemu symulacyjnego

8.3. Minimalny zestaw kryteriów akceptacji

Przed wyborem platformy należy zdefiniować co najmniej: docelową platformę XR i częstotliwość odświeżania, model i wielkość świata, opis układów współrzędnych, rodzaj sensorów i ich budżet błędu, minimalną liczbę równoległych instancji, potrzebne API i formaty eksportu, scenariusze awaryjne, limit kosztów oraz warunki dystrybucji. Parametry te stanowią podstawę prototypu porównawczego i mogą być powiązane z automatycznymi testami regresyjnymi.

8.4. Przykład planu walidacji na przestrzeni rozwoju projektu

Etap prototypu powinien rozpocząć się od minimalnego układu z jednym robotem lub jednym solverem, prostym środowiskiem i automatycznym zapisem wyników. Dopiero po wykryciu różnic w dynamice i synchronizacji można wprowadzić pełną scenę 3D oraz XR. W kolejnym etapie warto dodać realistyczne sensory, kontrolowaną randomizację oraz test przenoszenia algorytmu pomiędzy symulatorami. Zaawansowana wizualizacja nie powinna wyprzedzać walidacji danych, od których zależy poprawność hipotezy badawczej.

Dla zespołu badawczego korzystne jest prowadzenie dziennika zmian: wersji silnika, rozszerzeń, sterownika GPU, modeli, parametrów sceny i scenariuszy testowych. W wyniku każdej aktualizacji należy porównać nowe przebiegi z zestawem scen referencyjnych. Test regresyjny nie musi zawsze wymagać identycznych bitowo trajektorii; w układach niedeterministycznych może opierać się na dopuszczalnym zakresie odchyleń, rozkładzie błędu lub stabilności metryk. Kluczowe jest wcześniejsze ustalenie progu akceptacji i raportowanie wyjątków.

9. Wnioski i rekomendacje

Przeprowadzona synteza wskazuje, że najważniejszym wyborem nie jest nazwa silnika, lecz granica między odpowiedzialnościami poszczególnych komponentów. Dla systemu zorientowanego na XR naturalnym rdzeniem interfejsu będzie Unreal, Unity lub Godot, ale właściwą dynamikę i sensory trzeba dobrać odrębnie. Dla systemu robotycznego bardziej racjonalny jest Gazebo, MuJoCo lub Isaac Sim jako rdzeń symulacji,, a renderer stanowi komponent dodatkowy. W badaniach obliczeniowych solver domenowy powinien pozostawać niezależny od narzędzi prezentacyjnych.

Szczególnie istotne są trzy zasady: **(1) najpierw zdefiniować metryki i model, następnie wybrać technologię; (2) nie utożsamiać realizmu obrazu z wiernością symulacji; (3) utrzymywać neutralną reprezentację danych i warstwę walidacji poza silnikiem graficznym**. Pozwalają one budować środowiska bardziej odtwarzalne i łatwiejsze do modernizacji, bez zakładania istnienia uniwersalnej platformy.

Dla szerokiej infrastruktury łączącej robotykę i XR uzasadniony jest wariant z ROS 2 jako warstwą komunikacji, Gazebo lub MuJoCo jako symulatorem referencyjnym oraz Unreal lub Godot jako interfejsem immersyjnym. Isaac Sim może stanowić wyspecjalizowane środowisko dla sensorów i zadań sim-to-real,, a VTK/ParaView uzupełniać całość o analizę naukową. Wariant należy jednak potwierdzić testem dostosowanym do skali eksperymentu, sprzętu i ograniczeń prawnych [3, 4, 5, 22, 23, 21, 6, 7].

10. Ograniczenia opracowania i kierunki dalszych badań

Raport opiera się głównie na dokumentacji technicznej i syntetycznej ocenie inżynierskiej. Nie wyznaczono nowych wyników ilościowych dotyczących FPS, realizmu sensorów, przyspieszenia symulacji, kosztów lub dokładności fizycznej dla wspólnego zbioru przypadków testowych. Ranking ekspercki nie może zastąpić eksperymentu,, a część funkcji zależy od konkretnej wersji programu, sterownika i urządzenia. W szczególności nie należy uogólniać wymagań pojedynczej wersji Isaac Sim na całą rodzinę produktów ani traktować eksperymentalnych funkcji renderowania XR jako zapewnienia produkcyjnego [8,22].

Dalsze porównania powinny obejmować: (i) niezależną kalibrację modeli kontaktowych i sensorów, (ii) test czasu klatki XR w scenie o ustalonej geometrii, (iii) koszt odtwarzania scen między formatami USD, URDF i SDF, (iv) test integracji ROS 2 przy obciążeniu sieciowym, (v) rzeczywisty koszt portowania eksperymentu między silnikami oraz (vi) porównanie jakości danych syntetycznych z danymi pomiarowymi dla znanych przypadków. Tylko taki zestaw umożliwia w przyszłości wyprowadzenie wiarygodnej, ilościowej oceny jakości i kosztu poszczególnych architektur.

Bibliografia

[1] Khronos Group, “OpenXR Registry.” [Online]. https://registry.khronos.org/OpenXR/. [3 lis. 2026].
[2] Epic Games, “XR Best Practices in Unreal Engine.” [Online]. https://dev.epicgames.com/documentation/unreal-engine/xr-best-practices-in-unreal-engine. [3 lis. 2026].
[3] Gazebo, “Installing Gazebo with ROS.” [Online]. https://gazebosim.org/docs/latest/ros_installation/. [3 lis. 2026].
[4] Google DeepMind, “MuJoCo XLA (MJX), Documentation.” [Online]. https://mujoco.readthedocs.io/en/latest/mjx.html. [3 lis. 2026].
[5] NVIDIA, “Isaac Sim: ROS Integration Documentation.” [Online]. https://docs.isaacsim.omniverse.nvidia.com/6.0.0/installation/install_ros.html. [3 lis. 2026].
[6] Kitware, “The Visualization Toolkit (VTK): Overview.” [Online]. https://vtk.org/overview/. [3 lis. 2026].
[7] Kitware, “ParaView: High Performance Computing.” [Online]. https://www.paraview.org/high-performance-computing/. [3 lis. 2026].
[8] Epic Games, “Nanite and Lumen for XR in Unreal Engine.” [Online]. https://dev.epicgames.com/documentation/unreal-engine/nanite-and-lumen-for-xr-in-unreal-engine. [3 lis. 2026].
[9] Epic Games, “Lumen Technical Details in Unreal Engine.” [Online]. https://dev.epicgames.com/documentation/unreal-engine/lumen-technical-details-in-unreal-engine. [3 lis. 2026].
[10] DualSPHysics Project, “DualSPHysics: Open-Source SPH Solver.” [Online]. https://github.com/DualSPHysics/DualSPHysics. [3 lis. 2026].
[11] CARLA Team, “CARLA Simulator: Documentation.” [Online]. https://carla.readthedocs.io/en/latest/. [3 lis. 2026].
[12] Cesium GS, “Cesium for Unreal: Documentation.” [Online]. https://cesium.com/learn/cesium-unreal/ref-doc/. [3 lis. 2026].
[13] Unity Technologies, “Unity Robotics Hub.” [Online]. https://github.com/Unity-Technologies/Unity-Robotics-Hub. [3 lis. 2026].
[14] Unity Technologies, “Unity ML-Agents Toolkit.” [Online]. https://github.com/Unity-Technologies/ml-agents. [3 lis. 2026].
[15] Unity Technologies, “Industry Terms of Service.” [Online]. https://unity.com/legal/industry-terms-of-service. [3 lis. 2026].
[16] Godot Engine, “Godot XR Update, March 2026.” [Online]. https://godotengine.org/article/godot-xr-update-mar-2026/. [3 lis. 2026].
[17] Godot Engine, “Godot License.” [Online]. https://godotengine.org/license/. [3 lis. 2026].
[18] Open 3D Engine, “ROS 2 Gem: Robotics and Simulations.” [Online]. https://www.docs.o3de.org/docs/user-guide/interactivity/robotics/. [3 lis. 2026].
[19] Panda3D, “Panda3D: Open Source Framework.” [Online]. https://www.panda3d.org/. [3 lis. 2026].
[20] Cocos, “Cocos Creator XR Documentation.” [Online]. https://docs.cocos.com/creator/3.8/manual/en/xr/. [3 lis. 2026].
[21] Robot Locomotion Group, “Drake: Model-Based Design and Verification for Robotics.” [Online]. https://drake.mit.edu/. [3 lis. 2026].
[22] NVIDIA, “Isaac Sim 6.2: System Requirements.” [Online]. https://docs.isaacsim.omniverse.nvidia.com/6.2.0/installation/requirements.html. [3 lis. 2026].
[23] NVIDIA, “Isaac Sim 6.2: License FAQ.” [Online]. https://docs.isaacsim.omniverse.nvidia.com/6.2.0/common/license-faq.html. [3 lis. 2026].
[24] SpaceEngine, “SpaceEngine: License Information.” [Online]. https://spaceengine.org/manual/license. [3 lis. 2026].

[25] OpenSpace Project, “OpenSpace: About.” [Online]. https://www.openspaceproject.com/about/about.html. [3 lis. 2026].
[26] OpenSceneGraph, “OpenSceneGraph Documentation.” [Online]. https://openscenegraph.github.io/openscenegraph.io/. [3 lis. 2026].
[27] VulkanSceneGraph, “VulkanSceneGraph Project.” [Online]. https://github.com/vsg-dev/VulkanSceneGraph. [3 lis. 2026].
[28] Epic Games, “Unreal Engine End User License Agreement.” [Online]. https://www.unrealengine.com/eula/unreal. [3 lis. 2026].