Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

Kierunek: **Informatyka techniczna**
Specjalność: **Grafika i systemy multimedialne**

PRACA DYPLOMOWA MAGISTERSKA

Autorska implementacja algorytmów transformacji Hough'a dla wybranych metod akceleracji obliczeń w środowiskach języka JavaScript

Propietary implementation of Hough transformation algorithms for selected calculation acceleration methods in JavaScript language environments

Damian Koper

Opiekun pracy
Dr inż. Marek Woda

WROCŁAW, 2022

Streszczenie

blabla

Słowa kluczowe: raz, dwa, trzy, cztery

Abstract

blabla in english

Keywords: one, two, three, four

Spis treści

Spis rysunków

Spis tabel

Spis listingów

Skróty

EXAMPLE (ang. *Example*)

Rozdział 1

Wstęp

Prawo Moore'a mówi, że liczba tranzystorów w układach scalonych podwaja się co około dwa lata. Prawo Koomey'a opisuje natomiast trend wzrostu liczby obliczeń na jeden dżul energii, która podwaja się co 1.57 lat. Choć w ostatnich latach, w związku ze zmniejszającym się tempem miniaturyzacji tranzystorów, wartości te przestały być aktualne to wciąż mamy do czynienia ze zjawiskiem ustawicznego wzrostu mocy obliczeniowej. Dodatkowo, zgodnie z obserwacją nazwaną prawem Huang'a - prezesa firmy NVIDIA, wzrost wydajności układów graficznych wzrasta więcej niż dwukrotnie co dwa lata[4], co świadczy o obecnym rozwoju możliwości optymalizacji architektur i programów wykorzystujących przetwarzanie masowo równoległe. Wzrost wydajności z kolei zwiększa możliwości wykorzystania algorytmów, które wcześniej były zbyt intensywne obliczeniowo i nie mogły być wykorzystane w rozwiązaniach produkcyjnych. Powszechnie dostępne wydajne maszyny dają również możliwości szybszego prototypowania rozwiązań większej liczbie osób, co z kolei wpływa na rozwój samych algorytmów i ich zastosowań. Algorytmy i obliczenia opierają się na osiągnięciach analizy numerycznej oraz matematyki dyskretnej.

Analiza numeryczna zajmuje się opisywaniem i analizą metod pozyskiwania wyników dla problemów matematycznych. Dzięki jej osiągnięciom możliwe jest budowanie algorytmów, które są kompletnym i jednoznacznym opisem metody konstruowania rozwiązania owych problemów. Konstruowane algorytmy mogą mieć różne poziomy skomplikowania zaczynając od obliczania wartości funkcji, wielomianów, czy też znajdować rozwiązania układów równań. Dzięki nim możemy aproksymować wartości funkcji - obliczać pochodne oraz całki korzystając z metod prostokątów, trapezów, czy Simpson'a. Możemy obliczać przybliżenia funkcji trygonometrycznych korzystając z szeregów Taylor'a. Dzięki obliczeniom opartym o algebrę liniową i rachunek różniczkowy mogły rozwinąć się pola związana z uczeniem maszynowym. Znajdowanie wektorów i wartości własnych macierzy w metodzie PCA w celu redukcji wymiarowości danych, a w końcu algorytmy propagacji wstecznej szczególnie wykorzystywane w intensywnie rozwijającym się obszarze uczenia głębokiego[5].

Niezależnie od skali zaawansowania algorytmów dążą one do stanowienia rozwiązania konkretnych problemów świata rzeczywistego. Przykładem ich zastosowania jest przewidywanie pogody w oparciu o modele meteorologiczne, które zaczęło intensywnie rozwijać się wraz z dostępem do coraz większej mocy obliczeniowej i udoskonalaniem samych modeli. W ciągu 15 lat, od 1971 roku, spełnialność prognoz 36-godzinnych zrównała się ze spełnialnością prognoz 72-godzinnych[3]. Kolejnym wartym przytoczenia przykładem jest obszar analizy obrazów z wyszczególnieniem zagadnienia ich klasyfikacji, gdzie wzrost mocy obliczeniowej pozwolił na rozwój algorytmów. W ciągu 8 lat metryka dokładności klasyfikacji obrazów na zbiorze danych ImageNet wzrosła z 63% do 90%[3, 1].

Dynamiczny rozwój algorytmów napędzany jest przede wszystkim przez poszerzanie i budowanie wiedzy domenowej, specyficznej dla rozwiązywanego problemu. Jednak, aby uczynić taki rozwój możliwym, musi być on oparty na niezbędnych filarach jakimi są oprogramowanie, które służy do prototypowania, a następnie wdrażania rozwiązań oraz środowisko sprzętowe, które umożliwia przeprowadzanie obliczeń, wielokrotnych testów, prototypów na małą oraz wdrożeń na dużą skalę. Wraz ze wzrostem złożoności problemu liczba obliczeń niezbędnych do jego rozwiązania również rośnie i w przypadku ich większości ten wzrost jest wykładniczy lub większy. Niezbędnym zatem jest, aby oprogramowanie potrafiło wykorzystać wszelkie dostępne metody akceleracji zarówno te związane z optymalizacją samego algorytmu, jak i te związane z mechanizmami akceleracji sprzętowej.

1.1. Istota rzeczy

Wykonywanie obliczeń numerycznych wśród obecnie popularnych rozwiązań można podzielić na dwie grupy. Pierwsza z nich używa specjalnie do tego celu stworzonego języka programowania, często również w połączeniu ze zintegrowanym środowiskiem programistycznym (Integrated Development Environment, ang. IDE). Przykładem takiego rozwiązania jest środowisko i język MATLAB[8] oraz R[11], czy też środowisko Mathematica z językiem Wolfram[7].

Druga grupa używa języka ogólnego przeznaczenia do wykonywania obliczeń w oparciu o zewnętrzne biblioteki w zdecydowanej większości udostępniane jako oprogramowanie open-source, które dostarczają wymagany zestaw funkcjonalności niwelując potrzebę ich ręcznej implementacji. Języki takie możemy podzielić na te niskiego poziomu, zapewniające wysoką wydajność, oraz te wysokiego poziomu, interpretowane, zapewniające większą wygodę użytkowania. Najbardziej popularnym tego typu środowiskiem jest język Python, którego społeczność stworzyła liczne biblioteki (w postaci pakietów) do przetwarzania danych, obliczeń numerycznych i statystycznych, analizy obrazów, czy uczenia maszynowego. Najpopularniejsze z nich podane zostały w tabeli 1.1. W nawiasach została ujęta liczba pobrań danego pakietu w przeciągu ostatniego tygodnia na dzień 2022-04-27. Dla punktu odniesienia warto dodać, że w przeciągu tego samego tygodnia, w całym ekosystemie, pobrano łącznie 3.409.997.407 pakietów [10].

Biblioteki takich języków, czego przykładem jest biblioteka OpenCV, często implementowane są w językach kompilowanych bezpośrednio do kodu maszynowego takich jak C++ czy Rust udostępniając jednolity interfejs, którego metody, poprzez powiązania, wywoływać mogą języki skryptowe wysokiego poziomu, takie jak już wspomniany Python. Takie rozwiązania zapewnia możliwość zbudowania wersji biblioteki kompatybilnej z wieloma środowiskami i językami na podstawie jednego kodu bazowego, poddając adaptacji tylko elementy architektury integrującej bibliotekę niskopoziomową i język wysokiego poziomu. Skompilowana biblioteka poddana może być również procesom optymalizacji, co zwiększyć może jej wydajność. Wadę takiego rozwiązania stanowi konieczność kompilowania biblioteki niskiego poziomu dla wielu systemów operacyjnych, czy architektur procesora. Taka kompilacja odbywać może się przed publikacją samej biblioteki, a gotowe artefakty pobierane są przez użytkownika w momencie instalacji. Kompilacja może odbywać się również bezpośrednio na maszynie użytkownika w momencie instalacji.

Opisany podział nie skłania do uznania przewagi jednej z grup nad drugą w żadnym z aspektów. W środowiskach specyficznych i zintegrowanych brakujące funkcjonalności mogą zostać dodane przez twórców jako biblioteki standardowe języka oraz zaimplementowane przez społeczność w bibliotekach języków ogólnego przeznaczenia. Obie grupy rozwiązań z reguły są w stanie działać w środowisku tego samego systemu operacyjnego i wchodzić w interakcje z tym samym sprzętem, i wykorzystać idące za tym możliwości akceleracji obliczeń. Jednak nie wszystkie języki ogólnego przeznaczenia i technologie zyskały jednakową popularność w ob-

Tab. 1.1: Popularne biblioteki do przetwarzania danych w języku Python i ich odpowiedniki w języku JavaScript. Dane pochodzą z serwisów kolejno PyPI Stats oraz NPM, a w nawiasach znajduje się tygodniowa liczba pobrań biblioteki.

Python	**JavaScript**	**Zastosowanie**
numpy (26.067.844)	numjs (533)	Operacje na macierzach
pandas (19.778.648)	danfojs (1.029)	Operacje na strukturach danych
scipy (9.986.376)	simple-statistics (87.882) fft.js (8.027)	Operacje związane z analizą numeryczną, przetwarzanie sygnałów, algebra liniowa.
scikit-learn (7.705.438)	ml (156)	Uczenie maszynowe
matplotlib (6.457.099) plotly (1.632.246)	plotly.js (149.542) c3 (83.564)	Wizualizacja danych
tensorflow (3.348.986)	@tensorflow/tfjs (91.233)	Sieci neuronowe
opencv-python (1.236.711)	OpenCV.js (b.d [9]) jimp (1.479.783) image-js (4.103)	Operacja na obrazach, computer vision

szarze obliczeń numerycznych mimo swojej ogólnej popularności. Przykładem takowych są technologie webowe z językiem JavaScript na czele.

1.1.1. Duża i rosnąca rola technologii webowych

Technologie webowe zdefiniować można jako narzędzia i techniki umożliwiające wymianę danych pomiędzy różnymi urządzeniami przez internet. Technologie webowe mogą występować i spełniać różne zadania na wielu poziomach architektury aplikacji. W przypadku architektury klient-serwer środowiskiem frontend'owym umożliwiającym wyświetlanie i obsługę graficznego interfejsu użytkownika zwykle jest przeglądarka internetowa, gdzie za jego implementację na najniższym poziomie abstrakcji odpowiadają języki HTML, CSS i JavaScript. Serwerem może być na przykład aplikacja komunikująca się z klientem za pomocą API zaimplementowanego w architekturze REST, czy też za pomocą języka zapytań GraphQL uruchamiana w środowisku NodeJS.

Warto zaznaczyć, że serwer, jak i klient, nie muszą być zaimplementowane z wykorzystaniem języka JavaScript by być uznawanym jako aplikacja webowa. Języki takie jak PHP, Python, Ruby, Java, czy C# także oferują zaawansowane frameworki, jednak to język JavaScript, ze względu na historię swojego rozwoju ściśle powiązaną z przeglądarką internetową, uważany jest jako główne narzędzie i czynnik rozwoju technologii webowych zarówno w części klienckiej jak i serwerowej. Potwierdzają to badania, gdzie JavaScript w 2021 roku dziewiąty rok z rzędu został wyłoniony jako najpopularniejsza technologia wśród developerów [13].

JavaScript w obliczeniach numerycznych

Pomimo swojej popularności, w przeciwieństwie do języka Python, język JavaScript nie zyskał popularności wśród zadań obliczeń inżynierskich, naukowych, statystycznych, obróbki i analizy obrazów. Jest on starszy od języka JavaScript i w przeciwieństwie do niego od początku mógł pełnić zadanie narzędzia do implementacji algorytmów obliczeniowych, podczas gdy JavaScript ograniczony był jedynie do jednowątkowego środowiska przeglądarki internetowej. Dopiero w 2009 roku, wraz z pojawieniem się środowiska NodeJS, możliwe stało się uruchamianie kodu JavaScript po stronie serwera. Umożliwia to również powstały w 2018 roku środowisko Deno.

Innym czynnikiem, który warunkuje tempo rozwoju i potencjalny przepływ użytkowników ku lepszym ich zdaniem środowiskom jest dostępność metod akceleracji obliczeń, która warunkuje możliwości poprawy wydajności. W konsekwencji przekłada się to na wygodę użytkowania i możliwości prototypowania bardziej złożonych algorytmów dla problemów o większych rozmiarach. Przykładem takich są algorytmy uczenia maszynowego, w szczególności uczenia głębokiego.

Ostatnim z analizowanych czynników jest architektura systemu pakietów, która jest ściśle powiązana ze środowiskami uruchomieniowymi, które z nich korzystają. Pobierane pakiety zawierają biblioteki, które importowane są do projektu w postaci modułów. Python posiada jeden format modułów w przeciwieństwie do języka JavaScript ze względu na heterogeniczność środowisk jego wykonywania. Przeglądarka internetowa, NodeJS i Dino mimo, że wykonują kod w tym samym języku, posiadają znaczące różnice w sposobie komunikacji z użytkownikiem, systemem operacyjnym, w dostępności metod akceleracji i zarządzaniu modułami. Rodzi to problemy

Obecnie jednak, z technicznego punktu widzenia, nie istnieją zasadnicze różnice pomiędzy środowiskami języków Python i JavaScript, które hamowałyby w znacznym stopniu rozwój bibliotek służącym do prototypowania i wdrażania aplikacji zajmującymi się obliczeniami numerycznymi w środowiskach języka JavaScript.

W tabeli 1.1 przedstawiono porównanie popularnych bibliotek stosowanych przy obliczeniach numerycznych języka Python z ich odpowiednikami w języku JavaScript. Widać wyraźnie dysproporcję w liczbie pobrań bibliotek pomiędzy językami. W wielu przypadkach autorowi nie udało się znaleźć dokładnego odpowiednika danej biblioteki, a znalezione zestawy bibliotek języka JavaScript nie pokrywają w całości funkcjonalności biblioteki języka Python.

1.2. Cel badań i zawartość pracy

Celem niniejszej pracy jest zbadanie przystosowania środowisk języka JavaScript do przeprowadzania obliczeń numerycznych oraz wykonywania złożonych algorytmów. Analizie poddano popularne środowiska - NodeJS, Deno i przeglądarki internetowe Google Chrome i Mozilla Firefox. Dla tych środowisk zbadano dostępne metody akceleracji obliczeń, gdzie pojęcie akceleracji rozumiane jest jako każda modyfikacja algorytmu wpływająca pozytywnie na jego wydajność. Tyczy się to optymalizacji wersji sekwencyjnej algorytmu bez zmiany jego złożoności, jak i jego modyfikację do wersji zrównoleglonej. Metody akceleracji zaimplementowane dla badanych środowisk przedstawione zostały w tabeli 1.2. Zostaną one opisane dokładnie w dalszej części pracy. Analiza zorientowana jest na algorytmy analizy obrazów.

Tab. 1.2: Zaimplementowane metody akceleracji dla badanych środowisk.

	Chrome	Firefox	Node	Deno
Sequential	✓	✓	✓	✓
Native addon	✗ [1]	✗ [1]	✓	✗ [2]
asm.js	✓	✓	✓	✗ [2]
WASM	✓	✓	✓	✗ [2]
WASM+SIMD	✓	✓	✓	✗ [2]
Workers	✓	✓	✓	✓
WebGL	✓	✓	✗ [2]	✗ [1]
WebGPU	✗ [3]	✗ [3]	✗ [1]	✗ [3]

[1] Niedostępne w środowisku
[2] Wymaga zewnętrznych paczek lub kodu bazowego w języku innym niż C++
[3] Niestabilne lub eksperymentalne

Na to złożone zagadnienie trzeba spojrzeć z wielu perspektyw. Autor pracy abstrahuje jednak od zagadnienia poprawy złożoności obliczeniowej samego algorytmu w wersji sekwencyjnej, ponieważ rozważania takie wykraczają poza zakładaną w pracy tematykę, która koncentruje się na środowiskach języku JavaScript i możliwościach tych środowisk, jak i samego języka.

1.2.1. Perspektywa wydajności

Pierwszą z perspektyw jest wydajność algorytmu postrzegana przez pryzmat środowiska, w którym jest on wykonywany oraz wykorzystywanych przez niego metod akceleracji. W pracy zbadano wydajność zaimplementowanego algorytmu z wykorzystaniem metod wymienionych w tabeli 1.2, które szczegółowo omówione zostały w rozdziale 2.3.

1.2.2. Perspektywa kompatybilności i budowania bibliotek

Innym punktem widzenia jest kompatybilność budowanych bibliotek w wielu środowiskach. Jednak każde z nich jest na tyle zróżnicowane, że zbudowanie jednej wersji kodu źródłowego kompatybilnego z nimi wszystkimi naraz jest wymagające, a niekiedy niemożliwe. Analiza tego zagadnienia, możliwości i ograniczeń modularności kodu języka JavaScript, dodatkowo w aspekcie wykorzystywanych metod akceleracji, opisana została w rozdziale 2.2.

1.2.3. Perspektywa wygody użytkowania

Celem autorów bibliotek jest dostarczenie rozwiązań, które przede wszystkim będą aktywnie wykorzystywane. Autor, razem z opisem implementacji algorytmów i ich późniejszego wykorzystania, w rozdziale 5 analizuje możliwości i sposoby interakcji ze środowiskami, wyświetlanie wyników, ładowanie danych oraz wewnętrzne mechanizmy przetwarzania danych związane również z modelem wykonania języka JavaScript.

1.2.4. Transformacja Hough'a jako wspólny mianownik analizy

W celu porównania wydajności pomiędzy środowiskami dla tej samej metody akceleracji oraz porównania ich w ramach jednego środowiska zaimplementowano algorytmy transformacji Hough'a (czyt. Haf'a), która dokładniej opisana został w rozdziale 3. Jest to rodzina algorytmów deterministycznych wykorzystywana do detekcji kształtów parametrycznych i nieparametrycznych na obrazach. Własna implementacja algorytmów z tej rodziny, w przeciwieństwie do zestawów testowych takich jak na przykład Ostrich[2], daje możliwości granularnej kontroli nad samą implementacją i możliwości dostosowania jej do wszystkich analizowanych środowisk i metod akceleracji. Samo wykrywanie kształtów na podstawie transformacji Hough'a jest intensywne obliczeniowo, wieloetapowe, wykorzystuje operacje zmiennoprzecinkowe, w tym funkcje trygonometryczne, oraz zakłada wykonanie wielu iteracji po dużych, zależnych od rozmiaru problemu i próbkowania, strukturach danych. Czynniki te czynią tę rodzinę algorytmów, zdaniem autora, dobrym punktem zaczepienia podczas szerokiej analizy środowisk i metod akceleracji, które one udostępniają, co pozwala uzyskać ogólną orientację w analizowanych perspektywach oraz wskazać wartościowe kierunki przyszłych badań.

1.2.5. Zawartość pracy

Rozdział drugi opisuje język JavaScript, wprowadza w zagadnienie jego modelu wykonania, środowisk oraz opisuje dostępne dla nich metody akceleracji i sposoby budowania bibliotek. Wspólny mianownik analizy, czyli algorytmy transformacji Hough'a i sposób ich działania opisany został w rozdziale trzecim. Rozdział czwarty opisuje metodykę przeprowadzanych testów wydajności, dla potrzeb których zaimplementowana została osobna biblioteka odpowiedzialna za pomiary czasu wykonania, mając na uwadze jej kompatybilność ze wszystkimi środowiskami. Implementację algorytmów i konsekwencje idące za stosowaniem poszczególnych rozwiązań opisuje rozdział piąty. Rozdział szósty przedstawia rezultaty testów wydajności ze wskazaniem na porównanie środowisk i metod do bazowych wariantów sekwencyjnego wykonania

algorytmu, a rozdział siódmy stanowi podsumowanie całości rozważań i eksperymentów oraz wskazuje zidentyfikowane problemy i proponowane kierunki przyszłych badań.

Rozdział 2

Język JavaScript

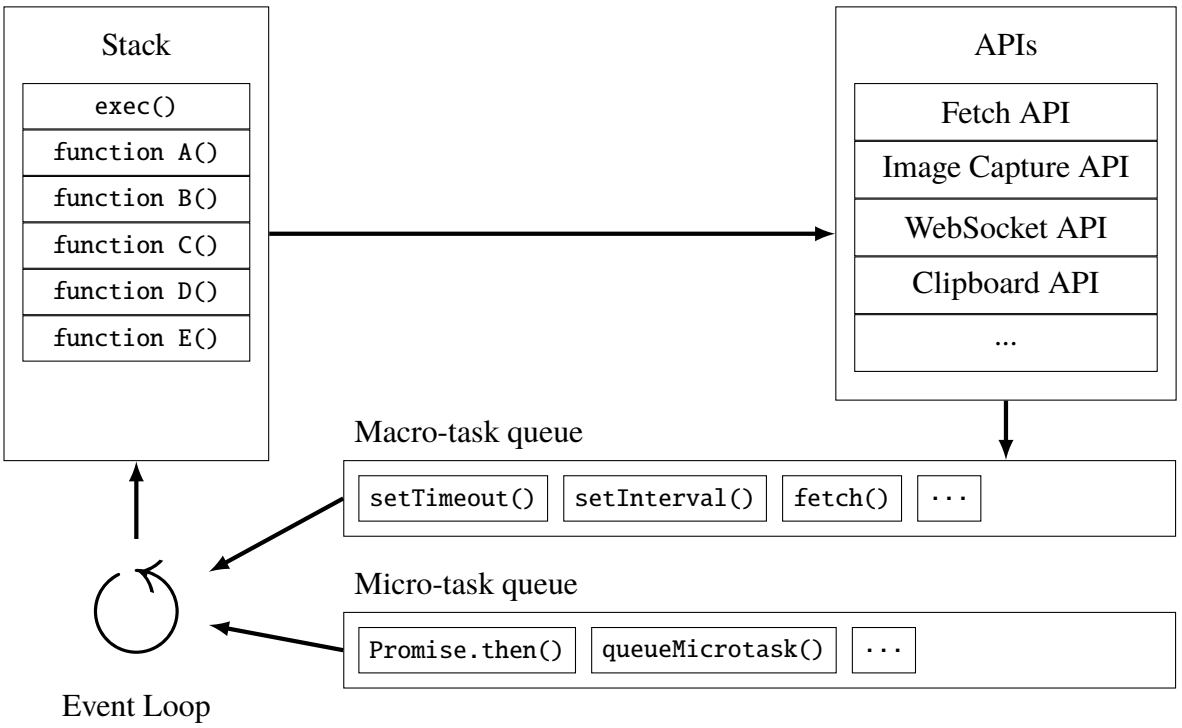
JavaScript od momentu swojego powstania w 1995 roku stanowił jeden z filarów rozwoju technologii webowych, zaczynając od dodania prostych mechanizmów interaktywności do statycznych stron internetowym, a kończąc na byciu nieraz jedynymi samodzielnym budulcem pełnowymiarowych aplikacji działających po stronie klienta i serwera, aplikacji działających w środowisku przeglądarki internetowej, ale też w środowiskach natywnych, desktopowych i mobilnych. Dlatego, aby zrozumieć w pełni specyfikę problemu, który stanowi przystosowanie języka do wykonywania obliczeń numerycznych, w tym rozdziale przybliżone zostały zagadnienia związane z modelem wykonania, środowiskami oraz sposobami na podział kodu na moduły oraz późniejsze ich wykorzystanie. Na końcu opisane zostały metody akceleracji, dla których przeprowadzono badania.

2.1. Model wykonania

Model wykonania języka JavaScript skoncentrowany jest w głównej mierze na obsłudze zdarzeń. W przeglądarce internetowej zdarzeniami takimi mogą być interakcje z użytkownikiem, na przykład kiedy naciśnięty zostanie przycisk, albo interakcje z siecią, kiedy otrzymamy odpowiedź na zapytanie z wykorzystaniem obiektu `XMLHttpRequest` lub skorzystamy z Fetch API. Po stronie serwera zdarzeniami takimi mogą być odebranie zapytania, które serwer musi obsłużyć, obsługa strumieni, ale także wszelkie odpowiedzi na interakcje z systemem operacyjnym. Podstawowymi interakcjami może być obsługa sygnałów, dostęp do plików, czy też obsługa sieci, która umożliwia połączenie na przykład z bazą danych.

Zdarzenia te obsługuje pętla zdarzeń (ang. event loop). Na rysunku 2.1 pokazano jej uproszczony model. Wyróżnia ona zadania, zwane także makro zadaniami, oraz mikro zadania. Dla każdego typu zadań utworzona zostaje osobna kolejka. Jeśli aktualnie wykonywane przetwarzanie sekwencyjne, którego ramki wywołań śledzone są na stosie, zakończy się, wtedy z pętli zdarzeń pobierane i wykonywane jest makro lub mikro zadanie. W pierwszej kolejności wykonywane są wszystkie mikro zadania, a gdy ich kolejka jest pusta, wykonywane jest kolejne makro zadanie.

Makro zadania dodawane są do kolejki, aby obsłużyć wspomniane już zdarzenia związane z działaniami użytkownika lub inne zewnętrzne zdarzenia. Są one również dodawane do kolejki, kiedy mija czas zadany podczas wywołań funkcji `setTimeout()` oraz `setInterval()`. Warto zaznaczyć, że wywołania tych funkcji nie gwarantują wykonania dokładnie po zadanym czasie, ale traktują go jako próg czasowy, po jakim zadana funkcja zostanie dodana do kolejki makro zadań [12]. Makro zadania dodane podczas jednej iteracji pętli nigdy nie zostaną wykonane w tej samej iteracji.



Rys. 2.1: Uproszczony model pętli zdarzeń środowisk języka JavaScript w wariancie z wyróżnieniem API przeglądarek internetowych.

Mikro zadania pochodzą tylko i wyłącznie z kodu użytkownika, bądź bibliotek i wykorzystywane są do obsługi asynchronicznych zadań, zarządzania ich kolejnością i obsługą błędów[6]. Do ich tworzenia wykorzystuje się głównie obiekt `Promise`, który reprezentuje zadanie, które w przyszłości zakończy się pomyślnie lub błędem. Dla takiego obiektu zdefiniować możemy funkcje, które wykonają się podczas scenariusza pomyślnego (`then`), błędnego (`catch`) oraz zawsze (`finally`) [14]. Mikro zadania pochodzić mogą również od obserwatorów na przykład `MutationObserver`, czy `ResizeObserver`.

Zrozumienie sposobu wykonywania kodu w asynchronicznym modelu języka JavaScript jest kluczowe do efektywnego wykorzystania możliwości, jakie idą za metodami akceleracji, których użycie możliwe jest tylko i wyłącznie poprzez asynchroniczne wywołania. Opisane w sekcji 2.3 metody bazujące na Worker'ach oraz WebGL wymagają interakcji poprzez wywołania asynchroniczne. Na listingu 2.1 pokazano przykład kodu asynchronicznego. Wywołania `console.log` wykonają się, zawsze drukując liczby w kolejności od 1 do 6.

Listing 2.1: Przykład kodu demonstrujący mechanizmy asynchroniczności w języku JavaScript.

```
1 console.log(1);
2 Promise.resolve().then(() => setTimeout(() => console.log(6)));
3 setTimeout(() => console.log(4), 0);
4 setTimeout(() => Promise.resolve().then(() => console.log(5)));
5 Promise.resolve().then(() => console.log(3));
6 console.log(2);
```

Linijki 1 oraz 6 zostają wykonene synchronicznie. `Promise` w linijce 5 zostaje wykonany jako następny, ponieważ jako mikro zadanie, wykona się zaraz po operacjach synchronicznych. Następnie funkcje `setTimeout` wykonają się w kolejności ich wywołania, gdzie w linijkach 2-4 na ich kolejność wpływ maja mechanizm `Promise`. Timeout z linijki 3, a potem 4 zostają wywołane jaki pierwsze. Jako ostatni wykonuje się timeout z linijki 2, ponieważ jego wywołanie, w postaci mikro taska, przeniesione zostało na koniec wykonania synchronicznego.

2.2. Środowiska i modularność kodu

https://libevent.org/ https://libuv.org/

2.3. Metody akceleracji

Rozdział 3

Transformacja Hough'a

3.1. Standard Hough Transform

3.2. Circle Hough Transform

Rozdział 4

Metodologia pomiarów

Rozdział 5

Zaimplementowane algorytmy

Rozdział 6

Wyniki pomiarów

Rozdział 7

Podsumowanie

asdsad

asdsad

Literatura

[1] Z. Dai, H. Liu, Q. V. Le, and M. Tan. Coatnet: Marrying convolution and attention for all data sizes. *Advances in Neural Information Processing Systems*, 34:3965–3977, 2021.

[2] Khan, Faiz and Foley-Bourgon, Vincent and Kathrotia, Sujay and Lavoie, Erick. Ostrich benchmark suite. `https://github.com/Sable/Ostrich` Na dzień 2022-04-27.

[3] P. Lynch. The origins of computer weather prediction and climate modeling. *Journal of computational physics*, 227(7):3431–3444, 2008.

[4] C. Mims. Huang's law is the new Moore's law, and explains why Nvidia wants arm. *The Wall Street Journal*, Sep 2020.

[5] G. M. Phillips and P. J. Taylor. *Theory and applications of numerical analysis*. Elsevier, 1996.

[6] In depth: Microtasks and the javascript runtime environment. `https://developer.mozilla.org/en-US/docs/Web/API/HTML_DOM_API/Microtask_guide/In_depth` Na dzień 2022-05-02.

[7] Mathematica. `https://www.mathematica.pl/` Na dzień 2022-04-24.

[8] Matlab. `https://www.mathworks.com/products/matlab.html` Na dzień 2022-04-24.

[9] Opencv.js. `https://docs.opencv.org/4.x/d4/da1/tutorial_js_setup.html` Na dzień 2022-04-27.

[10] Pypi stats. `https://pypistats.org/packages/__all__` Na dzień 2022-04-27.

[11] R. `https://www.r-project.org/about.html` Na dzień 2022-04-24.

[12] settimeout: Reasons for delays longer than specified. `https://developer.mozilla.org/en-US/docs/web/api/settimeout#reasons_for_delays_longer_than_specified` Na dzień 2022-05-02.

[13] Stackoverflow 2021 developer survey. `https://insights.stackoverflow.com/survey/2021` Na dzień 2022-04-27.

[14] Using promises. `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises` Na dzień 2022-05-02.

Dodatek A

Opis załączonej płyty CD/DVD

Dołączona płyta zawiera wszystkie pliki wykorzystane do stworzenia projektu, dokumentacji i niniejszej pracy. Pliki podzielono na następujące katalogi:

1. `.vscode` - ustawienia środowiska VsCode,