Symulacja w Badaniach i Rozwoju
Vol. 9, No. 1-2/2018

Jarosław KOSZELA, Maciej SZYMCZYK
Wojskowa Akademia Techniczna,
Wydział Cybernetyki, Instytut Systemów Informatycznych
ul. gen. Witolda Urbanowicza 2, 00-908 Warszawa 46
E-mail: jaroslaw.koszela@wat.edu.pl, maciej.szymczyk@wat.edu.pl

Rozproszona symulacja wirtualna – Chmura Symulacyjna

1 Wprowadzenie

Nauka i technologia rozwijają się coraz szybciej. Świadczyć może o tym prawo Moore'a. Jednocześnie zwiększają się nasze możliwości, a także wymagania oraz potrzeby życia codziennego, społecznego i gospodarczego.

Rozwój technologii jednostek obliczeniowych ma swoją specyfikę. Taktowanie procesorów na rynku komercyjnym od dłuższego czasu nie przekracza 4-5 GHz. Spowodowane jest to właściwościami krzemu, z którego zbudowane są procesory. W związku z tym przyśpieszenie osiąga się, mnożąc rdzenie oraz organizowanie systemów w sposób umożliwiający współbieżność oraz rozproszenie [1].

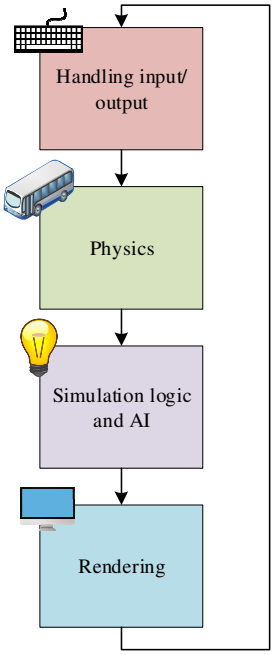
Wyróżniamy trzy rodzaje symulacji. Symulacja rzeczywista angażuje ludzi do operowania na fizycznie istniejących systemach według opracowanego scenariusza. Symulacja wirtualna angażuje ludzi do interakcji w ramach symulowanego komputerowo środowiska. Może wymagać umiejętności decyzyjnych, motorycznych, jak i komunikacyjnych. Symulacja konstruktywna jest to program komputerowy, którego działanie opiera się na zagregowanych obiektach symulacyjnych. Odzwierciedlają one te rzeczywiste w ramach kontekstu, którego dotyczy scenariusz symulacji [2].

Symulacja wirtualna będzie miała większą rozdzielczość oraz zgodność z rzeczywistym światem niż symulacja konstruktywna. Niestety w przypadku dużych, szczegółowych i/lub dynamicznych symulacji może okazać się, że moc obliczeniowa pojedynczej maszyny jest niewystarczająca. Jedną z możliwości jest przejście na symulację konstruktywną. Niestety agregacja wielu obiektów w jeden powoduje utratę szczegółów oraz konieczność kalibracji jego właściwości. Istnieje możliwość rozproszenia symulacji wirtualnej na wiele komputerów/węzłów obliczeniowych. Dzięki temu nie jesteśmy ograniczeni przez moc obliczeniową jednej maszyny.

2 Aktualne podejście

Pętla gry

Silnik każdej gry komputerowej oraz symulacji wirtualnej opiera się na pewnego rodzaju pętli. Podczas pojedynczego wykonania pętli obsługiwane są urządzenia wejścia, obliczana fizyka, logika, sztuczna inteligencja oraz renderowana grafika wyświetlana na ekran. Kroki te w klasycznym podejściu są sekwencyjne. Dopiero po zakończeniu jednego może zacząć się następny.



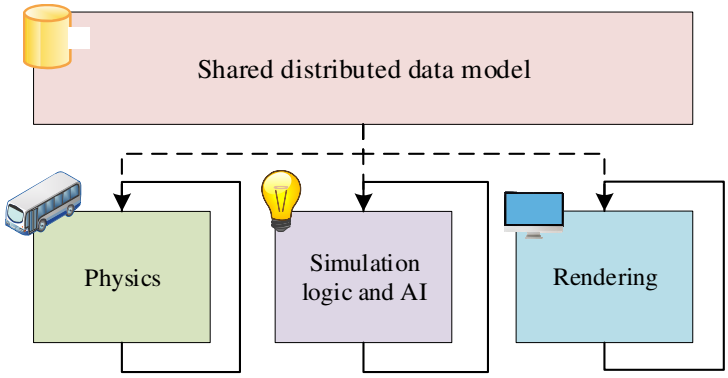
Rys. 1. Sekwencyjny model silnika gry. Źródło: opracowanie własne
Fig 1. Sequential game engine model. Source: own preparation

Aby zachować płynność rozgrywki, klatki na ekranie monitora muszą być wyświetlane z określoną częstotliwością. Aby uzyskać 50 klatek na sekundę, obliczenia związane z pojedynczym przebiegiem pętli muszą trwać nie więcej niż 20 milisekund. Akceptowalną dolną granicą dla gier jest 16 klatek [3]. W przeciwnym przypadku rozgrywka nie będzie płynna. Prezentuje to zestawienie liczby obiektów i renderowanych klatek na sekundę w tabeli 1. Dane otrzymane zostały z projektu „Pirates" platformy SpatialOS wykonanego na maszynie wirtualnej o parametrach 2 rdzenie procesora i7 4790k 4 Ghz i 4 GB pamięci RAM.

Tab. 1. Zestawienie liczby obiektów i klatek na sekundę dla pojedynczej maszyny
Table 1. List of the number of objects and frames per second for a single machine

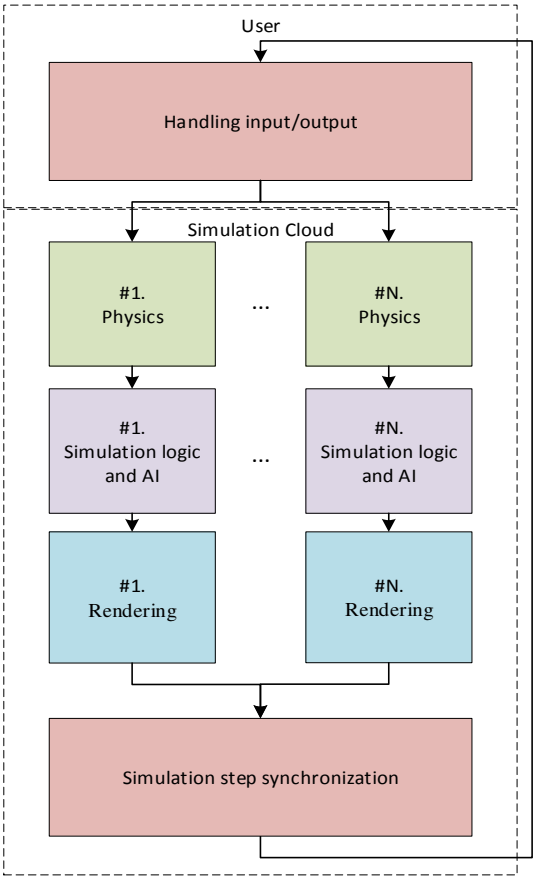
Number of objects	Frame time (ms)	FPS
80	20	50
130	20	50
280	27	37
400	50	20
530	357	2,8
1030	769	1,3

W momencie wprowadzenia architektury procesorów wielordzeniowych okazało się, że sekwencyjna pętla silnika gry jest niewydajna. Idealnym rozwiązaniem jest pętla asynchroniczna, w której zadania nie muszą czekać na wyniki innych zadań. Zamiast tego brany jest pod uwagę najnowszy obliczony wynik pobrany z wspólnego zasobu. Niestety model ten jest trudny do zaimplementowania w praktyce. Pewne zadania muszą być wykonywane sekwencyjnie.



Rys. 2. Asynchroniczny model silnika gry. Źródło: opracowanie własne
Fig 2. Asynchronous game engine model. Source: own preparation

Aby umożliwić zrównoleglenie zadań w pętli silnika gry, można podzielić dane w równoległych sekcjach aplikacji. Zamiast używania jednej głównej pętli, osobne wątki przetwarzają zestawy danych. Elementem ograniczającym ten model są zadania, które muszą zostać wykonane sekwencyjnie oraz potrzeba wymiany komunikatów między obiektami z różnych sekcji [4]. Zdarzenia powinny mieć możliwość rozprzestrzeniania się. Przykładem takiego zjawiska jest symulacja rozprzestrzeniania się zanieczyszczeń [5].



Rys. 3. Asynchroniczny model silnika gry z podziałem danych. Źródło: opracowanie własne
Fig 3. Asynchronous game engine model with data partition. Source: own preparation

Symulacja rozproszona

W przypadku gier komputerowych najczęściej wykorzystywaną architekturą w rozgrywce multiplayer jest *peer-to-peer* oraz klient-serwer.

Główną zaletą *peer-to-peer* jest brak centralnego serwera. Koszt utworzenia i utrzymania takiej sieci jest niski. Niestety rozwiązanie to nie jest skalowalne, prędkość łączy ogranicza wydajność, a implementacja jest skomplikowana. Niska wydajność łącza jednego z klientów może mieć wpływ na opóźnienia w rozgrywce pozostałych. Występują problemy z synchronizacją oraz konieczna jest stała liczba klatek dla wszystkich klientów. Przy zastosowaniu P2P utrudnione jest zapobieganie oszustwom.

Architektura klient-serwer polega na wykorzystaniu konkretnej maszyny, która będzie pełnić rolę serwera i udostępniać klientom usługi. Użytkownicy korzystają z aplikacji

klienckich komunikujących się z serwerem. Wymieniają się komunikatami, a otrzymują listę zmian do zrealizowania na lokalnym wirtualnym świecie [6].

Serwer może być pośrednikiem w wymianie komunikatów między klientami lub czynnie brać udział w prowadzeniu rozgrywki. W takim przypadku to serwer decyduje o zdarzeniach, które mają miejsce (np. który gracz pierwszy oddał strzał). Zapewnia to ochronę przed oszustwami.

Utrudnieniem w czerpaniu przyjemności oraz zapewnieniu realizmu rozgrywki są opóźnienia spowodowane przez infrastrukturę sieciową. Efektem ich są różnice stanów obiektów na serwerze gry i na urządzeniu klienckim. Rozwiązaniem tego problemu jest wykorzystanie mechanizmu predykcji, który szacuje przyszłe właściwości obiektu na podstawie aktualnych danych (np. pozycja, prędkość, kierunek).

Architektura tradycyjnych gier sieciowych zakładała jeden serwer w architekturze klient-serwer. W przypadku gier MMO RPG, czyli bardzo rozbudowanych i skomplikowanych gier obsługujących setki klientów, jeden serwer nie jest w stanie obsłużyć takiej ilości połączeń i danych. W takich przypadkach stosuje się wiele serwerów. Serwery te mogą być niezależne (stan o postępach w grze jest przechowywany na każdym serwerze osobno bez synchronizacji) lub ze wspólną bazą danych. W pierwszym przypadku użytkownik ma możliwość wyboru serwera, do którego chce się zalogować. W drugim przypadku natomiast występują automaty równoważące ruch na serwerach i to one przydzielają użytkownika do najmniej obciążonej maszyny [7].

W kontekście symulatorów najczęściej wykorzystywane są dwie architektury: *Distributed Interactive Simulation* (DIS) oraz *High Level Architecture* (HLA).

DIS jest to standard wymiany informacji między symulatorami. Nie posiada żadnego centralnego serwera zarządzającego. Symulatory mogą dołączać się i opuszczać symulację w każdym momencie. Stan symulacji jest zapisywany w wiadomości zwanej jako *Protocol Data Unit* (PDU) i wymieniany pomiędzy symulatorami poprzez dostępny protokół warstwy transportowej, wykorzystując multicast lub broadcast. Aktualna wersja DIS 7 definiuje 72 różne typy PDU [8].

High Level Architecture jest to architektura przeznaczona dla systemów komputerowych (w szczególności systemów symulacyjnych). Ideą tego rozwiązania jest uniezależnienie komunikacji od platform, na których znajdują się systemy. Symulator zgodny z HLA nazywany jest federatem. Systemy te połączone są do *Runtime Infrastructure* (RTI) i razem tworzą Federacje. Elementy systemu przekazują między sobą dane o obiektach, które znajdują się w *Federation Object Model* (FOM). Pomiędzy symulatorami przesyłane są również zdarzenia (interakcje) posiadające parametry [9].

Niestety zastosowanie wymienionych architektur dla wielu symulatorów ma jedną istotną wadę. Logicznie ten sam wirtualny świat symulowany jest na każdym federacie. Głównym celem architektur DIS i HLA jest synchronizacja różnych typów symulatorów. Nie umożliwia rozdzielenia wirtualnego świata pomiędzy symulatory.

3 Chmura symulacyjna

Wizualizacja w chmurze

Zastosowanie w symulacji wirtualnej rozwiązań wziętych z chmur obliczeniowych

sugeruje zastosowanie cienkich klientów. Aktualnie stosuje się grubych klientów, wyposażonych w szybkie procesory oraz karty graficzne.

Wykorzystanie cienkich klientów możliwe jest tylko w sytuacji, w której grafika dla każdego z klientów przetwarzana byłaby po stronie serwera. W środowisku gier komputerowych nazywa się to *cloud gaming*. Usługę taką oferuje między innymi NViDIA pod nazwą NVIDIA GRID.

Rozwiązanie to polega na przesyłaniu od klienta do serwera danych urządzeń wejścia. Klient otrzymuje natomiast strumień wideo, który jest dekodowany i wyświetlany na ekranie. Uniezależnia to grę lub symulację od platformy, jaką posiadać musi klient [10].

Węzły obliczeniowe

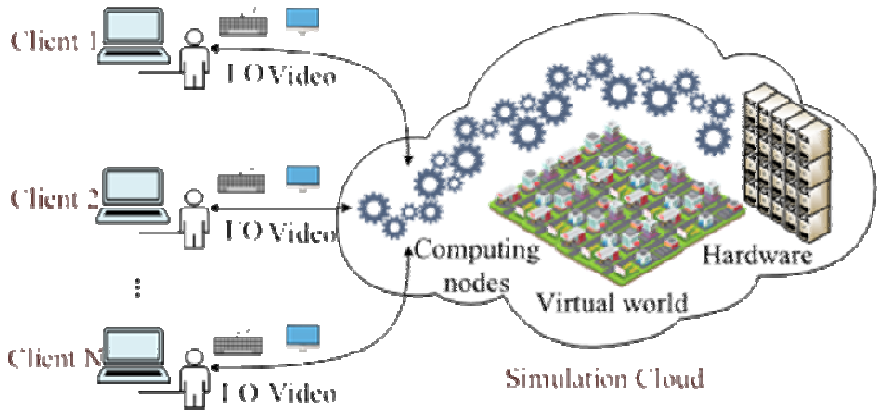
Jedną z cech, które musi posiadać chmura symulacyjna, jest silnik działający w sposób asynchroniczny oraz umożliwiający skalowanie w sposób horyzontalny. W ten sposób można inicjalizować węzły obliczeniowe w zależności od potrzeb. Ważne jest, aby każdy z węzłów mógł objąć swoją odpowiedzialnością różne obiekty symulacyjne oraz umożliwić interakcję między obiektami obsługiwanymi przez inne węzły.

Obiekty można przypisywać do węzłów obliczeniowych terytorialnie. Gdy obiekt znajdzie się na obszarze obsługiwanym przez dany węzeł, zostaje przez niego obsługiwany. Rozmiary obszarów mogą być różne. Przykładowo miasta będą gęściej zapełnione obiektami symulacyjnymi niż lasy.

Inicjalizacja oraz przydzielanie węzłów mogą być statyczne, jak i dynamiczne. Przeprowadzając symulację oraz analizę, można opracować zadowalający rozkład przydziału węzłów obliczeniowych. Operację tę należy powtórzyć dla każdego scenariusza. Przypisywanie dynamiczne polega na ciągłej analizie obciążenia węzłów obliczeniowych, powoływania nowych węzłów oraz zwalniania ich. Mechanizm ten pozwala na skalowanie zasobów zależnie od złożoności symulacji [11,12].

Zaletą dynamicznego przydzielania jest elastyczność oraz oszczędność zasobów chmury w momencie, gdy ich użycie nie jest konieczne. Z drugiej strony niesie za sobą obciążenie związane z próbkowaniem oraz analizą obecnego i przewidywanego obciążenia węzłów. Niewątpliwą zaletą dynamicznego przydzielania węzłów obliczeniowych jest większa odporność na awarie. W przypadku niespodziewanej utraty funkcjonalności węzła inicjowany jest nowy, a symulacja trwa dalej. Wykorzystanie chmury symulacyjnej jako usługi umożliwia płynne powoływanie nowych instancji oraz kontrolowanie kosztów symulacji [13].

Oprócz standardowych można zastosować dedykowane, specjalistyczne jednostki obliczeniowe. W prostym modelu silnika gry wyróżnić można obsługę fizyki, logiki oraz sztucznej inteligencji. Dedykowane pod każdy typ zadań węzły mogą wykorzystywać specjalistyczne karty graficzne, procesory lub komponenty wspierające sztuczną inteligencję i *machine learning* [14].



Rys. 4. Model chmury symulacyjnej. Źródło: opracowanie własne
Fig 4. Simulation cloud model. Source: own preparation

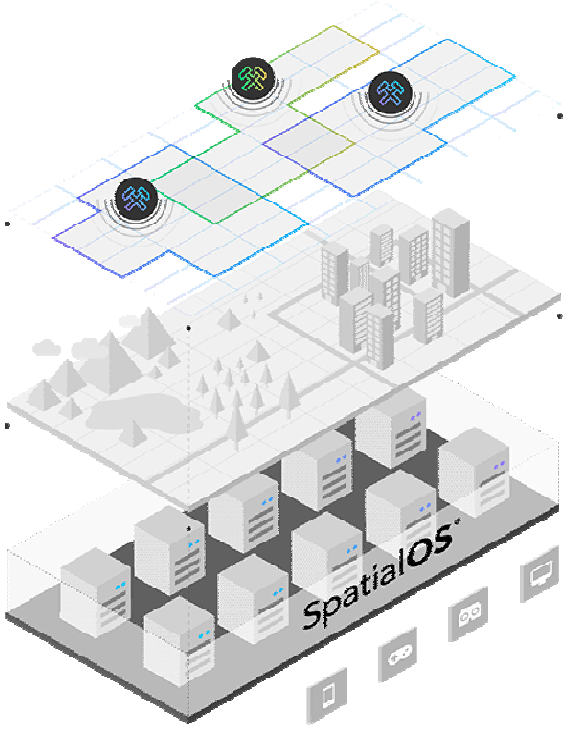
SpatialOS

Jednym z komercyjnych produktów wykorzystujących rozproszone przetwarzania w chmurze obliczeniowej dla symulacji wirtualnych jest platforma SpatialOS firmy Improbable. Rozwiązanie swoje opierają na Google Cloud Platform.

Symulacja wirtualna w SpatialOS obsługiwana jest przez węzy obliczeniowe zwane workerami. W przypadku zwiększenia potrzeb obliczeniowych dla symulacji powoływane są nowe instancje węzłów obliczeniowych. Wyróżnia się dwa rodzaje workerów : zarządzalne (*managed*) i zewnętrzne (*external*). Workery zarządzalne są to takie, których cykl życia zarządzany jest przez SpatialOS. Workery zewnętrzne to zazwyczaj klienci. SpatialOS nie ma wpływu na to, kiedy zewnętrzny worker się podłączy lub odłączy.

Wszystkie rodzaje workerów połączone są z platformą SpatialOS. W przypadku wystąpienia zdarzenia w jednym z węzłów (kliencie lub workerze) przesyłane jest ono do SpatialOS, który informuje o tym resztę węzłów. Tylko workery mogą dokonać zmian w wirtualnym świecie. Po stronie klienckiej zdarzenia obsługiwane są tylko w celu wyświetlenia modeli 3D, tekstur i animacji. Każdy worker zajmuje się tylko częścią wirtualnego świata. Zarządzalne workery zajmują się obliczeniami nad częścią świata, jaką przydzieliła im platforma SpatialOS. Zewnętrzne workery wiedzą tylko o części świata, jaka otacza obiekt symulacyjny (np. postać), z którym są związane.

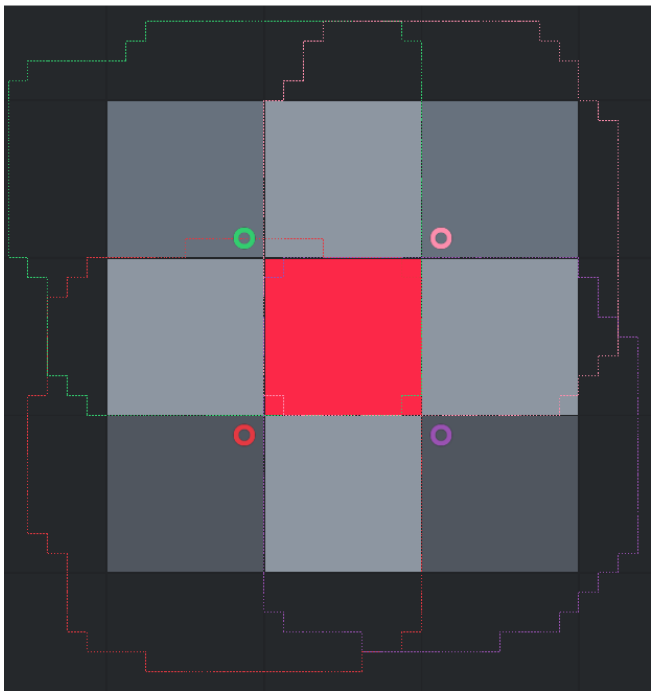
W ramach projektu można skonfigurować sposób organizacji węzłów obliczeniowych. W przypadku statycznego przydziału ustala się współrzędne węzłów oraz ich ilość. Dynamiczny przydział wymaga ustalenia maksymalnej liczby węzłów lub auto skalowania na podstawie maksymalnej ilości obsługiwanych obiektów [15].



Rys. 5. Model SpatialOS. Źródło: opracowanie własne
Fig 5. SpatialOS model. Source: own preparation

Testy wydajnościowe

Celem testów było rozpoznanie różnic wydajnościowych między symulacją opartą o 1, 2 lub 4 węzły obliczeniowe. Symulacja została oparta na platformie SpatialOS, z wykorzystaniem projektu „Piraci". W scenariuszu wykorzystano statyczną metodę przydzielania węzłów. Wyznaczono 4 pozycje węzłów na współrzędnych <-250,-250>, <-250,250>, <250,-250> oraz <250,250>. Obiekty umieszczane są losowo w promieniu 500 jednostek od pozycji <0,0>. Rozmieszczenie węzłów oraz obiektów prezentuje ilustracja nr 5.



Rys. 6. Rozmieszczenie węzłów obliczeniowych oraz gęstość obiektów symulacyjnych. Źródło: opracowanie własne

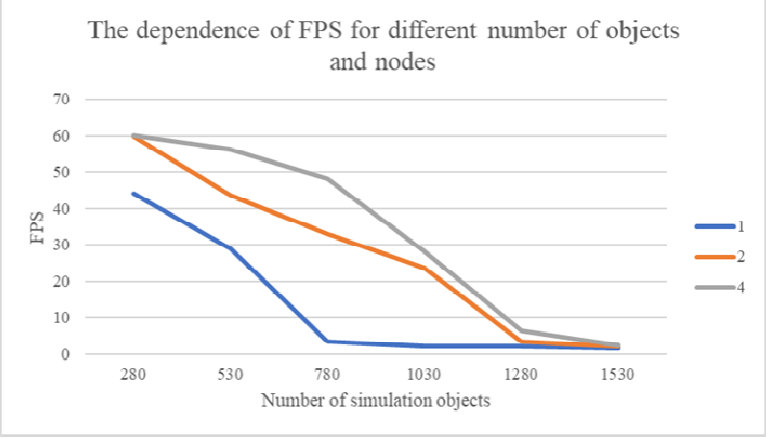
Fig 6. Arrangement of computing nodes and density of simulation objects. Source: own preparation

Badanie przeprowadzono w konfiguracji 1, 2, 4 węzły oraz 280, 530, 780, 1030, 1280 i 1530 obiektów, z czego 30 z nich to statyczne elementy otoczenia. Reszta to pirackie statki płynące w losowo wybranym kierunku. Wyniki badań zamieszczone są w tabeli nr 2.

Tabela 2. Zestawienie liczby obiektów i klatek na sekundę dla wielu węzłów obliczeniowych
Table 2. List of the number of objects and frames per second for many computing nodes

Liczba obiektow symulacyjnych	Liczba węzłów obliczeniowych					
	1		**2**		**4**	
	FPS	Objs/ node	FPS	Objs/ node	FPS	Objs/ node
280	44	280	59,8	208	60	147
530	29	530	43,5	383	56,4	273
780	3,3	780	33	574	48,1	402
1030	2,2	1030	23,6	744	28,2	540
1280	2,1	1280	3,3	911	6,5	653
1530	1,5	1530	2	1086	2,4	773

Wyniki badania pokazują, że przyrost klatek na sekundę nie jest liniowy względem zwiększenia liczby węzłów obliczeniowych. Jeden węzeł przestaje być wystarczający przy 780 obiektach. Wykres prezentujący dane z tabeli nr 2 znajduje się na rysunku nr 5. Zastosowanie 4 węzłów umożliwiło płynną symulację na poziomie około 30 fps przy 1030 obiektach.

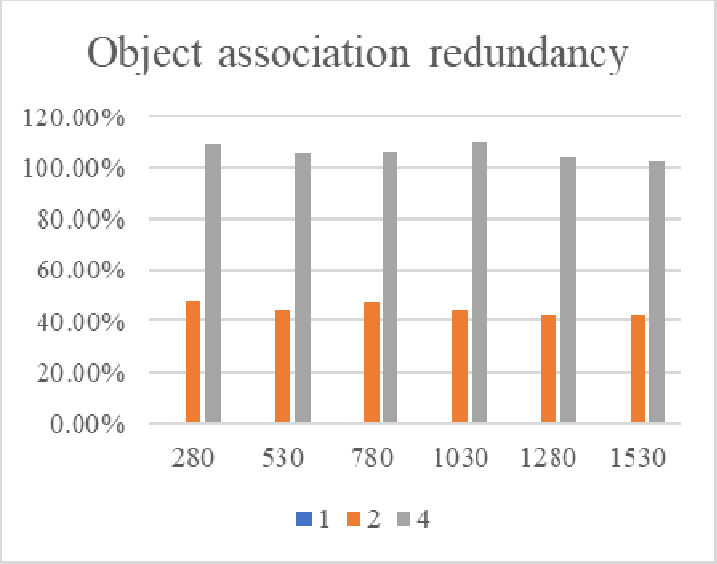


Rys. 7. Wykres zależności liczby klatek na sekundę dla ilości węzłów symulacyjnych. Źródło: opracowanie własne

Fig 7. The dependence of FPS for different numer of objects and nodes chart. Source: own preparation

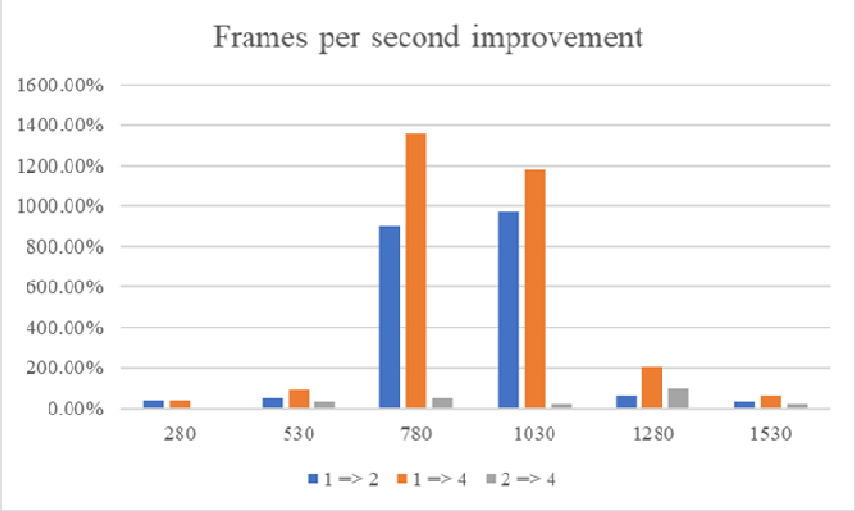
Największy przyrost FPS widoczny jest przy przejściu z 1 na 2 węzły. Najmniejszy przyrost FPS zaobserwować można przy przejściu z 2 na 4 węzły obliczeniowe. Zjawisko to widać na wykresie z rysunku nr 7. Wynika z tego, że zastosowanie

większej liczby węzłów niesie za sobą konieczność wykorzystania zasobów na ich obsługę i synchronizację.



Rys. 8. Wykres redundancji przypisania obiektów. Źródło: opracowanie własne
Fig 8. Object association redundancy chart. Source: own preparation

Niski przyrost wydajności prawdopodobnie spowodowany jest wysoką redundancją przypisania obiektów do węzłów, co widać na rysunku nr 6. W przypadku 4 węzłów obliczeniowych, suma wszystkich obiektów w węzłach przekracza dwukrotnie liczbę wszystkich obiektów w symulacji. Spowodowane jest to nakrywaniem się obszarów odpowiedzialności węzłów oraz dużą gęstością rozmieszczenia obiektów między wszystkimi węzłami. W przypadku równomiernego rozmieszczenia obiektów symulacyjnych redundancja będzie mniejsza, a wzrost FPS większy.



Rys. 9. Wykres zwiększenia klatek na sekundę. Źródło: opracowanie własne

Fig 9. Frames per second improvement chart. Source: own preparation

4 Wnioski

W artykule omówiono koncepcję wykorzystania rozproszonej symulacji wirtualnej. Przedstawione zostały problemy, z jakimi aktualnie mierzy się symulacja wirtualna w klasycznym wydaniu. Zaprezentowano rozwiązania wykorzystujące potencjał chmury obliczeniowej dla rozproszonej symulacji wirtualnej.

Wykorzystanie wielu węzłów wymaga poświęcenia mocy obliczeniowej na zarządzanie oraz wymianę komunikatów między nimi. Kierując się Prawem Amdahla, należy ograniczyć wykonanie kodu sekwencyjnego.

Istniejące rozwiązania można rozwinąć o renderowanie grafiki w chmurze (np. NVIDIA GRID). W artykule przedstawiono koncepcję oraz proste testy wydajnościowe platformy SpatialOS. Wykorzystanie środowiska rozproszonego umożliwiło podwojenie rozmiaru symulacji przy poczwórnej liczbie węzłów obliczniowych. W związku z powyższym wzrost wydajności jest znaczący, chociaż im większa jest liczba węzłów, tym mniejszy jest przyrost wydajności.

Do testów wykorzystano scenariusz, w którym obiekty nie zmieniały w krótkim czasie swojego położenia. W praktyce, zmieniające się wirtualne środowisko może powodować różnice w obciążeniach węzłów obliczeniowych. Dalszym kierunkiem badań będą metody i techniki przydzielania zadań (obiektów symulacyjnych) do węzłów obliczeniowych oraz określenie metryk wydajnościowych.

Literatura

1. Mattsson P. P.: *Why Haven't CPU Clock Speeds Increased in the Last Few Years?* https://www.comsol.com/blogs/havent-cpu-clock-speeds-increased-last-years/, dostęp 12.03.2018
2. Weatherly R. M., Wilson A. L., Canova B. S., Page E. H., Zabek A. A., Fisher M. C.: Advanced Distributed Simulation though the Aggregate Level Simulation Protocol, *Proceedings of the 29th Hawaii International Conference on Systems Sciences*, 407, 1996
3. Joselli M., Zamith M., Valente L., Feijó B., Leta F.R., Clua E.: A Distributed Architecture for Simulation Environments Based on Game Engine Systems, *Augmented Vision and Reality*, 4, 14, 2014
4. Multithreaded Game Engine Architectures, www.gamasutra.com/view/feature/130247/multithreaded_game_engine_.php?print=1, dostęp 12.03.2018
5. Tarapata Z., Antkiewicz R., Chmielewski M., Dyk M., Kasprzyk R., Kulas W., Najgebauer A., Pierzchała D., Rulka J.: A Computer System or CBRN Contamination Threats Analysis Support, Prediction Their Effects and Alarming the Population: Polish Case Study, *21st International Conference on Circuits, Systems, Communication and Computers*, 125, 2017
6. Cohen R., Ejlersen A., Kristensen R.: Distributed Game Engine for Massively Multiplayer Online Shooting Games, 10-12, 2009
7. Emilsson K.: Infinite Space: An Argument or Single-Sharded Architecture in MMOs, https://www.gamasutra.com/view/feature/132563/infinite_space_an_argument_for_.php?print=1 (dostęp 12.03.2018), 2018
8. *1278.1-2012 – IEEE Standard for Distributed Interactive Simulation—Application Protocols*, IEEE, 2012
9. *1516-2010 – IEEE Standard for Modelling and Simulation (M&S) High Level Architecture (HLA)—Framework and Rules*, IEEE, 2010
10. *NVIDIA GRID: Graphics accelerated VDI with the visual performance of a workstation* http://www.nvidia.com/content/grid/vdi-whitepaper.pdf (dostęp 12.03.2018)
11. Severiukhina O., Smirnov P. A., Bochenina K., Nasonov D., Butakov N.: Adaptive load balancing of distributed multi-agent simulations on heterogenous computational infrastructures, *6th International Young Scientists Conference in HPC and Simulation*, 2017
12. Bragard Q., Ventresque A., Murphy L.: Self-Balancing Decentralized DisAQtributed Platform for Urban Traffic Simulation, *IEEE Transactions on Intelligent Transportation Systems*, 18, 1190-1196, 2017
13. W. Xiong, W. Tsai, HLA-Based SaaS-Oriented Simulation Frameworks, *IEEE 8th International Symposium on Service Oriented System Engineering*, 1-7, 2014
14. *Google's dedicated TensorFlow processor, or TPU, crushes Intel, Nvidia in inference workloads* www.extremetech.com/computing/247199-googles-dedicated-tensorflow-processor-tpu-makes-hash-intel-nvidia-inference-workloads (dostęp 12.03.2018), 2018
15. *SpatialOS SDK Documentation*, docs.improbable.io (dostęp 12.03.2018), 2018

Streszczenie

Dzisiejszy sprzęt komputerowy posiada moc obliczeniową umożliwiającą przeprowadzanie symulacji wirtualnej. Nawet najmocniejsza maszyna w przypadku wykorzystania modelu o wysokiej szczegółowości oraz rozdzielczości może być niewystarczająca. Przejście na symulację konstruktywną spowoduje utratę na szczegółowości symulacji. Możliwe jest jednak zastosowanie rozproszonej symulacji wirtualnej w chmurze obliczeniowej. Przykładem zastosowania takiego rozwiązania jest produkt SpatialOS firmy Improbable.

Słowa kluczowe: symulacja rozproszona, VR, chmura obliczeniowa

Distributed virtual simulation – Simulation Cloud

Summary

Today's personal computers have the computing power to perform virtual simulation. However, even the most powerful machine in the case of using a high detail model and resolution may be insufficient. The transition to constructive simulation will result in a loss of detail in the simulation. However, it is possible to use a distributed virtual simulation in the computing domain. An example of such a solution is the SpatialOS product of the Improbable company.

Keywords: distributed simulation, VR, cloud computing