JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 12 (2019) 220-225
Wysłane: 2019-06-10
Przyjęte: 2019-06-21

Analiza wpływu technologii oraz metod przesyłania hologramu na parametry i efekty transmisji

Krzysztof Mazur*, Damian Mazur
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Celem artykułu jest porównanie i analiza wpływu technologii i technik przesyłania danych, pod kątem wyświetlania obrazu przy pomocy ostrosłupa holograficznego. Oceniając użyteczność rozwiązania pod uwagę będą brane parametry: czas przesyłania klatek obrazu, użycie parametrów fizycznych maszyny i parametry pracy Wirtualnej Maszyny Javy.

Słowa kluczowe: Hologram; OpenCv; Java; JDK

*Autor do korespondencji.

Adres e-mail: krzysztof.mazur94@gmail.com

Analysis of the impact of technologies and methods of hologram transmission on the parameters and effects of transmission

Krzysztof Mazur*, Damian Mazur
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The aim of the article is to compare and analyze the impact of technologies and data transfer techniques in term of displaying the image using a holographic pyramid. When assessing the usability of the solution, the following parameters will be taken into account: time of image transfer, use of physical parameters of the machine and parameters of the Java Virtual Machine.

Keywords: Hologram; OpenCv; Java; JDK

*Corresponding author.

E-mail address: krzysztof.mazur94@gmail.com

1. Wstęp

Człowiek od zarania dziejów starał się prezentować otaczający go świat przy pomocy obrazów. Pierwsze rysunki datowane są na okres kultury magdaleńskiej (17000 – 15000 lat p.n.e.) i znajdują się one w jaskini Lascaux. Przedstawiają one w większości zwierzęta, namalowane przy pomocy palców, pędzli czy prymitywnych aerografów zrobionych z kości.

Od powstania malowideł w Lascaux, minęły już tysiąclecia, a technika znacząco się rozwinęła. W 1920r. polski fizyk, profesor Mieczysław Wolfke, opracowując teoretyczne podstawy i rozbijając proces tworzenia obrazu na oddzielne fazy, stworzył podwaliny dzisiejszej holografii.

Myśląc o zastosowaniu holografii w pierwszym momencie pojawia się wizja trójwymiarowych projekcji, czyli holowizji znana z telewizji. Należy jednak pamiętać, że takie hologramy stoją jeszcze poza zasięgiem ludzkości. Jednak hologramy stosowane są już w medycynie, przy wykrywaniu komórek rakowych, w informatyce jako pamięci holograficzne, czy zabezpieczenia przed fałszowaniem dokumentów.

Przed osiągnięciem hologramów znanych z telewizji czy gier komputerowych, musimy pokonać jeszcze kilka barier technologicznych. Możemy jednak już z powodzeniem symulować podobny efekt przy pomocy urządzenia nazywanego piramidą holograficzną.

Przesyłanie obrazu holograficznego wymaga przesłania sekwencji obrazów holograficznych. Aby osiągnąć płynność wyświetlania tego typu projekcji niezbędne jest osiągnięcie odpowiedniej liczny klatek na sekundę. Dlatego też autorzy podjęli się określenia czynników wpływających na szybkość przesyłania tego typu danych. Zbadano wpływ użycia wybranych technologii oraz technik transmisji, a uzyskane wyniki przeanalizowano.

2. Piramida holograficzna

Piramida holograficzna przedstawiona na rysunku 1, jest to ostrosłup, ze ściętym szczytem, wykonany z przezroczystych ścian. Wstawiany jest on na ekranie urządzenia. Zapewnienie odpowiedniej jakości obrazu jest kluczowym elementem, by hologram sprawiał wrażenie rzeczywistego. Poszarpane krawędzie wyświetlanego obiektu sprawią, że obraz pokazany na ściankach piramidy, będzie wyglądał sztucznie.

Tworzenie hologramu w przypadku przeprowadzanych badań, polegało na pobraniu obrazu z kamery oraz wycięciu z klatki tła, co odróżnia proces od standardowego przesyłania wideo z kamer.

Rys. 1. Przykład piramidy holograficznej

3. Opis badań oraz aplikacji

Aplikacja przygotowana na potrzeby wykonania badań, umożliwiała dowolne wymienianie technologii użytych w procesie tworzenia hologramu. Napisana została w języku Java. Umożliwia ona pobranie klatek z podłączonych kamer, przeprocesowaniu ich przez algorytm wycinający tło oraz wysłaniu klatki do aplikacji klienckiej. Ważnym elementem aplikacji, jest jej konfigurowalność, umożliwiająca wyłączenie poszczególnych elementów procesu, dzięki czemu na badany jest rzeczywisty wpływ danej technologii np. dla testów WebSocket oraz Http wyłączono procesowanie klatki przez algorytmy wycinające tło, ponieważ badany był jedynie czas przesyłania danych od momentu wysłania klatki z serwera do odebrania jej po stronie klienta.

Analiza podzielona została na 4 etapy. Pierwszym z nich jest badanie metody przesyłania danych, w której to porównane zostały technologie Http i WebSocket. W tym celu te same obrazki, przesyłane były z użyciem obu technologii, na tym etapie analizy, brany pod uwagę był jedynie czas w jakim przesłano określoną liczbę klatek.

Kolejnym etapem analizy, było zbadanie wydajności działania aplikacji skompilowanej i uruchomionej przy pomocy różnych JDK (Java Development Kit). Wybrane zostały cztery najpopularniejsze wydania JDK: OpenJDK, Oracle JDK, GraalVM JDK i Zulu JDK. Badanie polegało na skompilowaniu i uruchomieniu aplikacji na każdym z podanych JDK. Jako kryteria oceny brane pod uwagę były: czas uruchomienia aplikacji, czas zajętości procesora, użycie pamięci RAM.

Przedostatnim badanym elementem, był serwer aplikacyjny. Aplikacja z poprzednich badań uruchomiona została na Jetty i Tomcat. Dla każdego z nich analizowano: czasy przesyłania klatek przy pomocy WebSocket, czas zajętości procesora i użycie pamięci RAM.

Ostatnie badanie polegało na przeprocesowaniu tego samego obrazka, przy pomocy dwóch algorytmów, służących do wycinania tła. Analizowane były algorytmy, z biblioteki OpenCV, KNN i MOG. Oba algorytmy wycinały tło z tego samego obrazka, a badane były: czas procesu oraz jakość efektu algorytmu.

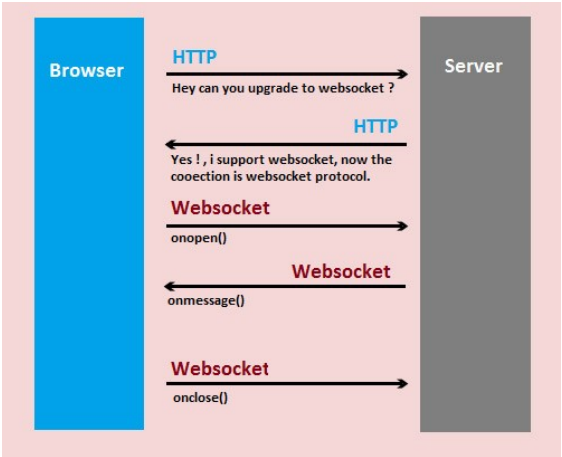
Przeprowadzenie wszystkich badań, pozwoliło na wybranie najlepszych zestawów technologii, służących do wyświetlania hologramu przy pomocy piramidy holograficznej, dla aplikacji napisanych w języku Java.

4. Http i WebSocket

Przesyłanie informacji jest głównym zagadnieniem do którego wykorzystywany jest Internet. Na przestrzeni lat, opracowano kilka protokołów, którymi można komunikować ze sobą oddzielne jednostki.

Jednym z nich jest protokół HTTP. Komunikacja odbywa się przy pomocy żądań klienta do serwera oraz odpowiedzi serwera do klienta na te żądania. Oba elementy mają określoną strukturę, którą muszą spełniać. HTTP zaliczany jest do protokołów bezstanowych, czyli serwer w momencie wysłania odpowiedzi do klienta, nie przechowuje informacji o obsłużonym żądaniu. W przypadku chęci zapamiętania stanu, należy wykorzystać mechanizm ciasteczek lub utrwalać sesję po stronie serwera. Architekturą komunikacji bazującą na protokole HTTP, jest REST

WebSocket jest protokołem komunikacyjnym zapewniającym dwustronną komunikację. Odbywa się ona na zasadzie zestawienia połączenia, między dwiema jednostkami i wymianie wiadomości. Różni się od HTTP ze względu na fakt, że wiadomości mogą być wysyłane, przez zestawione połączenie, bez schematu żądań/odpowiedzi.

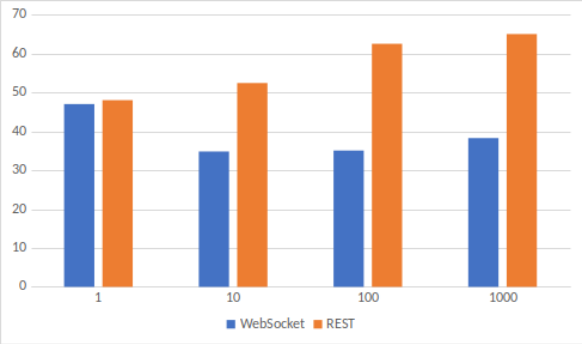


Rys. 2. Schemat zestawiania połączenia WebSocket, między przeglądarką a serwerem [https://www.pushtechnology.com/support/kb/understanding-the-java-virtual-machine-heap-for-high-performance-applications/]

Jak można zauważyć na rysunku 2 nawiązywanie połączenia WebSocket podzielone jest na kilka etapów. Pierwszym z nich jest wysłanie żądania HTTP rozpoczynając fazę "uścisku dłoni". W momencie, gdy serwer odpowie, że umożliwia komunikację przy pomocy WebSocket, połączenie przechodzi aktualizację i z HTTP zamieniane jest na połączenie WebSocket. Od tego momentu i klient i serwer mogą przesyłać wiadomości przy pomocy zestawionego połączenia [1].

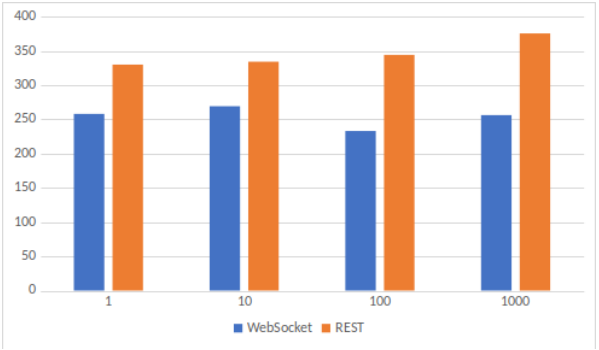
Analizę wpływu protokołu na czas przesyłania danych podzielono na trzy części. W każdej z nich użyto pliku o innej

wielkości, są to odpowiednio: 250KB, 500KB i 1MB. Każdy z obrazków przesyłano przy pomocy WebSocket i HTTP wykorzystując architekturę REST.



Rys. 3. Średnia czasu obsługi żądania dla obrazku 250KB

Jak widać na wykresie z rysunku 3, przesyłając jedną paczkę przy pomocy obu technik nie ma dużej różnicy. Można powiedzieć, że czasy są takie same. Różnica zaczyna być istotna dla dziesięciu paczek. Średnia czasów dla WebSocket jest mniejsza niż wynik dla jednej paczki. Jest to spowodowane tym, że przed zestawieniem połączenia WebSocket, musi zostać ono zaktualizowane z połączenia HTTP. Każda kolejna wiadomość nie jest już obarczona dodatkowym czasem. Średnia czasów REST jest większa niż dla jednego żądania. Największa różnica jest widoczna dla stu próbek. Średnia WebSocket jest dwukrotnie mniejsza niż dla REST i wynosi 35ms (milisekund), gdzie użycie protokołu HTTP wymaga średnio 60ms. Dla tysiąca próbek widać tą samą tendencję co dla poprzedniego badania.



Rys. 4. Średnia czasu obsługi żądania dla obrazku 250KB

Wyniki dla badania przeprowadzonego dla obrazu o wielkości 1MB, rysunek 4, potwierdzają wcześniej wyciągnięte wnioski.

5. Java

Java jest językiem programowania, którego główną zaletą jest możliwość uruchomienia na dowolnych systemie, jest to możliwe, dzięki użyciu Wirtualnej Maszyny Javy. Oczywiście wraz z korzyściami jakie niesie za sobą takie rozwiązanie, trzeba liczyć się z tym, że uruchamiany program wykona się wolniej, niż kod natywny pod konkretną platformę.

Wydajność aplikacji Javowych, zależy m.in. od użytego JDK (Java Development Kit - z ang. Zestaw Narzędzi Deweloperskich), użytej technologii przesyłania danych czy od rodzaje serwera, na którym aplikacja jest uruchomiona. Każde JDK, mimo że spełniają jeden standard, to różnią się działaniem i szczegółami implementacyjnymi. Podobnie jest w przypadku serwerów, choć każdy z nich jest w stanie uruchomić te same aplikacje, sposób działania może się różnić, a co za tym idzie, mogą mieć lepsze i gorsze zastosowania. Wymiana jednego z elementów, może skutkować poprawą działania programu lub spadkiem wydajności.

Aplikacja została skompilowana i uruchomiona na czterech różnych JDK:

- Oracle JDK,
- Zulu JDK,
- OpenJDK,
- GraalVM.

Każdy z programów działał przez 2 minuty i 35 sekund. Po dwudziestu sekundach działania programu, rozpoczyna się publikowanie klatek obrazu z odstępem stu milisekund.

5.1. Czas uruchomienia programu

Program przygotowany wykorzystuje framework Spring Boot w wersji 2.1.3. Jest to dodatkowy moduł do frameworka Spring, który upraszcza przygotowanie konfiguracji. Czas uruchomienia aplikacji różnił się znacząco między JDK:

- GraalVM - 2.6 sekund,
- OpenJDK - 3.6 sekund,
- Zulu JDK - 4.1 sekund,
- Oracle JDK - 2.2 sekund.

Każdy z JDK załadował inną liczbę klas do JVM.

Tabela 1. Liczba załadowanych klas

Nazwa JDK	Liczba klas
GraalVM	6741
OpenJDK	6468
Oracle JDK	6544
Zulu JDK	6558

Podobnie, jak w przypadku badania przeprowadzonego przez autora artykułu [2], GraalVM załadował więcej klas niż pozostałe JDK, lecz jest to o wiele mniejsza różnica. W artykule było to 25% w porównaniu do OpenJDK, w aktualnym badaniu różnica wynosiła tylko 5%. Jest prawdopodobne, że wraz z kolejnymi wersjami, GraalVM zoptymalizuje mechanizmy ClassLoadera. Należy jednak pamiętać, że sam fakt ładowania większej ilości klas do pamięci nie jest problematyczny, ale może mieć to wpływa na czas działania mechanizmów JVM np. Garbage Collectora.

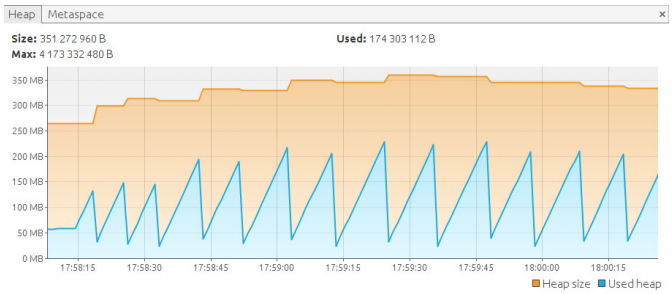
5.2. Użycie procesora

Wykresy pracy programu dla wszystkich JDK pokazały, że skoki użycia procesora pokrywają się z uruchomieniem Garbage Collectora. W standardzie Javy 8, domyślnym algorytmem Garbage Collectora jest Parallel GC [3].

Charakteryzuje się on częstymi uruchomieniami, wykorzystuje dwie strategie Mark-Copy dla nowej generacji oraz Mark-Sweep-Compact dla starej generacji. Faza stop-the-world blokuje wykonywanie programu przez cały czas trwania usuwania obiektów. Dodatkowo, mimo obaw, aplikacja uruchomiona na GraalVM nie wykorzystuje procesora w większym stopniu niż pozostałe JDK.

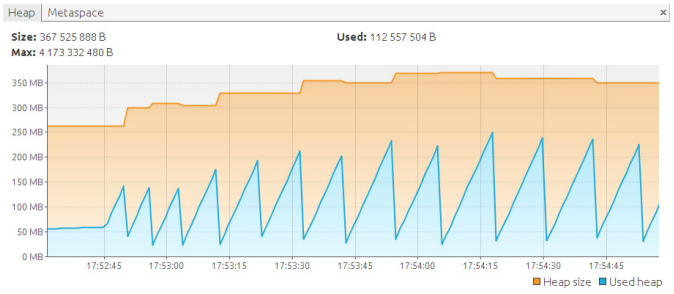
5.3. Użycie pamięci

Wszystkie aplikacje uruchamiane były bez dodatkowych parametrów określających wielkość sterty. Garbage Collector uruchamiał się automatycznie, bez wymuszania jego pracy.



Rys. 5. Wykres użycia pamięci dla Oracle JDK

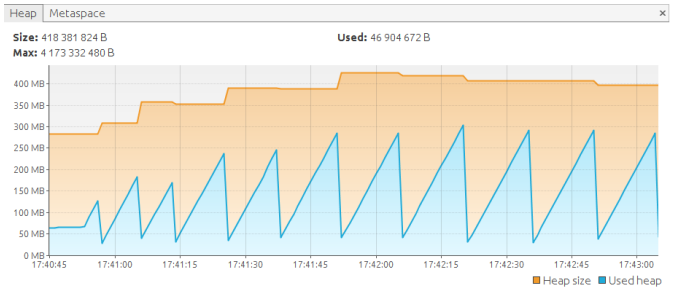
Analizując wykres wielkości sterty dla Oracle JDK z rysunku 5, można zauważyć, że aplikacja wykorzystywała w szczytowym momencie około 240MB pamięci. Każde uruchomienie Garbage Collectora powodowało usunięcie zbędnych obiektów i zwolnienie zasobów do poziomu około 50MB. Garbage Collector uruchamiał się w podobnych momentach. Przez pierwsze 30 sekund działania granicą startu algorytmu było około 150MB, następnie nastąpiła ewaluacja algorytmu i granica została przesunięta do poziomu około 200MB, by od minuty po uruchomieniu programu zatrzymać się na około 240MB.



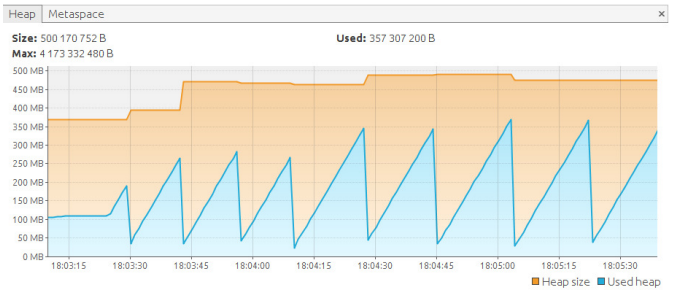
Rys. 6. Wykres użycia pamięci dla Open JDK

Podobne zachowanie, można zauważyć w przypadku wykresu dla Open JDK (rysunek 6). Profil uruchamiania się Garbage Collectora jest zbliżony zachowaniem do Oracle JDK. Jedyną różnicą jest maksymalne użycie pamięci, sięga ono równo 250MB, gdzie dla Oracle JDK wynosiło około 240MB. Jest to jednak nieznaczna różnica i może wynikać z obciążenia środowiska w trakcie działania aplikacji.

GraalVM wykorzystał nieznacznie więcej pamięci, największe użycie wynosiło 300MB (rysunek 7). Zdecydowanie więcej zasobów zajęło Zulu JDK, osiągając 350MB RAMu.



Rys. 7. Wykres użycia pamięci dla GraalVM JDK



Rys. 8. Wykres użycia pamięci dla Zulu JDK

Wszystkie badania przeprowadzane były z użyciem domyślnych ustawień dla JDK 8. Java oferuje dużą ilość parametrów pozwalającą skonfigurować środowisko w zależności od potrzeb. Jednym z takich ustawień jest *Xms* i *Xmx*. Oba parametry dotyczą ustawień sterty, pierwszy odpowiada za jej minimalną wielkość, a drugi za maksymalną. Pozwala to na ograniczenie pamięci jaką JVM będzie zajmować w systemie oraz bezpośrednio wpływa na uruchamianie algorytmu Garbage Collectora. Sam Garbage Collector posiada kilka wersji. Domyślnie dla JDK 8 używany jest Parallel, charakteryzuje się on tym, że nawet w przypadku znaczącego zmniejszenia wielkości sterty, zasoby nie są oddawane do systemu. Stosując dodatkowy parametr *XX:+UseG1GC* [4], mamy możliwość włączenia algorytmu G1, który znacząco zmienia sposób usuwania zbędnych obiektów z pamięci oraz w przeciwieństwie do Parallel zwalnia on zasoby systemu.

6. Serwer aplikacyjny

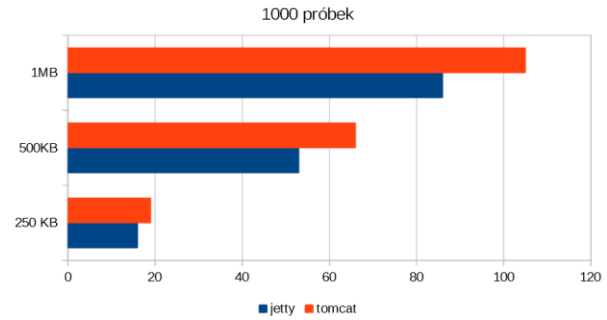
Aplikacje udostępniające dane do Internetu, napisane w języku Java, wymagają uruchomienia na serwerze lub kontenerze aplikacyjnym. Wynika to z faktu, że wykorzystując one technologię Servlet. Każdy z Servlet'ów posiada własny adres, którego to kontener będzie szukał w celu przekazania żądania.

Cykl życia servletu [5]:

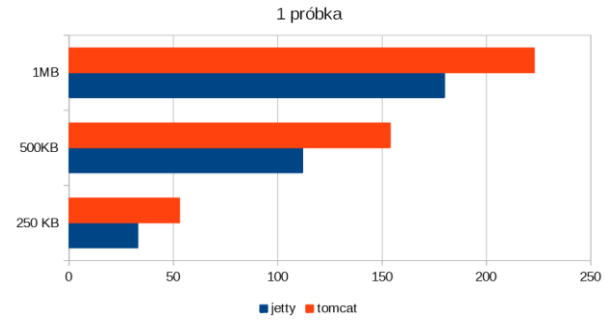
- kontener otrzymuje żądanie przez jedno ze swoich połączeń,
- kontener biorąc pod uwagę adres szuka servletu, który obsługuje żądanie,
- kontener sprawdza czy plik klasy jest załadowany, jeżeli nie, ładuje go, po czym tworzy instancję servletu,
- wywoływana jest metoda init(),
- wywoływana jest metoda service(),

- na zakończenie cyklu życia wywoływana jest metoda destroy().

Aplikacja została uruchomiona na Tomcacie oraz Jettym. Porównane zostały użycie pamięci oraz procent zajętości procesora w czasie działania aplikacji. Główną badaną statystyką jest czas w jakim przesłana zostanie klatka obrazu. Aplikacja wysyłała obrazy, przy pomocy technologii WebSocket, o rozmiarach odpowiednio 250KB, 500KB i 1MB. Próbki podzielone zostały na paczki po 1, 10, 100 i 1000 próbek.

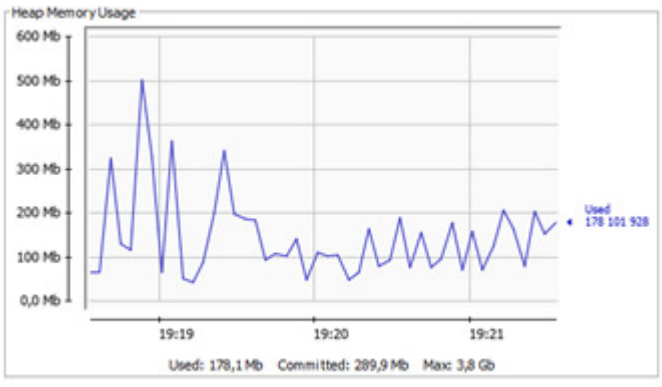


Rys. 9. Czasy dla przesyłu dla Tomcat i Jetty dla 1000 próbek

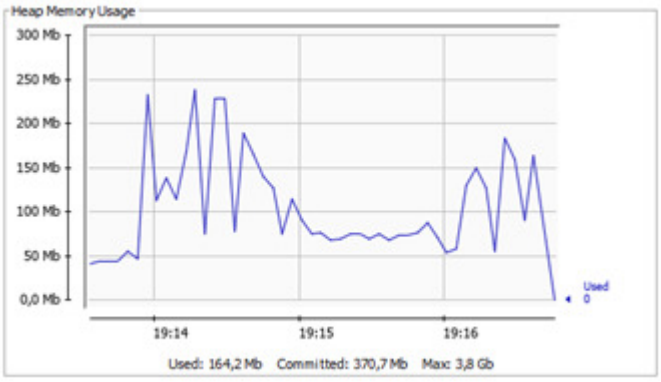


Rys. 10. Czasy dla przesyłu dla Tomcat i Jetty dla 1 próbki

Dane na wykresach z rysunków 9 i 10. faworyzują wybór Jetty nad Tomcatem. W obu przypadkach przesyłanie obrazu trwało krócej dla Jettiego. Podobnie było w przypadku paczek po 10 i 100 klatek. Analizując wykresy użycia pamięci (rysunki 11 i 12), zarezerwowanej na stertę, można dojść do wniosku, że Tomcat jest lepszym wyborem, w momencie gdy serwer posiada mniej pamięci RAM. Tomcat maksymalnie osiągnął zajętość około 240MB, co jest dwukrotnie mniejszą wielkością niż w przypadku Jettiego.



Rys. 11. Wykres użycia pamięci dla Jetty



Rys. 12. Wykres użycia pamięci dla Tomcat

Należy jednak zwrócić uwagę na fakt, że oba wykresy prezentują nierównomierne uruchamianie się Garbage Collectora.

7. OpenCV

Open Source Computer Vision jest to biblioteka programistyczna do przetwarzania obrazu w czasie rzeczywistym. Prace nad biblioteką rozpoczęły się oficjalnie w 1999 roku. Dotychczas zostało wydane 5 wersji aplikacji, natomiast pierwsza z nich została zaprezentowana w 2000 roku na konferencji poświęconej technikom rozpoznawania wzorców. Została ona napisana w jeżyku C, udostępniony jest jednak zbiór nakładek umożliwiający wykorzystanie biblioteki w językach takich jak:

- Java
- C
- C++
- Python.

Biblioteka OpenCv posiada wiele metod i technik przetwarzania obrazu w czasie rzeczywistym, dziedziny w których najczęściej wykorzystywana jest biblioteka to:

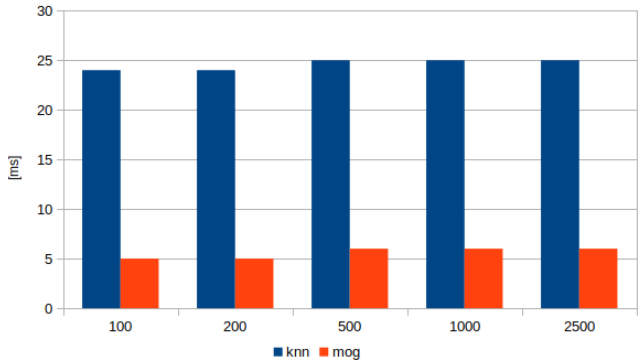
- rozpoznawanie twarzy,
- rozpoznawanie gestów,
- segmentacja obrazu,
- rozszerzona rzeczywistość,
- stereowizja,
- interakcja człowieka komputer.

Istotnym elementem pracy była implementacja mechanizmu do przetwarzania obrazu.

Przetwarzanie obrazu odbywało się z wykorzystaniem biblioteki OpenCv i polegało na wyekstrahowaniu pierwszego planu i wyświetlenie go na jednolitym tle którym zastąpiono plan drugi.

W bibliotece OpenCV do wyboru, są dwa algorytmy MOG i KNN [6]. Oba bazują na algorytmie GMM - Gaussian Mixture Model (z ang. Model Mieszaniny Gausa), w którym każdy piksel dostaje wagę w zależności od tego, jak długo piksel pozostaje w tym samym kolorze. Piksele, których kolor nie zmienia się długi czas, zostają sklasyfikowane jako tło [7].

Przeprowadzone badania, polegały na wyliczeniu czasu potrzebnego na ukończenie dwóch etapów, różnicowania i maskowania, dla pojedynczej klatki obrazu. Dodatkową zmienną był poziom progowania ustawiony odpowiednio na 100, 200, 500, 1000, 2500 jednostek.



Rys. 13. Czasy różnicowania dla algorytmów KNN i MOG

Na rysunku 13 widać znaczącą różnice w czasach. Algortym KNN potrzebował średnio pięciokrotnie więcej czasu, na różnicowanie tej samej klatki, dla tych samych wartości progowania.

Drugi etap algorytmu – maskowanie – trwał w obu przypadkach około 1ms.

Analizując wykresy, można stwierdzić, że algorytmy z grupy MOG, powinny być wybierane najczęściej, ponieważ są znacznie szybsze niż algorytm KNN. Dodatkowo dla jednej klatki, różnica w jakości uzyskanego, wyciętego obrazu jest znikoma. Należy jednak pamiętać, że w przypadku algorytmu MOG, każdorazowe poruszenie obiektu wycinanego z tła, powoduje zniekształcenie i konieczność ponownego ustalenia klatki tła. Dodatkową wadą jest wpływ oświetlenia, które musi być stałe, ponieważ w innym przypadku konieczne będzie ponowne dostrajanie. Na uzyskany efekt, duży wpływa mają parametry kamery. W przypadku słabego kontrastu i optyki, algorytm MOG może w zły sposób rozróżniać obiektu.

8. Wnioski

Celem przeprowadzonych badań było porównanie i analiza wpływu technologi i techniki przesyłania danych, pod kątem wyświetlania obrazu przy pomocy ostrosłupa holograficznego.

Początkowo badania przeprowadzane były przy pomocy dwóch komputerów. Na jednym pobierany był obraz, a drugi wyświetlał go przy pomocy ostrosłupa holograficznego. Dużą przeszkodą dla tej wersji badania okazywało się połączenie internetowe, które w znacznym stopniu wpływało na wahania czasów. Badania musiały zostać powtórzone i wykonane tylko na jednej jednostce roboczej, tak by odizolować środowisko badawcze od wpływów czynników zewnętrznych.

Oczywiste jest też to, że na polepszenie jakości, z zachowaniem dużej prędkości działania rozwiązania, mają wpływ parametry komputera. Badania przeprowadzone były przy użyciu dwóch komputerów: stacjonarnym i laptopie. Jednostka stacjonarna posiadała lepszy procesor, co skutkowało krótszymi czasami przetwarzania.

Podsumowując badania, można łatwo dojść do wniosku, że najlepszym zestawem technologii i metod przesyłania jest:

- WebSocket - jako metoda przesyłu, jest zdecydowanie szybsza i używa mniej transferu niż porównywany z nią REST,
- Zulu JDK - ponieważ cechował się szybkością działania, odpowiednim zarządzaniem pamięcią oraz jest posiada on darmową licencję,
- Jetty - mimo używania większej ilości zasobów, jest to szybki kontener i w przypadku, gdy posiadamy lepszą infrastrukturę, powinien być pierwszym wyborem. Oczywiście może zostać z powodzeniem zastąpiony Tomcatem, w momencie, gdy zależy nam na wydajnym używaniem maszyn,
- KNN dla poruszających się obiektów / MOG dla statycznych obiektów - wybór algorytmu musi być podyktowany zastosowanym źródłem hologramu. Nie ma generycznego algorytmu pozwalającego na uzyskiwanie idealnego odróżnienia tła od obiektu.

Literatura

[1] M. West, An Introduction to WebSocket, https://blog.teamtreehouse.com/an-introduction-to-websockets [05.06.2019].

[2] K. Kumar, What are the various components of JDK (Java Development Kit) environment?, http://cs-fundamentals.com/tech-interview/java/components-of-jdk-environment.php [12.05.2019].

[3] Standard Edition HotSpot Virtual Machine Garbage Collection Tuning Guide, https://docs.oracle.com/javase/8/docs/technotes/guides/vm/gctuning/ergonomics.html#ergonomics [05.06.2019].

[4] Tuning JVM Garbage Collection for Production Deployments, https://docs.oracle.com/cd/E40972_01/doc.70/e40973/cnf_jvmgc.htm [05.06.2019].

[5] Servlet Lifecycle, https://docs.oracle.com/javaee/6/tutorial/doc/bnafi.html [05.06.2019].

[6] A. Kaehler, G. Bradski. Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library, https://books.google.pl/books?id=SKy3DQAAQBAJ&pg=PT26&redir_esc=y#v=onepage&q&f=fals [11.04.2019].

[7] Reynolds, Douglas A., Quatieri, Thomas F., Dunn, Robert B, Speaker verification using adapted Gaussian mixture models. Digital Signal Process, https://www.sciencedirect.com/science/article/pii/S1051200499903615?via%3Dihub [11.04.2019].