JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 5 (2017) 167-173
Wysłane: 2017-06-21
Przyjęte: 2017-06-28

Analiza wydajnościowa silnika Unity3D w kontekście aplikacji uruchamianych w przeglądarkach internetowych

Przemysław Berdak*, Małgorzata Plechawska-Wójcik
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Tematem artykułu są badania wydajnościowe silnika Unity3D do tworzenia aplikacji, które są uruchamiane z wykorzystaniem przeglądarki internetowej oraz technologii WebGL. W pracy wykonany został przegląd literaturowy z zakresu tworzenia aplikacji multimedialnych uruchamianych w przeglądarkach internetowych, silników graficznych, czy technologii WebGL. Dodatkowo omówione zostały zagadnienia techniczne związane z konwertowaniem aplikacji tworzonych w silniku Unity3D do WebGL. Realizacja badań oparta jest o symulację komputerową z użyciem pięciu scen w środowisku Unity3D, które zawierają główne możliwości tego silnika. Na końcu zaprezentowano wyniki owych badań oraz przedstawiono wnioski wyciągnięte na ich podstawie.

Słowa kluczowe: Unity3D; WebGL; Przeglądarki Internetowe; Silniki Graficzne

Autor do korespondencji *.

Adres e-mail: przemyslaw.berdak@gmail.com

Performance analysis of Unity3D engine in the context of applications run in web browsers

Przemysław Berdak*, Małgorzata Plechawska-Wójcik
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The aim of the work is the performance analysis of the Unity3D engine in the contex of applications running with a web browser and WebGL technology.. The paper presents literature review was carried out in the scope of creating multimedia applications run in web browsers, graphics engines and WebGL technology . Additionally, technical issues related with converting Unity3D application to WebGL technology are discussed. The research are performed based on computer simulations with five scenes created with Unity3D environment, which include the main capabilities of this engine. Finally, it was presented the results of these studies and the conclusions.

Keywords: Unity3D; WebGL; Web Browsers; Graphics Engines

Corresponding author *.

E-mail address: przemyslaw.berdak@gmail.com

1. Wstęp

Sieć Internet oraz przeglądarki internetowe stają się coraz popularniejszą platformą aplikacyjną, której można użyć do publikowania aplikacji multimedialnych, takich jak na przykład gry wideo. Przez lata istniały różne metody tworzenia takich aplikacji, lecz z różnymi możliwościami graficznymi czy fizycznymi oraz z róż nym stopniem skomplikowania. Od roku 2009 takie aplikacje można tworzyć z wykorzystaniem technologii WebGL, która rozszerza język JavaScript o elementy API do tworzenia grafiki bazującej na OpenGL ES. Taki podzbiór może zostać użyty na przykład przez bibliotekę graficzną, lecz może mieć ona zbyt mało możliwości, aby stworzyć nowoczesną, bogatą w wiele funkcji aplikację multimedialną. Rozwiązaniem może być użycie silnika graficznego, który oferuje bardzo bogate możliwości rozwoju, nawet bardzo skomplikowanych aplikacji, lecz do niedawna mogły być one uruchamiane tylko desktopowo. Od maja 2015 roku silnik Unity3D oferuje możliwość przekonwertowania nawet bardzo skomplikowanych aplikacji do technologii WebGL, dzięki czemu mogą być one uruchomione z wykorzystaniem przeglądarki internetowej. Takie aplikacje, według twórców Unity3D, jakościowo stoją na bardzo zbliżonym, o ile nie na takim samym poziomie, co aplikacje natywne generowane do kodu odpowiadającego danej platformie.

Z faktu, że takie rozwiązanie jest stosunkowo nowe pojawia się konieczność zbadania jego wydajności, aby określić możliwości poprawnej pracy pod różnymi przeglądarkami internetowymi. Z tego też powodu głównym celem pracy dyplomowej, której dotyczy niniejszy artykuł jest zbadanie wydajności silnika Unity3D w powyższym zakresie i wyciągnięcie wniosków dotyczących możliwości jego stosowalności w kontekście przeglądarek internetowych. Realizacja tak omówionego celu polegała na wygenerowaniu pięciu scen, które były badane w powyższym kontekście.

2. Badania literaturowe

Badania literaturowe rozpoczęto od artykułu omawiającego przegląd głównych sposobów renderowania grafiki 3D poprzez przeglądarkę internetową. Takim artykułem był *3D graphics on the web: A survey* autorstwa *Aluna Evansa, Maco Romeo, Arasha Behrehmanda, Javiego Agenjo i Josepa Blata.*[1] Autorzy przywołali tutaj podział

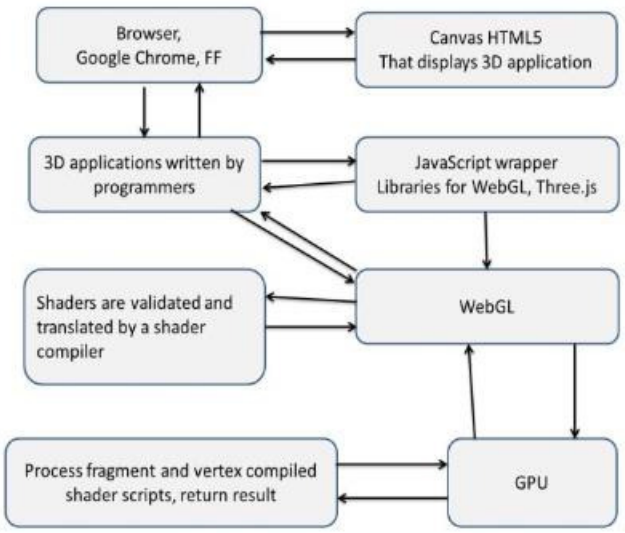
sposobu tworzenia grafiki wyświetlanej przez przeglądarkę internetową na imperatywny i deklaratywny, który może być stosowany do grafiki 2D oraz 3D. Grafika deklaratywna i imperatywna może dotyczyć zarówno grafiki dwuwymiarowej jak i trójwymiarowej. W przypadku grafiki dwuwymiarowej podejściem deklaratywnym może być tworzenie grafiki wektorowej, natomiast podejściem imperaktywnym rysowanie na elemencie *Canvas*. W opisie imperatywnego sposobu renderowania grafiki trójwymiarowej autorzy tego artykułu przywołują przykład pluginu Adobe Flash, który pozwala na uruchomienie zawartości multimedialnej wewnątrz strony internetowej. Poza technologią Adobe Flash przywołano tutaj inne sposoby tworzenia aplikacji uruchamianych w przeglądarkach internetowych, takich jak Microsoft Silverlight, czy Google O3D. Publikacja kończy się nawiązaniem do renderowania grafiki z wykorzystaniem elementu Canvas, takie podejście może być, według autorów, traktowane jako prekursor technologii WebGL.

Uzupełnieniem powyższego może być artykuł *On the Development of Browser Games - Technologies of an Emerging Genre* Juha-Mattiego Vanhatupa [2]. Tutaj też przedstawiono przegląd poszczególnych technologii umożliwiających uruchomienia aplikacji z użyciem przeglądarki internetowej. Technologie Adobe Flash, czy Microsoft Silverlight zostały przez autora uzupełnione przez Shockwave, czy plugin Unity Web Player.

Takie technologie bardzo często nie wykorzystywały możliwości silników graficznych, do czego wyjątkiem jest wspomniany plugin Unity Web Player, dlatego też w ramach badań literaturowych przyjrzano się bliżej zagadnieniu związanym z silnikami graficznymi. Ciekawą pracą naukową mogącą rzucić więcej światła na dane zagadnienie jest *Graphics Engine Technology Research and Implementation* autorstwa Liu Yijun, Chen Wenbin i He Xiaoman [3]. Publikacja ta została opublikowana jako wynik prac związanych z opracowaniem konceptu i implementacji silnika graficznego. Poruszona jest tutaj kwestia istotności realistyczności grafiki w wielu aplikacjach, które mają za zadanie tworzyć animacje filmowe, rozszerzoną rzeczywistość, gry wideo, czy obiekty architektoniczne. Autorzy publikacji zauważają, że silnik graficzny składa się z wielu elementów i nie służy tylko do renderowania grafiki. Typowy silnik graficzny składa się również z silnika fizyki, systemu animacji, czy bibliotek modeli, obrazów oraz dźwięków. Jego kwintesencją jest jednak rysowanie obrazu dwuwymiarowego na podstawie sceny trójwymiarowej, dlatego też ten element powinien być najlepiej dopracowany.

Dobry silnik graficzny musi być też dobrze zorganizowany, aby tworzone aplikacje wyglądały i działały możliwie jak najlepiej. W publikacji publikacji *A Survey of Frameworks and Game Engines for Serious Game Development,* której autorami są Brent Cowan oraz Bill Kapralos [4] przedstawiono kilka cech dobrego silnika graficznego, do których zaliczają się między innymi możliwość skryptowania obiektów, renderowania grafiki, animacje oraz właściwości fizyczne, a także edytory poszczególnych tych cech.

Dzięki publikacji *The The Web as an Application Platform: The Saga Continues* autorstwa Antero Taivalsaariego i Tommiego Mikkonena [5] można przejść z tematu silników graficznych do tematu użycia sieci Internet oraz przeglądarek internetowych jako platformy, która może zostać użyta do uruchamiania aplikacji. Publikacja skupia się na możliwości wykorzystania HTML5 i WebGL jako narzędzi eliminujących ograniczenia sieci i przekształcających sieć WWW w platformę aplikacyjną, gdyż według autorów w przyszłości większość oprogramowania dostarczana będzie właśnie przez sieć Internet.



Rys. 1. Stos WebGLa według Rovshena Nazarova i John Galletly [6]

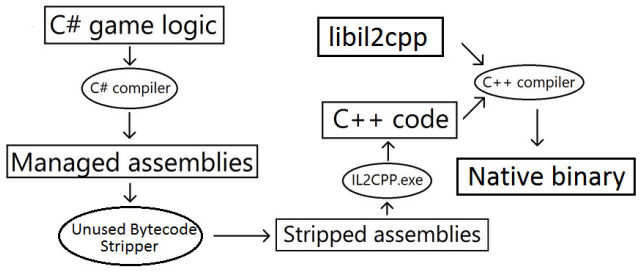
Możliwość wykorzystania technologii WebGL i HTML5 do tworzenia aplikacji multimedialnych w swojej pracy *Native browser support for 3D rendering and physic using WebGL, HTML5 and Javascript* poruszają też Rovshen Nazarow oraz John Galletly [6], którzy zaprezentowali sposób wyświetlania grafiki z użyciem tych technologii. W pracy tej zostało dokładniej omówione zagadnienie WebGL, natomiast autorzy przedstawili również stos tej technologii widoczny na rysunku numer 1.

3. Opis obiektu badań

Silnik Unity3D jest jednym z najpopularniejszych silników graficznych przeznaczonych do tworzenia trójwymiarowych gier wideo oraz innych materiałów. Silnik ten dostarcza szereg komponentów dla twórców gier, aby mogli oni zamodelować to co chcą z możliwie największą dokładnością, a sam proces był możliwie najłatwiejszy. Silnik ten dostarcza praktycznie wszystkie elementy konieczne do tworzenia efektownych aplikacji, co czyni go jednym

z najpopularniejszych silników na świecie. Poza typowymi elementami, takimi jak możliwość tworzenia scen, grafiki, shaderów, czy animacji silnik Unity dostarcza możliwość tworzenia skryptów, które można napisać w języku C#, bądź JavaScript [7].

Wszystkie elementy aplikacji są jednak tłumaczone na natywny kod obsługiwanej platformy, czyli jednej z dwudziestu dwóch, które obsługuje Unity3D. Aby kod aplikacji został przetłumaczony na jedną z tych platform musi istnieć mechanizm, który kod aplikacji przetłumaczy na odpowiedni język, najczęściej C/C++. Takim mechanizmem jest kompilator IL2CPP, który służy do przetwarzania kodu napisanego w języku C# do języka właśnie C/C++, proces ten przedstawiono na rysunku numer 2. [8]



Rys. 2. Diagram budowy kodu C++ na podstawie C# z wykorzystaniem technologii IL2CPP [8]

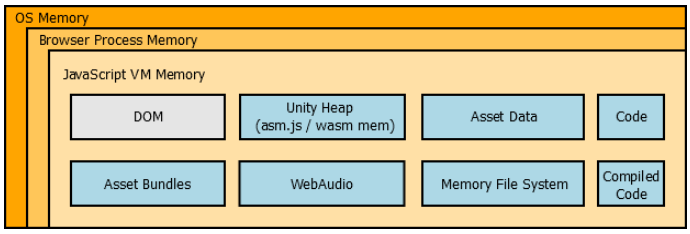
Tak powstały kod jest dodatkowo kompilowany do języka asm.js, który jest podzbiorem języka JavaScript. Został on opracowany aby wydajnościowo zbliżyć język JavaScript do poziomu kodu natywnego i według autorów kod natywny jest tylko 1.5 razy szybszy. Aby kod powstały powyżej mógł zostać skompilowany do kodu asm.js wykorzystywany jest kompilator Emscripten oraz kod LLVM, a schemat postępowania w przypadku takiego kodu jest widoczny na rysunku numer 3. [9]

Rys. 3. Schemat kompilacji kodu C/C++ do JavaScript [9]

Tak powstały kod w języku JavaScript wykorzystuje następnie technologię WebGL, która dostarcza API do interfejsu wyświetlania grafiki bazującego na OpenGL ES, który jest wykorzystywany na elemencie Canvas HTML5. [9]

Jako, że WebGL bazuje na JavaScripcie, jego wydajność jest bezpośrednio powiązana z prędkością tego języka i choć jeszcze kilka lat temu jego wydajność była niezadowalająca, dzisiaj może być używany do tworzenia wydajnych aplikacji renderujących grafikę trójwymiarową. Z powodu swojej popularności standard ten jest obecnie wspierany w różnym stopniu przez niemal wszystkie przeglądarki internetowe, zarówno te desktopowe, jak i mobilne. Na rysunku numer 4 widoczne są wszystkie komponenty pamięci, które powstają dla aplikacji WebGL wygenerowanej za pomocą Unity3D. [9]

Jednym z większych problemów podczas tworzenia aplikacji tworzonych z użyciem silnika Unity3D, które mają zostać uruchomione w przeglądarce internetowej jest problem z niewystarczającą ilością pamięci. Użytkownicy końcowi mogą uruchomić aplikację na różnych przeglądarkach, na różnych urządzeniach oraz na różnych systemach operacyjnych. W związku z tym konieczne jest nie tylko odpowiednie zarządzanie pamięcią, ale także odpowiednia kompresja zasobów aplikacji [10].



Rys. 4. Rozmieszczenie poszczególnych elementów aplikacji Unity WebGL w przeglądarce internetowej [10]

Niestety problemy z pamięcią nie są jedynymi problemami aplikacji tworzonymi z użyciem silnika Unity3D i konwertowanych do WebGL. Spośród nieobsługiwanych przez WebGL elementów systemu Unity, który najbardziej może zaważyć na efektywności takich aplikacji jest brak wielowątkowości. Spowodowane jest to brakiem wątków w języku JavaScript, do którego konwertowane są zasoby Unity. Dodatkowo mogą wystąpić pewne limity związane z wyświetlaniem grafiki spowodowane limitami w OpenGL ES 2.0, a także limity związane z bardziej zaawansowanym audio. Z ograniczeń ściśle programistycznych nie ma możliwości użycia programowania refleksyjnego.[9]

Tak tworzona i uruchamiana aplikacja jest poddawana badaniom w pracy dyplomowej, na bazie której opracowana została dana publikacja.

4. Opis metodyki badawczej

Badanie zostało przeprowadzone z wykorzystaniem eksperymentu polegającego na wykorzystaniu symulacji komputerowej. Metodyka badawcza zakłada zdefiniowanie pięciu scenariuszy badawczych, które podlegają symulacji komputerowej. Wyniki takiej symulacji są porównywane z symulacją przeprowadzoną dla aplikacji wygenerowanych do kodu natywnego oraz jako aplikacja mobilna.

Badanymi elementami silnika Unity3D są:

- Animacje
- Grafika i shadery
- System cząstek
- Trójwymiarowa fizyka obiektów
- Generowanie i usuwanie obiektów

Aby przeprowadzić taką analizę należy wygenerować kilka scen, które będą zawierać elementy dotyczące każdego z powyższych kryteriów. Każda tak wygenerowana scena będzie oddzielnym scenariuszem testowym. Takie sceny powinny być uruchamiane oddzielnie na platformie testowej.

Dodatkowo badanie jest przeprowadzane pod różnymi

przeglądarkami internetowymi.

Wśród badanych przeglądarek wyróżniono:

- Mozilla Firefox 52
- Google Chrome 58
- Microsoft Edge 38
- Opera 45

Aby porównać wydajność badanej technologii badania są przeprowadzane również dla scen wygenerowanych dla aplikacji natywnej uruchamianej na laptopie oraz dla natywnej aplikacji natywnej uruchamianej na telefonie komórkowym. Platformą testową jest laptop z procesorem Intel Core i3 2370M 2,4GHz, 4 GB RAM DDR3 i kartą graficzną AMD Radeon HD 7670M + Intel HD Graphics 3000, natomiast użytym sysemem operacyjnym był Windows 10. Do badań nad aplikacją natywną wykorzystano smartfon LG G3 z systemem operacyjnym Google Android 6.0, procesorem czterordzeniowym Qualcomm Snapdragon 801 2.50 GHz, układem graficznym Adreno 330 i GB pamięci RAM.

Metodyka badawcza wymaga też zdefiniowania kryteriów porównawczych, które opiszą przeprowadzony eksperyment. Wśród tych zdefiniowanych znajdują się:

- Czas całkowitego załadowania aplikacji (sekundy)
- Średnia liczba klatek na sekundę (klatki na sekundę)
- Zajętość pamięci w systemie operacyjnym przeglądarki z aplikacją, bądź samej aplikacji w przypadku aplikacji natywnych (megabajty)
- Procent zużycia procesora (procent)

Metodyka badawcza zakłada także opracowanie wyników eksperymentu z wykorzystaniem zagadnienia analizy wielokryterialnej. Do analizy została wybrana Metoda Sumy Ważonej, natomiast normalizacja wyników została przeprowadzona Metodą Unitaryzacji Zerowej.

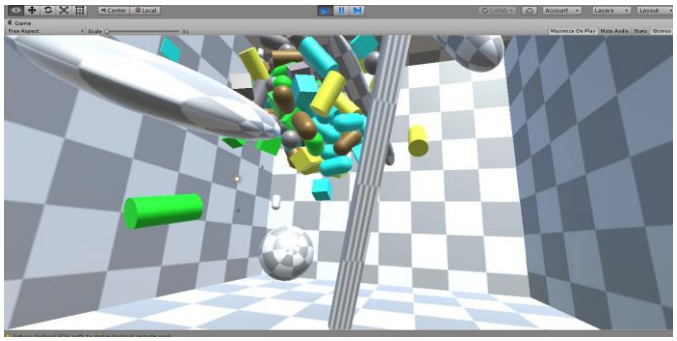
5. Model symulacji

Ze względu na dobrane kryteria oraz zdefiniowane sceny na potrzeby badań użyto pięciu różnych aplikacji, z których każda była odpowiedzialna za zbadanie innego komponentu silnika Unity3D w kontekście aplikacji uruchamianych w przeglądarkach internetowych.

Przechodząc do poszczególnych scen pierwszą z nich jest scena bazująca na animacjach. Aby wygenerować taką scenę należy wykorzystać kilka modeli, które wykonują jak najwięcej ruchów oraz animacji. Do sceny dodane zostało pięć głównych modeli z systemem animacji, a wszystkie te modele zostały następnie skopiowane pewną liczbę razy tak, aby na scenie było możliwie jak najwięcej animowanych obiektów. Do sceny dodana została trawa, na którą działają silne podmuchy wiatru, aby dołożyć do sceny jeszcze więcej animacji. Gotowa scena jest widoczna na rysunku piątym.

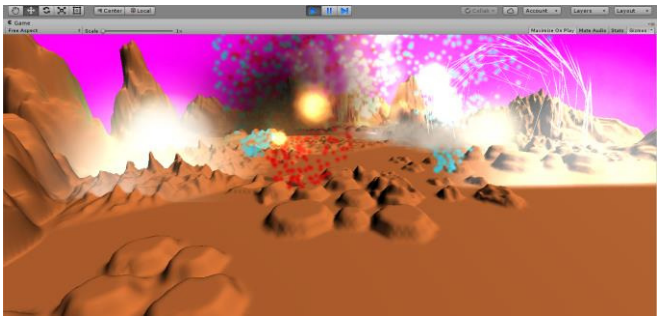
Rys. 5. Scena bazująca na animacjach

Drugą ze scen jaka zostanie przygotowana jest scena, której zadaniem jest zbadanie właściwości fizyki trójwymiarowej w silniku Unity3D. Dobrym pomysłem na zbudowanie takiej sceny jest utworzenie wielu prostych komponentów, a następnie dodaniu im różnych właściwości fizyki trójwymiarowej by w końcu wszystkie te zderzały się ze sobą. Wszystkie elementy znajdują się w kostce sześciennej, natomiast w czasie działania aplikacji zderzają się ze sobą. Wszystkie kolorowe elementy mają tutaj inne właściwości fizyczne, Scena jest widoczna na rysunku numer 6.



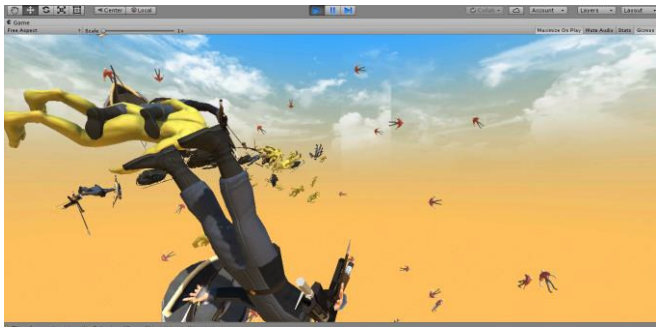
Rys. 6. Scena bazująca na fizyce trójwymiarowej

Trzecią sceną była scena, której zadaniem było wykonanie badań z pod kątem wykorzystania systemu cząsteczek. Scena taka została zbudowana z wykorzystaniem standardowego terenu oraz głównie standardowego systemu cząsteczek. Na wygenerowany teren zostało nałożone możliwie jak najwięcej systemów cząsteczek. Jest ona widoczna na szóstym rysunku 7.



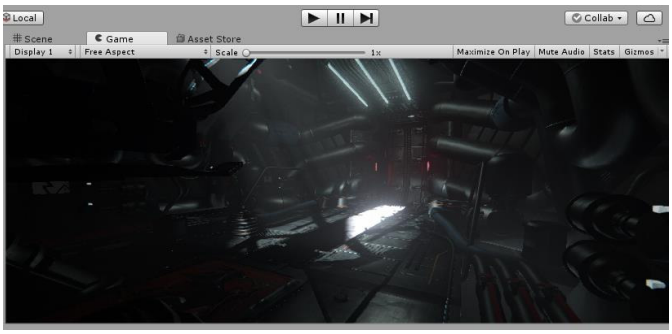
Rys. 7. Scena bazująca na systemie cząsteczek

Czwartą badaną sceną była scena badająca szybkość renderowania i usuwania obiektów gry. Do budowy sceny skorzystano głównie z tych samych elementów jak w przypadku sceny pierwszej oraz dodatkowego skryptu, którego zadaniem było zarządzanie generowaniem nowych i niszczeniem obiektów. Działanie tej sceny zostało zademonstrowane na rysunku 8.



Rys. 8. Scena bazująca na generowaniu i usuwaniu obiektów

Ostatnią z badanych scen jest scena, której zadaniem było zbadanie określenie wydajności aplikacji z zaawansowaną grafiką. W tym celu posłużono się sceną utworzoną przez *Unity Technologies* o nazwie *Adam Interior Scene*. Scena taka zawiera na tyle zaawansowane elementy graficzne, że idealnie nadaje się do badań. Została zaprezentowana na rysunku 9.



Rys. 9. Scena bazująca na grafice

6. Prezentacja rezultatów badań

Dla wszystkich omawianych scen zostały przeprowadzone badania. W tabeli pierwszej widoczne są wyniki badań dla sceny bazującej na animacjach w odniesieniu do zdefiniowanych kryteriów. Wyniki dla tej sceny zostały wybrane jako przykładowe, ponieważ są najbliżej średniej dla wyników ze wszystkich scen. Badania zostały przeprowadzone na systemie oraz sprzęcie przedstawionym w rozdziale czwartym. W tabeli widać, że aplikacje natywne osiągają większą liczbę klatek na sekundę od aplikacji przeglądarkowych oraz uruchamiają się zdecydowanie szybciej. Zużywają także mniej pamięci oraz mniej obciążają procesor.

Ocena możliwości wydajnościowych poszczególnych przeglądarek w kontekście aplikacji Unity WebGL uwzględniająca wszystkie wybrane kryteria wymaga przeprowadzenia analizy wielokryterialnej. Analiza taka może dać jednoznaczną odpowiedź na pytanie dotyczące analizy wydajnościowej silnika Unity3D w kontekście aplikacji uruchamianych w przeglądarkach internetowych

Tabela 1. Wyniki dla sceny bazującej na animacjach

		Czas załadowania [s]	Średnia liczba klatek na sekundę [fps]	Zajętość pamięci [mb]	Zużycie procesora [%]
Przeglądarki internetowe (Windows 10)	Mozilla Firefox 52	43.54	14	1828	26.7
	Google Chrome 58	45.26	13	2189	34.1
	Microsoft Edge 38	65.04	9	2305	33.3
	Opera 45	46.55	10	2158	22
Aplikacje natywne	Android	15.11	17	594	18
	Windows 10	10.48	21	835	14.8

W celu przeprowadzania analizy wielokryterialnej dane musiały zostać odpowiednio znormalizowane. Nastąpiło z użyciem Metody Unitaryzacji Zerowej. Metoda ta wymaga, aby każdy badany element został poddany normalizacji za pomocą stymulatora z życiem pierwszego wzoru poniżej albo destymulatora poprzez drugi z wymienionych niżej wzorów.

$$Z_{ij} = \frac{X_{ij} - \min(X_{ij})}{\max(X_{ij}) - \min(X_{ij})} \quad (1)$$

$$Z_{ij} = \frac{\max(X_{ij}) - X_{ij}}{\max(X_{ij}) - \min(X_{ij})} \quad (2)$$

Normalizacja została wykonana dla wszystkich otrzymanych wyników, natomiast tabela druga przedstawia wyniki normalizacji dla sceny bazującej na grafice.

Tabela 2. Normalizowane wyniki dla sceny bazującej na animacjach

		Czas załadowania	Średnia liczba klatek na	Zajętość pamięci	Zużycie procesora
Przeglądarki internetowe (Windows 10)	Mozilla Firefox 52	1	1	1	0.61
	Google Chrome 58	0.92	0,8	0.24	0
	Microsoft Edge 38	0	0	0	0.07
	Opera 45	0.86	0.2	0.31	1
Waga kryterium		0.3	0.4	0.2	0.1

Do przeprowadzenia analizy wielokryterialne konieczne jest ustalenie wag dla poszczególnych kryteriów. Przy ustalaniu wag uwzględniono, że dwoma kryteriami, które najbardziej wpływają na komfort korzystania z aplikacji jest liczba klatek na sekundę oraz czas załadowania aplikacji, dlatego też sumę ich wag ustalono na 0.7. Ważniejszym z tych dwóch jest kryterium określające liczbę klatek na sekundę, dlatego została mu przyznana waga 0.4, natomiast kryterium określającym czas załadowania 0.3. Przyglądając dwóm spośród pozostałych kryteriów zaobserwowano, że zbudowane aplikacje podczas działania procentowo wymagają większych zasobów pamięci niż procesora, dlatego też kryterium dotyczącym zajętości pamięci przydzielono wagę 02, natomiast kryterium związanym z zużyciem procesora przydzielona wagę 0.1.

Analiza wielokryterialna została wykonana Metodą Sumy Ważonej, gdzie były cztery główne kryteria zdefiniowane wcześniej oraz widoczne w tabeli pierwszej. Sama analiza została przeprowadzono z użyciem wzoru widocznego poniżej.

$$F(x) = \sum_{i=1}^{4} w_i f_i(x) \tag{3}$$

Wyniki analizy wielokryterialne są widoczne w tabeli trzeciej.

Tabela 3. Suma ważona jako wyniki analizy wielokryterialnej

	Animacje	Fizyka	System Cząsteczek	Gen/Usuw. obiektów	Grafika
Mozilla Firefox 52	0.961	0.3	0.571	0.751	0.108
Google Chrome 58	0.644	0.337	0.212	0.532	0.504
Microsoft Edge 38	0.007	0.506	0.477	0.035	0.6
Opera 45	0.5	0.4	0.4	0.742	0.57

Wyniki widoczne w tabeli trzeciej zostały dodatkowo uśrednione dla wszystkich kryteriów, aby zobaczyć która z przeglądarek najlepiej radzi sobie z uruchamianiem aplikacji Unity WebGL. Rezultaty badań na systemie operacyjnym Windows 10 są widoczne w czwartej tabeli. We wszystkich tabelkach wartości większe reprezentują lepszą wartość badanej cechy. Dodatkowo jako wyniki zaprezentowano wartości dla przeglądarek internetowych.

7. Dyskusja wyników i wnioski.

Otrzymane wyniki próbują odzwierciedlić jakość wykorzystania silnika Unity3D do budowy aplikacji, które mogą zostać uruchomione z użyciem przeglądarki internetowej. Porównując dane z przeglądarek internetowych można wskazać, że aplikacje natywne jednak lepiej radzą sobie z zadaną tematyką. Takie aplikacje z reguły osiągały lepsze wyniki od wszystkich przeglądarek internetowych. Wyniki aplikacji natywnych były lepsze we wszystkich badanych kryteriach, od czasu uruchomienia, poprzez liczbę klatek na sekundę aż do zasobności komponentów sprzętowych.

Tabela 4.Uśredniona suma ważona dla badanych przeglądarek

	Średnia suma ważona
Mozilla Firefox 52	0.5382
Google Chrome 58	0.4458
Microsoft Edge 38	0.325
Opera 45	0.5224

Żeby porównać wydajność poszczególnych przygotowanych scen oraz poszczególnych przeglądarek warto przejść do zestawienia zawierającego opracowanie wykonane z użyciem analizy wielokryterialnej. Można tutaj zauważyć, że przeglądarka Mozilla Firefox 52 najlepiej radzi sobie z animacjami poszczególnych obiektów. Jest to niezmiernie ważne, gdyż znaczna część tworzonych aplikacji przez silnik Unity3D zawiera obiekty animowane, których wydajność wpływa na ogólne wrażenia z uruchamianej aplikacji. Na bardzo podobnym poziomie z animacjami radzą sobie przeglądarki Opera 45 oraz Google Chrome 58. Najgorzej na tym polu wypada przeglądarka wydana przez firmę Microsoft, czyli Edge 38. Inna sytuacja jest z elementami graficznymi, gdyż jak widać w tabeli przeglądarka Firefox 52 jest tą, która najgorzej radzi sobie z tego typu elementami, natomiast wszystkie przeglądarki konkurencyjne wyglądają na tym polu lepiej.

Jednym z elementów, który miał wpływać na wydajność technologii Unity WebGL miał być język asm.js, do którego kompilowany jest finalny kod. Przeglądarka Firefox 58, której autorzy są jednocześnie współautorami tego rozwiązania osiągnęła w badaniu najlepszy wynik, co może powodować, że użycie tej technologii może wpłynąć na wydajność technologii Unity WebGL w przeglądarce internetowej. Technologia te jest również częściowo wspierana przez inne przeglądarki, dlatego ich wyniki nie są znacznie gorsze.

Podsumowując otrzymane wyniki można stwierdzić, że dla różnych przeglądarek wyniki były na bardzo podobnym poziomie, lecz póki co, pod względem wydajnościowym technologia Unity3D odstaje nieco od aplikacji przekonwertowanych do kodu natywnego.

Literatura

[1] A. Evans, M. Romeo, A. Bahrehmand, J. Agenjo i J. Blat, 3D graphics on the web: A survey, 2014.

[2] J.-M. Vanhatupa, On the Development of Browser Games - Technologies of an Emerging Genre, w 2011 7th International Conference on Next Generation Web Services Practices, Salamanca, Hiszpania, 2011.

[3] L. Yijun, C. Wenbin i H. Xiaoman, 3D Graphics Engine Technology Research and Implementation, w 2010 3rd IEEE International Conference, Chengdu, Chiny, 2010.

[4] B. Cowan i B. Capralos, A Survey of Frameworks and Game Engines for Serious Game Development, w 2014 IEEE 14th International Conference on Advanced Learning Technologies , Ateny, Grecja, 2014.

[5] A. Taivalsaari i T. Mikkonen, The Web as an Application Platform: The Saga Continues, w 2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications, Oulu, Finlandia, 2011.

[6] R. Nazarov i J. Galletly, Native browser support for 3D rendering and physics using WebGL, HTML5 and Javascript, w 6th Balkan Conference in Informatics, BCI-LOCAL 2013, Saloniki, Grecja, 2013.

[7] M. Geig, Unity: przewodnik projektanta gier, Gliwice: Helion, 2015.

[8] https://docs.unity3d.com/Manual/IL2CPP.html. [17.06.2017]

[9] https://docs.unity3d.com/Manual/webgl.html. [17.06.2017]

[10] https://blogs.unity3d.com/2016/09/20/understanding-memory-in-unity-webgl/. [17.06.2017]