Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Центр післядипломної освіти
(повна назва)

Кафедра: програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

рівень вищої освіти: другий (магістерський)
(повна назва)

Дослідження ефективності застосування WebGL і D3.js
при створенні та використанні графічних елементів у веб-застосунках
(тема)

Виконала:
здобувачка 2 року навчання,
групи ІПЗздм-23-1
Наталія ПОНІКРОВСЬКА
(прізвище, ініціали)

Спеціальність: 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми: освітньо-наукова

Освітня програма: Інженерія програмного забезпечення

Керівник: доц. Наталія РУСАКОВА
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (власне ім’я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет: Центр післядипломної освіти

Кафедра: програмної інженерії

Рівень вищої освіти: другий (магістерський)

Спеціальність: 121 – Інженерія програмного забезпечення

Тип програми: освітньо-наукова програма

Освітня програма: Інженерія програмного забезпечення

(шифр і назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

“____” _____ 2025 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві: Понікаровській Наталії Андріївні

(прізвище, ім’я, по батькові)

1. Тема роботи: “Дослідження ефективності застосування WebGL і D3.js при створенні та використанні графічних елементів у веб-застосунках”
Затверджена наказом по університету від 21 квітня 2025 р. № 61Стз

2. Термін подання здобувачем роботи до екзаменаційної комісії 11.06.2025

3. Вихідні дані до роботи: *операційна система Ubuntu 18.04.6 LTS, веб-браузер Google Chrome, середовище розробки Visual Studio Code, бібліотеки WebGL та D3.js, тестові набори даних у форматі JSON.*

4. Перелік питань, що потрібно опрацювати в роботі: *провести аналіз предметної області, сформулювати критерії оцінювання та побудувати ієрархічну модель прийняття рішень за методологією MCDA, провести тестування технологій за обраними критеріями, здійснити аналіз чутливості, сформулювати практичні рекомендації щодо вибору технологій візуалізації залежно від вимог проєкту*.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	04.02.2025	*виконано*
2	Аналіз предметної галузі і постановка задачі	18.02.2025	*виконано*
3	Експериментальні дослідження	14.03.2025	*виконано*
4	Аналіз результатів експериментальних досліджень та розробка рекомендацій	31.03.2025	*виконано*
5	Підготовка до апробації результатів дослідження. Публікація матеріалів	24.04.2025	*виконано*
6	Підготовка пояснювальної записки	23.05.2025	*виконано*
7	Підготовка презентації та доповіді	30.05.2025	*виконано*
8	Перевірка на плагіат	02.06.2025	*виконано*
9	Нормоконтроль	04.06.2025	*виконано*
10	Рецензування	05.06.2025	*виконано*
11	Попередній захист	06.06.2025	*виконано*
12	Занесення диплома в електронний архів	06.06.2025	*виконано*
13	Допуск до захисту у зав. кафедри	07.06.2025	*виконано*

Дата видачі завдання 3 лютого 2025р.

Здобувач _____ Наталія ПОНІКРОВСЬКА
(підпис)

Керівник роботи _____ доц. Наталія РУСАКОВА
(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить 86 с., 16 рис., 12 табл, 51 джер.

БАГАТОКРИТЕРІАЛЬНИЙ АНАЛІЗ, ВІЗУАЛІЗАЦІЯ, ВЕЛИКИЙ НАБІР ДАНИХ, КРИТЕРІЙ, D3.JS, WEBGL, MCDA

Об’єктом дослідження є технології візуалізації великих обсягів даних у веб-застосунках.

Метою дослідження є багатокритеріальний аналіз ефективності технологій WebGL та D3.js для візуалізації великих обсягів даних у веб-застосунках.

У дослідженні застосовано методологію багатокритеріального аналізу (MCDA), що включає структуризацію задачі, формування критеріїв оцінювання, агрегацію отриманих результатів та аналіз чутливості.

У результаті проведеного дослідження здійснено комплексну багатокритеріальну оцінку ефективності технологій WebGL та D3.js, що застосовуються для візуалізації великих обсягів даних у веб-середовищі. На основі сформованої методологічної моделі порівняння розроблено практичні рекомендації щодо вибору технології відповідно до специфіки задачі, вимог до продуктивності та контексту використання веб-застосунку.

MULTI-CRITERION ANALYSIS, VISUALIZATION, LARGE DATASET, CRITERIA, D3.JS, WEBGL, MCDA.

The object of the research is visualization technologies for large-scale data in web applications.

The aim of the research is to perform a multi-criteria analysis of the effectiveness of WebGL and D3.js technologies for visualizing large volumes of data in web applications.

The study employs a multi-criteria decision analysis (MCDA) methodology, which includes problem structuring, formulation of evaluation criteria, aggregation of results, and sensitivity analysis.

As a result of the conducted research, a comprehensive multi-criteria evaluation of the effectiveness of WebGL and D3.js technologies, used for visualizing large volumes of data in web environments, was carried out. Based on the developed methodological comparison model, practical recommendations were formulated for selecting the appropriate technology according to the task’s specifics, performance requirements, and the context of use within a web application.

Завідувачу кафедри ПІ
проф. Кирилу СМЕЛЯКОВУ

ЗАЯВА

щодо самостійності виконання кваліфікаційної роботи та можливості її публікації (та/або публікації анотації кваліфікаційної роботи) в електронному архіві відкритого доступу ElAr KhNURE

Я, Понікаровська Наталія Андріївна, здобувач вищої освіти на другому (магістерському) рівні вищої освіти академічної групи ІПЗздм-23-1 кафедра програмної інженерії, заявляю: моя кваліфікаційна робота на тему «Дослідження ефективності застосування WebGL і D3.js при створенні та використанні графічних елементів у веб-застосунках», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в репозиторії "ElArKhNURE". Погоджуюся з авторським договором, відповідно до Положення про репозиторій ХНУРЕ "ElArKhNURE". Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з вимогами академічної доброчесності, згідно з якими виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

05.06.2025 Наталія ПОНІКАРОВСЬКА

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

WebGL – Web Graphics Library.

D3 – Data-Driven Documents.

CSS – Cascading Style Sheets (каскадні таблиці стилів).

XML – EXtensible Markup Language.

HTML – HyperText Markup Language.

SPA – Single Page Application.

GPU – Graphics Processing Unit.

CPU – Central Processing Unit.

MCDA – Multi-Criteria Decision Analysis.

AHP – Analytic hierarchy process.

ПЗ – програмне забезпечення.

IDE – Integrated Drive Electronics (інтегроване середовище розробки).

SVG – Scalable Vector Graphics.

API – Application Programming Interface.

FPS – Frames per second.

RT – Rendering Time.

ВСТУП

Візуалізація у веб-застосунках є одним із інструментів аналізу великих масивів даних, який дозволяє виявляти закономірності та аномалії, спрощує інтерпретацію результатів. Цю функцію чітко окреслено у визначенні мети візуалізації, запропонованому Майком Бостоком, автором бібліотеки D3.js: “The purpose of visualization is insight, not pictures” [1]. У сфері веб-розробки актуальною є задача вибору технології. Наразі існує велика кількість рішень, що ґрунтуються на різних технологічних підходах, але їхню ефективність складно оцінити без формалізованого аналізу. Багатокритеріальний підхід дозволяє проводити порівняння технологій за низкою показників, що мають прикладне значення.

Кваліфікаційна робота виконана в межах наукового напряму кафедри програмної інженерії “Інтелектуальний аналіз даних”, що охоплює розробку методів і засобів виявлення, обробки та інтерпретації залежностей у великих масивах інформації. Дослідження зосереджене на багатокритеріальному аналізі сучасних технологій візуалізації (D3.js та WebGL) у веб-середовищі як інструментів, що забезпечують ефективну підтримку процесів аналізу даних.

Метою дослідження є багатокритеріальний аналіз ефективності технологій WebGL та D3.js для візуалізації великих обсягів даних у веб-застосунках.

Задачі дослідження:

- проаналізувати предметну галузь та виявити актуальні проблеми візуалізації великих обсягів даних у веб-середовищі;
- дослідити функціональні можливості та архітектуру технологій D3.js і WebGL у контексті їх застосування для побудови графічних елементів у веб-застосунках;
- розробити графічні модулі на основі D3.js та WebGL, які реалізують однакові сценарії візуалізації;
- зібрати метрики продуктивності (RT, FPS, частка навантаження на GPU);

- здійснити багатокритеріальний аналіз ефективності технологій D3.js та WebGL;
- сформулювати практичні рекомендації щодо вибору технології візуалізації великих наборів даних у веб-середовищі.

Об’єктом дослідження є технології візуалізації великих обсягів даних у веб-застосунках.

Предметом дослідження є критерії та методи оцінки ефективності WebGL і D3.js для візуалізації даних у веб-застосунках.

Для досягнення мети використано методологію багатокритеріального аналізу (MCDA). Задачу структуровано у вигляді ієрархії. Вагові коефіцієнти визначено за допомогою матриць попарних порівнянь та пропорційного методу. Емпіричні дані зібрано шляхом інструментального вимірювання продуктивності реалізованих графічних модулів та фіксації релевантних метрик. Агрегацію виконано за допомогою лінійної адитивної згортки після нормалізації критеріїв за min–max. Проведено аналіз чутливості для оцінки стійкості результатів.

У роботі проведено багатокритеріальний аналіз технологій WebGL і D3.js для візуалізації великих обсягів даних у веб-середовищі. Побудовано ієрархічну модель оцінювання на основі методології MCDA, адаптовану до специфіки задачі. Результати поглиблюють підходи до вибору технологій візуалізації та сформували основу практичних рекомендацій щодо їх застосування з урахуванням вимог проекту.

Основні положення роботи було апробовано під час виступів на конференціях PIC S&T 2024 та eStream 2025. Сформульовані рекомендації можуть бути використані для вибору засобів візуалізації в веб-застосунках. Документальним підтвердженням впровадження є публікації [2, 3] за матеріалами доповідей, прийняті до друку у збірниках вказаних конференцій.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної галузі

Термін візуалізація, або унаочнення, походить від латинського слова visualis, що означає видимий, зоровий [4, с. 5]. Термін візуалізація означає “одержання (подання) видимого зображення яких-небудь предметів, явищ, процесів, недоступних для безпосереднього спостереження”, а візуалізувати - “формувати зоровий образ; представляти у візуальній формі” [5, с. 186].

Візуалізація у веб-застосунках - це процес створення графічного або інтерактивного представлення даних за допомогою веб-технологій (HTML, CSS та JavaScript, SVG, WebGL тощо). Основною метою створення візуалізації є надання користувачам можливості взаємодіяти з даними безпосередньо у браузері та забезпечити інтерактивність та зручність аналізу. Візуалізації у веб-застосунках не потребують спеціального програмного забезпечення (ПЗ) чи інсталяції браузерних розширень, вони доступні “із коробки”.

Візуалізація у веб-середовищі набувала розвитку разом із розвитком технологій. У 1990-х роках веб сторінки використовували простий HTML суто для представлення певної інформації (див. рис. 1.1).

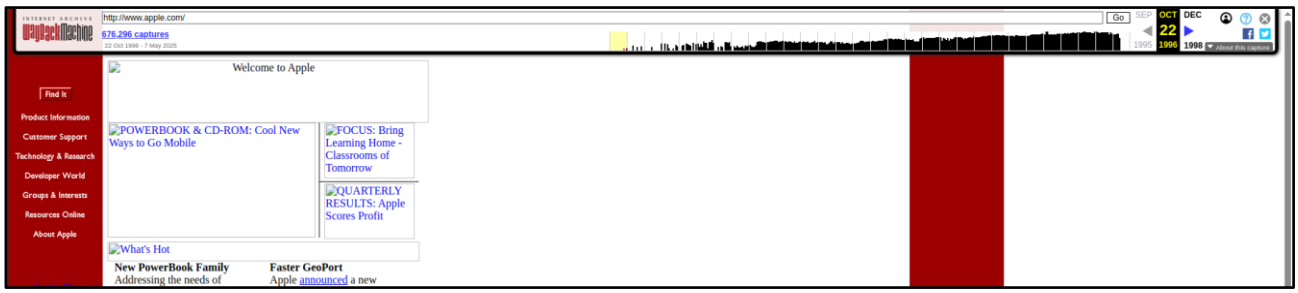

Рисунок 1.1 – Дизайн сайту apple у 1996 році [6]

У листопаді 1994 [7] року на Веб-конференції у Чікаго була презентована перша “чернетка” CSS. Роком пізніше компанія Netscape поставила за мету інтегрувати мову програмування в браузер Netscape Navigator, для чого було запрошено американського програміста Брендана Айка. Початкова концепція передбачала мову, схожу на Scheme. Згодом вона була розширена до скриптів, які можна інтегрувати в HTML для взаємодії з елементами сторінки. Третя бета-

версія браузера Netscape, після ряду вдосконалень та перейменувань, вже містила JavaScript у звичному вигляді (версія 1.1). Вигляд веб-сторінок суттєво змінився (див. рис. 1.2), з'явилась виражена структура, окремі навігаційні елементи, більш привабливе оздоблення.

Рисунок 1.2 – Дизайн сайту apple у 2000 році [6]

З 1999 [7] року World Wide Web Consortium починає розробку стандарту SVG, нової мови розмітки. SVG – це формат, у якому графіка описується через XML теги. Кожен елемент – лінія, коло, текст, задається у вигляді тегу - <line>, <circle>, <text> з певними атрибутами. Файли можна створювати вручну у IDE або текстовому редакторі для програмування. Пізніше технологія також була впроваджена у графічні редактори. SVG стала першим кроком до створення візуально привабливих та інтерактивних візуалізацій даних у веб-застосунках. Також SVG-формат має переваги через малу вагу зображень, оскільки зберігає інструкції для побудови графіки у вигляді текстових тегів, які можна анімувати або додати до них слухачів подій.

У 2004 [7] році було розроблено елемент <canvas>. Вперше його впровадила компанія Apple у браузері Safari для потреб платформи Mac OS Dashboard. Пізніше <canvas> увійшов до специфікації HTML5, яка почала формуватись у 2008 році, і поступово став стандартом для рендерингу динамічної графіки у браузерах.

З березня 2011 [7] відбувся офіційний реліз WebGL - веб-стандарту для створення 2D та 3D-графіки без використання плагінів. Стандарт був розроблений консорціумом Khronos Group за підтримки Mozilla Foundation. WebGL базується

на OpenGL ES 2.0 і дозволяє використовувати апаратне прискорення безпосередньо у браузері. Цей стандарт став революційним кроком у розвитку інтерактивної візуалізації у вебі, відкривши можливість створювати складні сцени, моделі та анімації.

Протягом останніх двадцяти років виникали й інші технології, зокрема Adobe Flash та Java-аплети. Але через обмежену безпеку, потребу в плагінах і слабку підтримку мобільних пристроїв ці рішення поступово втратили актуальність та поступилися відкритим стандартам, що забезпечують кращу продуктивність, масштабованість і нативну підтримку в сучасних браузерах.

Сучасні вбудовані можливості браузера, такі як WebAssembly (дозволяє виконувати у браузері код, що написаний на інших мовах програмування) та Web Workers API (дозволяє виконувати JavaScript-код у фоновому потоці без блокування основного інтерфейсу сторінки) продовжують розширювати можливості візуалізації у веб-застосунках. Разом сучасні технології надають потужності для обробки та відображення складних наборів даних у реальному часі та роблять візуалізацію більш доступною для широкого спектру галузей, таких як бізнес-аналітика, наукові дослідження, освіта, соціальні медіа та маркетинг, IoT, кібербезпека тощо.

1.2 Дослідження інструментів візуалізації у веб-застосунках

Аналіз ресурсів GitHub та npm показав, що число JavaScript-бібліотек для візуалізації, включно із вузькоспеціалізованими, перевищує 100. Для даного дослідження інтерес становлять бібліотеки для створення інтерактивних, динамічних та високопродуктивних графіків. Бібліотеки відрізняються за різноманіттям пропонованих типів чартів, технологіями, на яких вони базуються (SVG, Canvas, гібридні рішення) та інтерфейсом.

Особливістю переважної більшості подібних JavaScript-бібліотек є конфігураційний підхід до реалізації інтерфейсу (див. рис. 1.3). Даний підхід передбачає налаштування візуалізації через параметри (об'єкт опцій), що описує тип, дані, осі, стилі, інтерактивну поведінку графіку тощо. Його особливістю є

розміщення основної логіки рендерингу всередині бібліотеки, а її зміна є немоюживою без втручання безпосередньо у код бібліотеки.

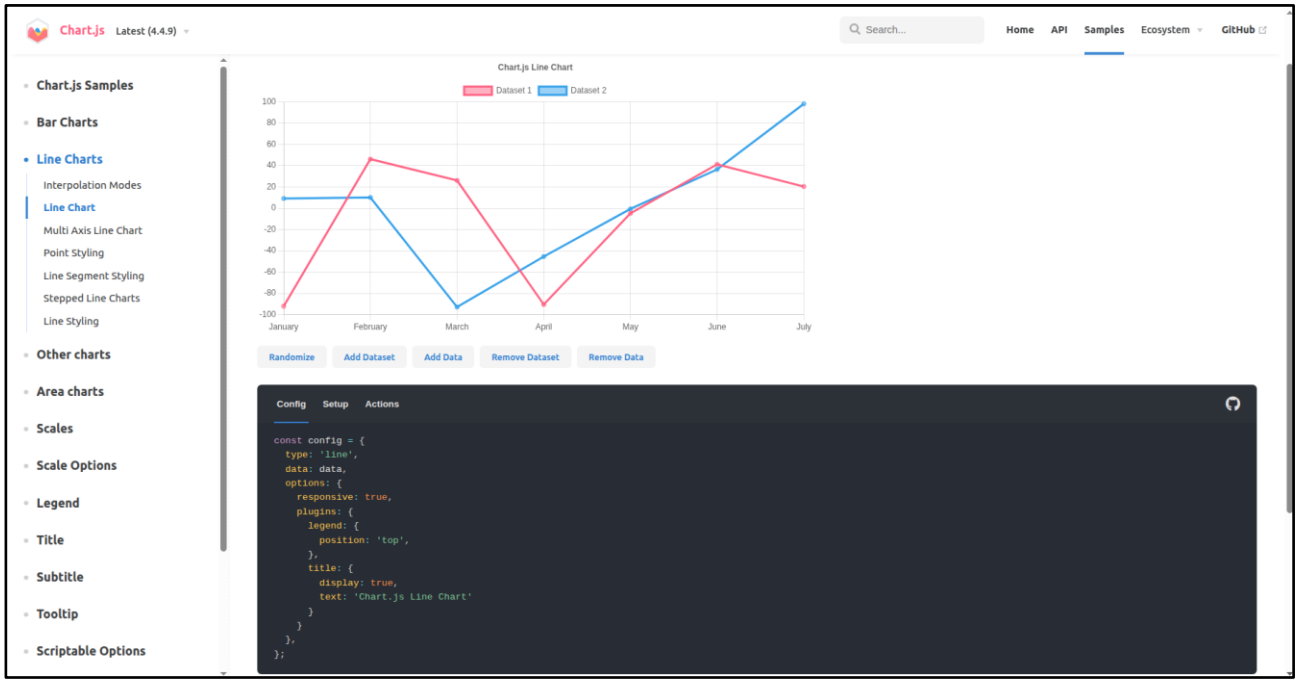

Рисунок 1.3 – Приклад реалізації конфігураційного підходу.
Документація Chart.js [8]

Перевагою конфігураційного підходу є швидкість розробки та простота використання. Недоліком є обмежена гнучкість: реалізація складних або нестандартних візуалізації є неможливою, якщо бібліотека не передбачає відповідних параметрів, елементів або поведінки. Прикладами таких бібліотек є: C3.js [9], Billboard.js [10], Dygraphs [11], Apache ECharts [12], Chart.js [8] тощо.

З розвитком концепції SPA конфігураційних підхід був адаптований для створення чартів через декларативну структуру JSX, що дозволяє інтегрувати візуалізацію безпосередньо у логіку SPA фреймворку, зберегти простоту конфігурації та гнучкість реактивного інтерфейсу. Прикладами подібних рішень для React.js є Recharts [13], Victory [14] тощо. Також розповсюдженою практикою є адаптація існуючих бібліотек для SPA фреймворків, наприклад, для Vue.js: vue-chartjs [15], vue-echarts [16] тощо.

JavaScript-бібліотеки, що реалізують конструкторський підхід, представлені у значно меншій кількості через, вірогідно, вищу складність використання.

Конструкторський підхід потребує глибшого розуміння принципів побудови графіків (шкала, координати, геометрія фігур тощо), що значно підвищує вхідний поріг і ускладнює реалізацію типових завдань. Попри високу гнучкість, такі інструменти використовуються переважно в складних або нестандартних випадках, коли бібліотеки із конфігураційним інтерфейсом не забезпечують необхідного рівня контролю або їх можливості не покривають вимог проекту. Прикладами є visx [17], RawGraphs [18], D3.js [19].

Ще менш розповсюдженим підходом для створення візуалізації у веб-застосунках є використання безпосередньо низькорівневого API, який передбачає пряме управління графічним рендерингом через інструкції до GPU. Цей підхід дає максимальну продуктивність і повний контроль над відображенням, але вимагає значно більше знань та часу, оскільки розробник самостійно визначає геометрію, кольори, освітлення, трансформації та логіку оновлення сцени без готових компонентів чи абстракцій, працюючи безпосередньо з шейдерами та буферами WebGL.

Використання низькорівневих інструментів дозволяє створювати складні, високопродуктивні та візуально насичені динамічні сцени з глибокою взаємодією та реалістичною графікою. Саме завдяки повному контролю над рендерингом ці технології широко застосовуються у створенні 3D-ігор, симуляцій, візуалізації великих обсягів наукових чи технічних даних, а також у графічно складних інтерактивних веб-застосунках.

Деякі сучасні бібліотеки для візуалізації у вебі використовують WebGL як графічний рушій: three.js [20], regl [21], deck.gl [22], plotly.js [23]. Але варто зазначити, що серед бібліотек, які працюють на базі WebGL, фактично відсутні ті, що реалізують конструкторський підхід для побудови графіків. Переважна більшість таких інструментів орієнтовані на конфігураційний інтерфейс, що значно спрощує використання WebGL, але обмежує гнучкість при створенні низькорівневих графічних зв’язків між даними та візуальними елементами.

1.3 Постановка задачі

Для візуалізації великих обсягів даних у веб-застосунках необхідні технології, здатні забезпечити їх ефективну обробку та рендеринг у режимі реального часу. У цьому контексті вибір відповідного інструменту має принципове значення, оскільки якість візуалізації безпосередньо впливає на здатність користувача інтерпретувати зв’язки, виявляти патерни та розпізнавати аномалії в даних. Водночас розробка будь-якого програмного рішення відбувається в умовах обмежених ресурсів. Для керівника проекту важливим є не лише створення функціональної та надійної системи, але й оцінка витрат на її реалізацію, рівня складності впровадження, а також тривалості навчання розробників.

Таким чином маємо коло питань, які значно впливають на процес прийняття рішень. А саме:

- обмежену продуктивність традиційних рішень при візуалізації великих об'ємів даних у веб-застосунках;
- рівень технічної складності застосування певних технологій;
- час навчання розробників;
- довготривалість рішення.

З огляду на велику кількість чинників, що впливають на вибір технології візуалізації, виникає необхідність у системному порівнянні на основі узгоджених критеріїв. Відтак, задача дослідження полягає у проведенні багатокритеріального аналізу ефективності застосування WebGL і D3.js для створення та використання графічних елементів у веб-застосунках у різних сценаріях використання.

2 МЕТОДОЛОГІЯ ДОСЛІДЖЕННЯ

Прийняття рішень у складних багатофакторних контекстах вимагає інтеграції численних критеріїв різної природи та важливості та ускладнюється ризиком суб'єктивності. Застосування певної методології прийняття рішень дозволяє об'єднати всі аспекти задачі в єдину систему, формалізувати суб’єктивні судження у кількісні показники та мінімізувати упередженість. Для вирішення поставленої задачі пропонується застосування методології MCDA [24], яка забезпечує системний підхід до інтеграції різноманітних показників у єдиний зведений індекс. Проблеми, що потребують застосування MCDA зазвичай є багатофакторними, тому дивіз “divide and conquer” [25, с. 14] по суті є основою даного підходу. Її впровадження передбачає реалізацію 4 кроків.

Перший крок - визначення типу завдання, структуризація проблеми та встановлення пріоритетів.

У MCDA представлено наступні типи основних завдань [24, с.3]:

– вибір (спрямоване на визначення найкращої альтернативи з множини можливих варіантів);
– ранжування (передбачає упорядкування альтернатив за ступенем їхньої відповідності критеріям, від найкращої до найгіршої);
– сортування (передбачає класифікацію альтернатив за попередньо визначеними категоріями чи групами, наприклад, групування технологій за рівнем складності або продуктивності);
– опис (спрямоване на детальну характеристику альтернатив та їх оцінку за критеріями без необхідності вибору, ранжування чи сортування).

Проблема структурується у вигляді ієрархії, де на найвищому рівні знаходиться мета ухвалення рішення, другий рівень відповідає критеріям, а найнижчий – альтернативам. Структуризація в MCDA проводиться з метою чіткого формулювання основної цілі, визначення критеріїв та альтернатив, тобто перетворення проблеми у логічну ієрархію.

Визначення пріоритетів відбувається через побудову матриць попарних

порівнянь, де кожен елемент порівнюється з іншим за ступенем переваги щодо певного критерію. На основі цих матриць обчислюються ваги, які відображають відносну важливість альтернатив або критеріїв у контексті прийняття рішення.

Другий крок – опис критеріїв. На цьому етапі здійснюється детальний опис та формалізація критеріїв оцінювання. З метою забезпечення об'єктивності порівняння, для кожного критерію визначаються відповідні кількісні або якісні метрики, які можуть бути отримані шляхом емпіричного вимірювання, аналітичного аналізу або документаційного огляду.

Критерій у контексті MCDA – це формалізований параметр або характеристика, за якою здійснюється оцінювання альтернатив відносно поставленої мети. Він виступає проміжною ланкою між загальним завданням вибору та множиною можливих варіантів рішення на основі узгоджених підстав. Наявність чітко визначених і релевантних критеріїв є ключовою передумовою для об’єктивного аналізу, оскільки саме через них реалізується порівнюваність альтернатив.

Третій крок – агрегація результатів. На цьому етапі здійснюється об’єднання оцінок альтернатив за всіма критеріями з метою отримання загального показника, який дозволяє впорядкувати варіанти за ступенем відповідності загальній меті.

Для цього етапу застосовуються допоміжні методи обробки даних. Зокрема, для нормалізації значень критеріїв використовується підхід із урахуванням min та max, що дозволяє привести всі показники до єдиної шкали та забезпечити коректність порівняння між критеріями, значення яких можуть мати різну природу та одиниці виміру.

Стандартна нормалізація до шкали 0;1 є фундаментальною для вагового аналізу. Проте в умовах даного дослідження було важливо зберегти вплив кожного критерію, навіть за мінімального значення, тому метод було адаптовано таким чином, щоб уникнути появи нульових значень (формула 2.1):

$$f = \frac{f_{meas} - f_{min}}{f_{max} - f_{min}} * 0.9 + 0.1 \quad (2.1)$$

де: f – нормалізоване значення критерію,

f_{meas} – фактичне (виміряне) значення критерію,

f_{min} – мінімальне значення критерію в межах вибірки,

f_{max} – максимальне значення критерію в межах вибірки/

Четвертий крок – аналіз чутливості. Його мета полягає у дослідженні змін вагових коефіцієнтів критеріїв залежно від сценаріїв використання та оцінці впливу таких змін на підсумкове ранжування альтернатив. Такий аналіз дозволяє виявити порогові значення, за яких змінюється пріоритетність варіантів, та оцінити стійкість прийнятого рішення до коливань ваг. У прикладних умовах це забезпечує адаптацію моделі до конкретного контексту задачі та підвищує обґрунтованість результатів. Для встановлення відносної важливості критеріїв застосовується пропорційний метод визначення вагових коефіцієнтів.

3 ОПИС АЛЬТЕРНАТИВ

У процесі дослідження на підготовчому етапі було розглянуто 30 JavaScript-бібліотек для візуалізації великих наборів даних у веб-застосунках: C3.js, Billboard.js, Dygraphs, Apache ECharts, Chart.js, Recharts, Victory, AnyChart, ZoomChart, visx, RawGraphs, D3.js, three.js, regl, deck.gl, plotly.js, Highcharts, amCharts, FusionCharts, ApexCharts, ZingChart, Chartist.js, TauCharts, G2Plot, LightningChart, Plottable.js, CanvasJS, WebGL, Taucharts-plus.

Альтернативи були порівняні за такими критеріями, як доступність навчання, підтримка в браузерах, популярність серед розробників, кількість підтримуваних типів чартів та можливість розширення. Після застосування методу Парето-оптимізації для виключення рішень, які поступаються іншим за всіма параметрами, у фінальній множині залишилися лише D3.js та WebGL як інструменти, що демонструють найвищу гнучкість, широку підтримку в спільноті та практично необмежені можливості у створенні кастомізованих графіків.

3.1 D3.js

JavaScript-бібліотека D3.js [19] є однією з найвідоміших технологій у сфері веб-візуалізації. Вона була створена у 2011 році Майком Бостоком (Mike Bostock) в межах проєкту Stanford Visualization Group. Основна мета розробки полягала у створенні гнучкого інструмента для зв’язування даних з документами на основі DOM та SVG, що дозволяє реалізовувати висококастомізовані інтерактивні візуалізації.

Бібліотека активно використовується у наукових візуалізаціях, журналістиці, аналітиці, урбаністиці, біоінформатиці, освітніх платформах тощо. Серед відомих прикладів використання D3.js веб-сайти газети The New York Times [25] (див. рис. 3.1), газети The Guardian [26], онлайн-платформа для створення, обміну та публікації інтерактивних візуалізацій та аналітичних нотаток Observable [27], онлайн-платформа популяризації фактів і статистичних даних про світ Gapminder [28], інтерактивні звіти OECD та численні академічні проекти.

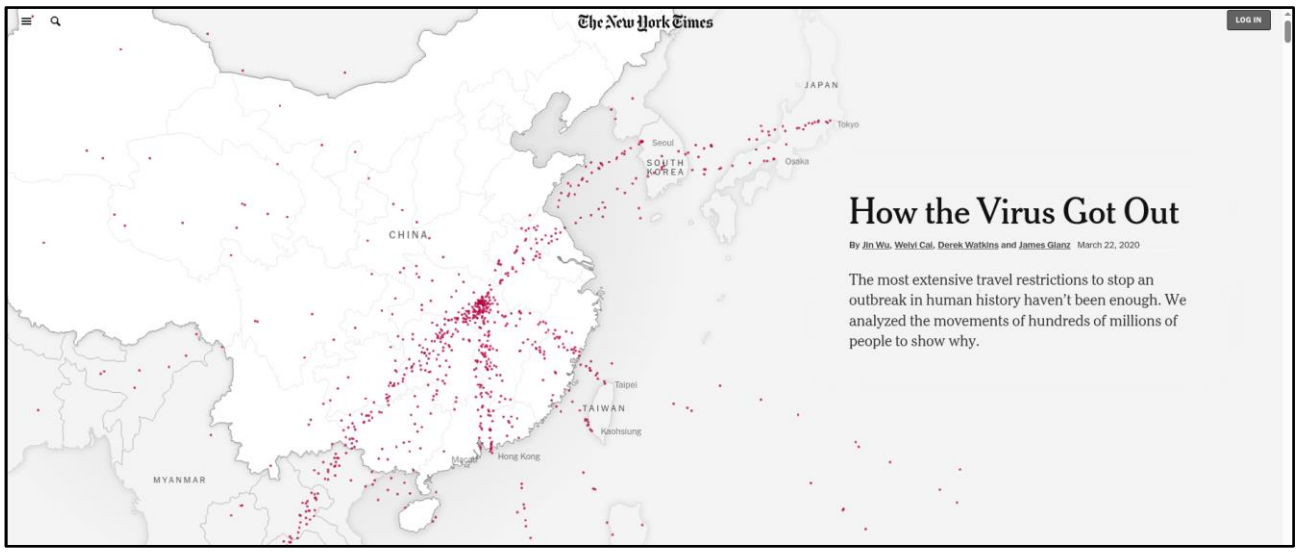

Рисунок 3.1 – Інтерактивна візуалізація поширення коронавірусу реалізована з використанням D3.js [25]

D3.js ґрунтується на конструкторському підході, що забезпечує розробнику повний контроль над усіма аспектами візуалізації - від структури та масштабів до анімацій і взаємодії. Як приклад розглянемо процес створення простої стовпчикової діаграми (bar chart).

Першим кроком створюється основне “полотно” графіку, визначаються ширина та висота:

```
const svg = d3
      .select("#chart")
      .append("svg")
        .attr("width", width + margin.left + margin.right)
        .attr("height", height + margin.top + margin.bottom);
```

Змінна xScale задає шкалу для осі X. Вона використовує метод d3.scaleBand() (підходить для категоріальних значень). У даному прикладі доменною областю цієї шкали є індекси кожного елемента в масиві даних. Діапазон шкали - це координати від лівого до правого краю графіка з урахуванням зовнішніх відступів (margin.left і width - margin.right). Для більш естетичного вигляду додається внутрішній відступ (padding(0.1)).

```
const xScale = d3.scaleBand()
```

```
        .domain(data.map((_, i) => i))
        .range([margin.left, width - margin.right])
        .padding(0.1);
```

Змінна yScale – це лінійна шкала для осі Y, яка використовується для відображення значень висоти барів. Доменною областю є діапазон від 0 до максимального значення з масиву даних. Вивід у пікселях (range) - це координати від нижнього краю графіка (height - margin.bottom) до верхнього (margin.top).

```
const yScale = d3.scaleLinear()
        .domain([0, d3.max(data, d => d.value)])
        .range([height - margin.bottom, margin.top]);
```

Шкали використовуються для перетворення абстрактних значень з масиву даних у конкретні координати на SVG-полотні.

У наступному фрагменті коду до SVG-елемента додаються дві групи <g>, які відповідають за побудову вісей графіка: горизонтальної (X) і вертикальної (Y).

```
svg.append("g")
        .attr("class", "x-axis")
        .attr("transform", `translate(0,${height - margin.bottom})`)
        .call(d3.axisBottom(xScale).tickFormat(i => `${i + 1}`));

svg.append("g")
        .attr("class", "y-axis")
        .attr("transform", `translate(${margin.left},0)`)
        .call(d3.axisLeft(yScale));
```

Перша частина коду створює групу для вісі X. Вона додається до SVG з класом “x-axis” і розташовується на висоті, яка дорівнює загальній висоті SVG мінус нижній відступ (margin.bottom). За допомогою методу call() до цієї групи застосовується функція d3.axisBottom(xScale), яка створює нижню вісь на основі шкали xScale.

Друга частина коду створює групу для вісі Y. Вона також додається до SVG з класом “y-axis” і зміщується праворуч на величину лівого відступу (margin.left), що дозволяє розмістити вертикальну вісь точно біля початку області з барчартом. Так само, як і з віссю X, метод call() викликає генератор осі d3.axisLeft(yScale),

який малює ліву вертикальну шкалу з позначками відповідно до значень шкали yScale.

Наступний фрагмент коду відповідає за створення прямокутників (барів) на графіку на основі масиву data. Він використовує типовий для D3 підхід до роботи з даними: вибір елементів, прив’язка даних, об'єднання та встановлення атрибутів.

```
svg.selectAll(".bar")
      .data(data)
      .join("rect")
      .attr("class", "bar")
      .attr("x", (d, i) => xScale(i))
      .attr("y", d => yScale(d.value))
      .attr("width", xScale.bandwidth())
      .attr("height", d => yScale(0) - yScale(d.value));
```

За допомогою svg.selectAll(“.bar”) здійснюється пошук усіх існуючих елементів з класом "bar" на SVG-полотні. На цьому етапі таких елементів ще існує, але ця конструкція необхідна для наступного кроку. Метод .data(data) прив’язує масив даних до майбутніх елементів. Кожен елемент масиву data відповідає одному прямокутнику. Метод .join(“rect”) створює нові елементи <rect>, якщо вони ще не існують, і одразу вставляє їх у DOM. Далі встановлюються атрибути для кожного прямокутника:

- class: задається клас “bar” для стилізації через CSS та подальшої вибірки елементів у D3;
- x: координата розташування прямокутника по осі X, яка визначається за допомогою шкали xScale(i), де i - індекс елемента у масиві;
- y: координата верхнього краю прямокутника по осі Y, розрахована за допомогою шкали yScale(d.value), яка враховує величину значення;
- width: ширина прямокутника визначається як xScale.bandwidth(), що є стандартною шириною для шкали scaleBand;
- height: обчислюється як різниця між базовою лінією (тобто yScale(0)) і позицією верхнього краю прямокутника.

Тобто, остання частина коду візуалізує всі дані як вертикальні стовпчики (бари) масштабовані відповідно до значень у масиві.

Наведений приклад (див. рис. 3.2) демонструє, що побудова стовпчикової діаграми за допомогою D3.js є відносно простою й інтуїтивно зрозумілою, а бібліотека забезпечує надзвичайну гнучкість і точний контроль над кожним графічним елементом.

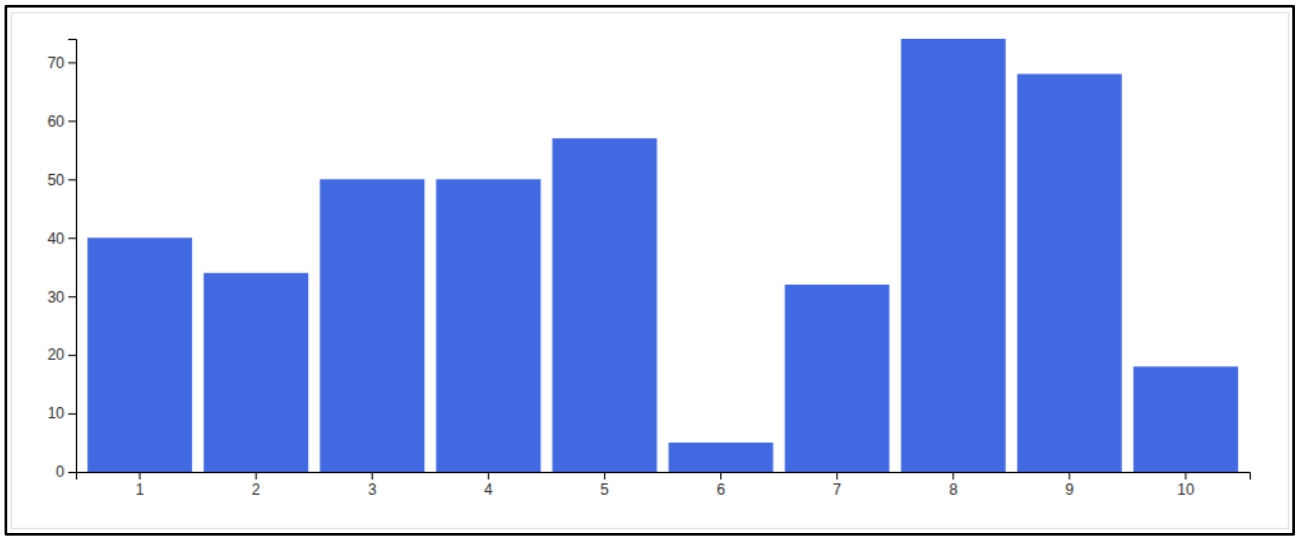

Рисунок 3.2 – Результат реалізації стовпчикової діаграми на D3.js (рисунок виконаний самостійно)

Значущість D3.js для розвитку візуалізації у веб-застосунках полягає в тому, що вона не лише популяризувала декларативний опис даних, а й створила фундамент для численних похідних бібліотек і фреймворків. Її поява ознаменувала перехід від статичних або жорстко структурованих графіків до динамічних, гнучких, інтерактивних і повністю керованих даними інтерфейсів.

3.2 WebGL

WebGL – це низькорівнева JavaScript-технологія для рендерингу 2D та 3D графіки без потреби в додаткових плагінах. WebGL широко використовується у візуалізації наукових даних, геопросторовому аналізі, освіті, інженерних симуляціях та веб-аналітиці. Дана технологія стала базою для геймдеву у вебі (див. рис. 3.3). Серед прикладів використання інтерактивна платформа Cesium для 3D-картографії [29], освітня WebGL-лабораторія від NASA [30], онлайн-

платформи візуалізації великих масивів даних deck.gl [22], а також численні експериментальні та академічні проекти.

Рисунок 3.3 – Ігрові моделі створені на Three.js [20]

На відміну від високорівневих бібліотек, WebGL не надає готових елементів інтерфейсу, розробник має самостійно оперувати геометрією, шейдерами та буферами. Як приклад також розглянемо процес створення простої стовпчикової діаграми (bar chart).

Спочатку створюється масив вершин для рендерингу барів. Кожен бар будується як два трикутники, координати яких трансформуються у простір WebGL (від -1 до 1). WebGL (як і OpenGL, на якому він базується) може напряму рендерити лише трикутники, лінії або точки.

```
vertices.push(x1, y1, x2, y1, x1, y2);
vertices.push(x1, y2, x2, y1, x2, y2);
```

Вершини – це координати прямокутників, які представляють кожен стовпчик графіка. Далі створюється буфер для цих вершин і передається у GPU:

```
const vertexBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
```

```
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);
```

gl.createBuffer створює буферну структуру у GPU, gl.bindBuffer робить її поточною, а gl.bufferData записує в неї масив координат. Після цього створюються два шейдери: вершинний (vertex shader) і фрагментний (fragment shader). Вершинний визначає, як розташувати кожну точку:

```
attribute vec2 a_position;
void main() {
  gl_Position = vec4(a_position, 0, 1);
}
```

Це базовий шейдер, який просто передає координати WebGL. Фрагментний шейдер визначає колір кожного фрагмента:

```
void main() {
  gl_FragColor = vec4(0.27, 0.51, 0.71, 1);
}
```

Наступний фрагмент коду створює виконувану WebGL-програму. vs і fs - це попередньо скомпільовані вершинний і фрагментний шейдери.

```
const program = gl.createProgram();
gl.attachShader(program, vs);
gl.attachShader(program, fs);
gl.linkProgram(program);
gl.useProgram(program);
```

Далі WebGL зв’язує атрибути шейдера з буфером даних:

```
const posAttrib = gl.getAttribLocation(program, "a_position");
gl.enableVertexAttribArray(posAttrib);
gl.vertexAttribPointer(posAttrib, 2, gl.FLOAT, false, 0, 0);
```

Кожна вершина має 2 координати типу FLOAT, і вони передаються у атрибут a_position. Перед рендерингом задається фоновий колір, очищується полотно та відбувається рендеринг:

```
gl.clearColor(1, 1, 1, 1); // білий фон
gl.clear(gl.COLOR_BUFFER_BIT);
gl.drawArrays(gl.TRIANGLES, 0, vertices.length / 2);
```

Попри схожий кінцевий результат, реалізація стовпчикової діаграми на WebGL є суттєво складнішою за D3.js як за обсягом коду, так і за архітектурою. У WebGL відсутні будь-які високорівневі абстракції для візуалізації даних, тому навіть базові графічні компоненти, такі як прямокутники, координатні осі та підписи, потрібно будувати вручну. Для кожної колонки діаграми необхідно розраховувати положення її вершин у нормалізованих координатах (clip space), а також трансформувати їх із піксельного простору. Візуалізація одного прямокутника потребує створення двох трикутників, тобто шести вершинних координат, що перетворює навіть просту діаграму на масивну структуру з великою кількістю вручну сформованих вершин.

Додатково потрібно явно створити та скомпілювати шейдерні програми (vertex і fragment), ініціалізувати буферну пам’ять на GPU, налаштувати атрибути, активувати потрібні режими та викликати метод gl.drawArrays для відтворення геометрії. Оскільки WebGL не надає жодних засобів для рендерингу тексту або сітки координат, побудову осей, міток і підписів доводиться реалізовувати окремо через додатковий canvas з 2D-контекстом. D3.js автоматизує більшість з цих завдань: масштаби, осі, підписи, координати, прямокутники, а розмітка і відмальовка реалізовані через декларативні конструкції.

Ще однією особливістю роботи з WebGL є те, що шейдери визначаються у вигляді рядкових літералів (template literals) безпосередньо у JavaScript-коді. Такий підхід є необхідним через архітектуру WebGL, яка не підтримує окремі шейдерні файли у вигляді модулів, що, в свою чергу, створює певні складнощі при розробці: відсутність синтаксичної підсвітки, автодоповнення та статичної перевірки у більшості середовищ розробки. Помилки компіляції виводяться лише під час виконання коду, що значно знижує швидкість розробки порівняно з високорівневими бібліотеками або окремими GLSL-файлами. Ускладнюється

також повторне використання або масштабування таких фрагментів коду в рамках більших застосунків.

WebGL-реалізація (див. рис. 3.4) вимагає майже вдвічі більше коду (3.7kB проти 2.0kB), глибоких знань щодо трансформацій, буферизації, шейдерної мови GLSL і узгодження між системами координат, що істотно ускладнює не лише реалізацію, а й налагодження та підтримку навіть у випадку простих статичних візуалізацій.

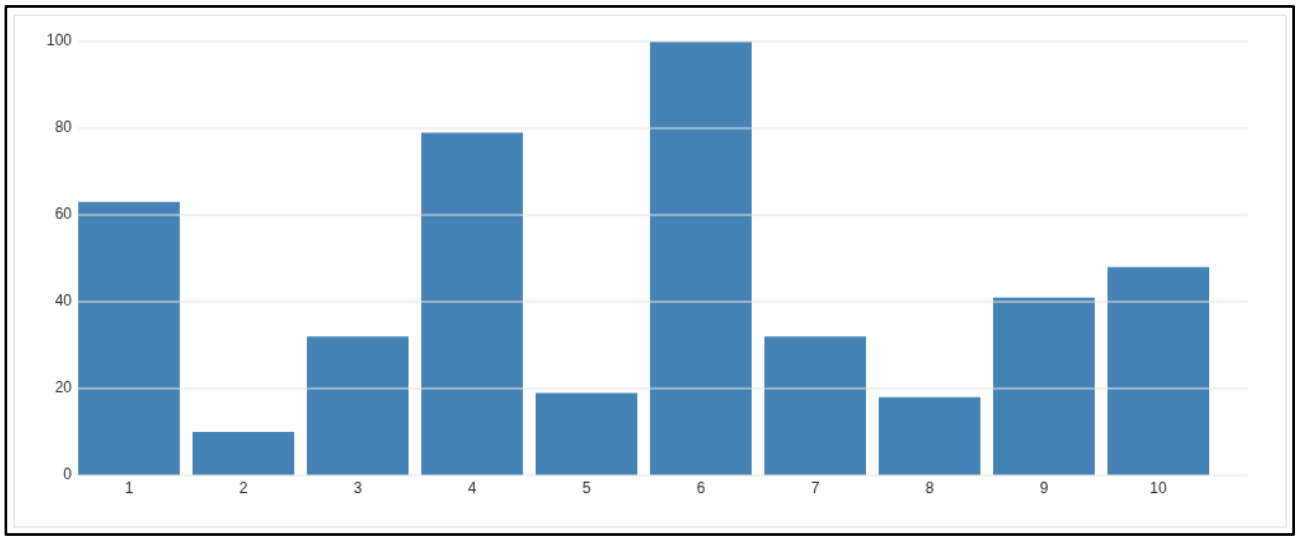

Рисунок 3.4 – Результат реалізації Scatter Plot на WebGL
(рисунок виконаний самостійно)

Незважаючи на складність базової реалізації, WebGL залишається надзвичайно потужною технологією, яка відкриває широкі можливості для створення складних, високопродуктивних візуалізацій. Її поява стала справжнім проривом у сфері веб-графіки, адже вперше дала змогу повноцінно використовувати ресурси GPU без встановлення додаткових плагінів або програм.

3.3 Підходи до використання D3.js і WebGL

В сучасних дослідженнях, присвячених бібліотеці D3.js, візуалізація даних розглядається як процес інтеграції структурованої інформації з можливостями керування DOM-елементами. У роботах [31, 32] автори зосереджені на декларативному стилі, що базується на принципі прив’язки даних до графічних

елементів, і який дозволяє створювати як прості, так і складні графіки шляхом трансформації масивів даних у SVG або HTML-структури. У [33] описаний підхід поглиблюється за рахунок впровадження принципів модульності, повторного використання коду та інкапсуляції, що дозволяє створювати масштабовані та підтримувані компоненти, інтегровані у великі веб-застосунки.

Дослідники [34, 35] пропонують підходи, орієнтовані на взаємодію з користувачем та акцентують увагу на інтерактивності, гнучкості візуальних рішень і здатності D3.js адаптуватися до змін у даних у реальному часі. Значну роль відіграє концепція “візуалізація як історія”, за якою дані необхідно організовувати у певну послідовність, що має аналітичну або пояснювальну мету. У даному підході передбачено не лише технічну реалізацію графіків і діаграм, а й стратегічне планування візуальної комунікації, що забезпечує кращу зрозумілість, доступність та ефективність сприйняття інформації.

Підходи до розробки візуалізацій з використанням WebGL передбачають імперативний стиль програмування, що надає розробникам повний контроль над графічним конвеєром. У роботі [36] автори пропонують поетапний підхід до створення інтерактивної 3D-графіки, який охоплює базові концепції роботи з шейдерами, буферами, текстурами та трансформаціями. Такий стиль вимагає від розробника глибокого розуміння роботи GPU та архітектури WebGL. У джерелах [37, 38] увагу приділено вивченню фундаментальних принципів WebGL API та системному підходу до побудови візуалізацій. Внаслідок відсутності вбудованих абстракцій та високого рівня складності, ефективна реалізація можлива лише за умови чіткого архітектурного планування. Робота [39] доповнює технічну площину аналізом безпекових ризиків, що виникають при використанні WebGL у відкритому веб-середовищі.

Сучасні наукові дослідження формують актуальний контекст для розвитку візуалізації у веб-застосунках та охоплюють широкий спектр питань: обробка даних, адаптивність, проблеми локалізації, тонкощі користувацької взаємодії тощо. У роботі [40] обґрунтовано застосування логічних асоціативних структур для аналізу та синтезу даних, що відкриває перспективи для оптимізації процесів

обробки великих інформаційних масивів, зокрема у візуалізаційних задачах. Дослідження [41] пропонує системну методику оцінки відкритих СУБД, яка сприяє вибору оптимальних рішень для інтеграції з інструментами візуалізації. Робота [42] пропонує адаптивний алгоритм кластеризації зображень у просторі RGB, застосування якого є ефективним інструментом для попередньої обробки візуальних даних у веб-застосунках.

У дослідженні [43] проаналізовано можливості трекінгу погляду з використанням веб-камер, що серед іншого дає змогу створювати адаптивні інтерфейси візуалізацій, зокрема теплові карти уваги. У роботі [44] представлено системний підхід до оцінки юзабіліті та побудови користувацько-орієнтованого дизайну, який дозволяє знизити когнітивне навантаження при взаємодії з графічними інтерфейсами. У [45] розглянуто проблему локалізації великих даних, що є надзвичайно важливою для розробки мультимовних візуалізацій, орієнтованих на різні культурні середовища.

Отже, підходи до веб-візуалізації сьогодні базуються на поєднанні гнучких інструментів D3.js із високопродуктивними рішеннями на основі WebGL. У контексті актуальних досліджень, що охоплюють питання обробки, адаптивності, користувацької взаємодії та локалізації, ці інструменти отримують нові можливості для ефективної інтеграції у складні інформаційні системи.

4 ЗБІР ТА ФОРМАЛІЗАЦІЯ МЕТРИК D3.JS І WEBGL ДЛЯ MCDA

4.1 Реалізація графічних модулів

Згідно з задачами дослідження було створено два графічні модулі для побудови графіків розсіювання (Scatter Plot) з використанням технологій D3.js та WebGL. Програмні засоби реалізовано мовою JavaScript (стандарт ECMAScript 2015).

Структура ПЗ (див. рис. 4.1):

– .gitignore: перелік директорій, які мають бути виключені з git;
– README.md: текстовий файл з описом проєкту та інструкцією з інсталяції та запуску;
– d3.html: HTML-файл, що ініціалізує графік розсіювання на D3.js;
– data: JSON-файли з наборами тестових даних;
– helpers: допоміжні функції модулів;
– js: містить окремі модулі для рендерингу графіків з використанням D3.js (d3-module.js) і WebGL (webgl-module.js);
– node_modules: директорія зі встановленими залежностями проекту;
– package-lock.json: фіксує версії встановлених пакетів;
– package.json: конфігураційний файл проекту, описує залежності та скрипт запуску;
– style: стилі оформлення інтерфейсу (CSS);
– webgl.html: HTML-файл, що ініціалізує графік розсіювання на WebGL.

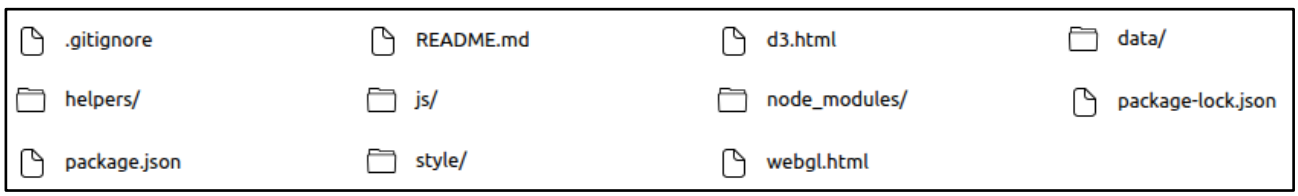

Рисунок 4.1 – Структура застосунку (рисунок виконаний самостійно)

Інтерфейс першого модуля візуалізації (D3.js) включає заголовок, панель керування сценаріями та область побудови графіка. Доступні три варіанти запуску: візуалізація 1000 точок, 10000 точок, динамічне додавання 10000 точок з

інтервалом у 5 секунд. Натискання відповідної кнопки ініціює побудову графіку розсіювання з використанням заданого обсягу даних. На зображенні (див. рис. 4.2) представлено результат запуску першого сценарію, який візуалізує 1000 точок у координатному просторі з діапазоном значень по осях від 0 до 1. Кнопка Reset дозволяє скинути графік та повторити експеримент з обраними параметрами.

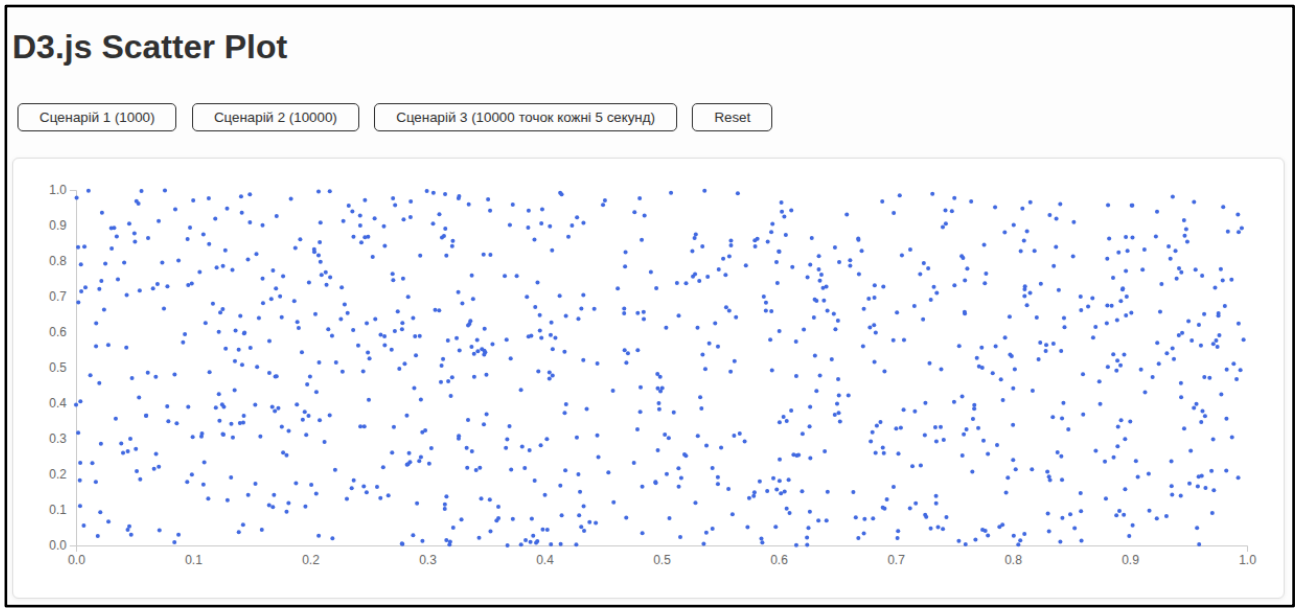

Рисунок 4.2 – Результат запуску першого сценарію на D3.js
(рисунок виконаний самостійно)

Загальний алгоритм побудови графіків за допомогою бібліотеки D3.js (ініціалізація SVG полотна → побудова шкал → створення осей → рендеринг примітивів) описано в п. 3.1. Основна відмінність між графіками полягає у фінальному етапі, тобто у виборі примітивів для відображення даних. У випадку побудови графіка розсіювання фінальний етап реалізується через прив’язку масиву точок до елементів <circle> у просторі SVG.

```
svg
        .selectAll("circle")
        .data(data)
        .join("circle")
        .attr("cx", d => xScale(d.x))
        .attr("cy", d => yScale(d.y))
        .attr("r", 2)
         .style("fill", "#4169e1");
```

Для побудови графіку було використано вбудований метод бібліотеки join(“circle”), що створює кола для кожного об’єкту у масиві data. Координати центрів (cx, cy) обчислюються за допомогою масштабувальних функцій xScale і yScale, які проектують значення у межах 0;1 на відповідні координати візуального контейнера. Радіус точок (r = 2) та колір (#4169e1) фіксовані.

Другий графічний модуль (WebGL) забезпечує доступ до аналогічного інтерфейсу візуалізації. Структура екрана повністю повторює функціональність модуля на D3.js, а основною відмінністю є використання WebGL як рушія рендерингу. На рисунку (див. рис. 4.3) представлено результат виконання другого сценарію - побудови 10000 точок у нормалізованому координатному просторі.

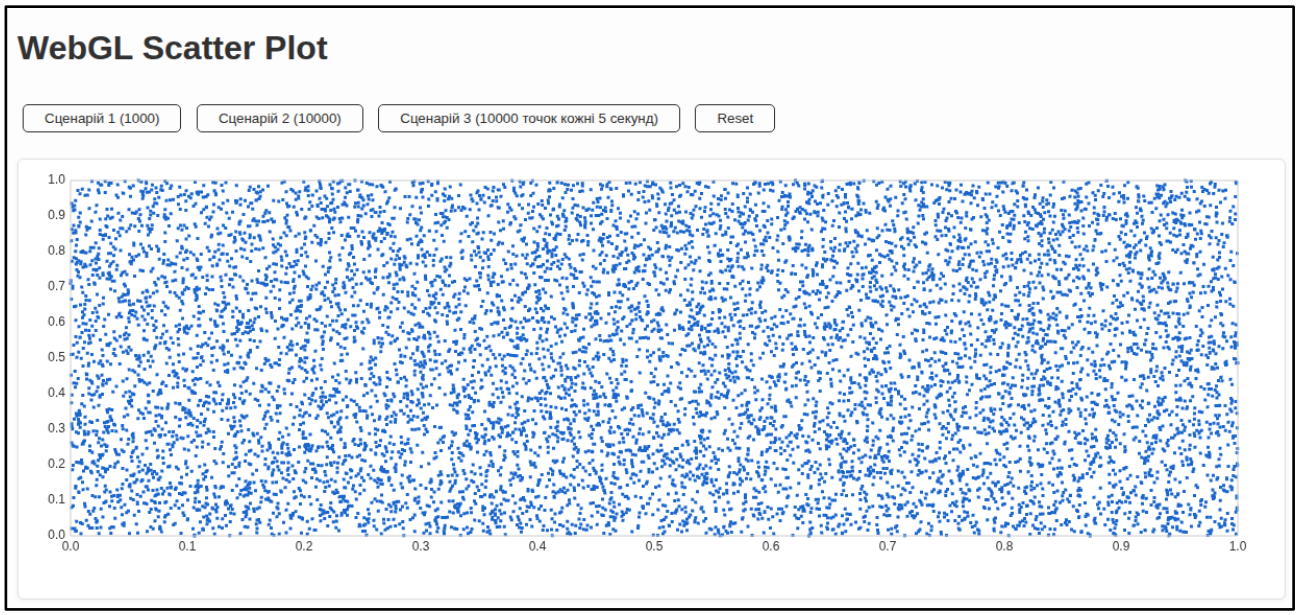

Рисунок 4.3 – Результат запуску сценарію №2 на WebGL
(рисунок виконаний самостійно)

Загальний принцип побудови графіків за допомогою технології WebGL (ініціалізація canvas → нормалізація координат → компіляція шейдерів → рендеринг примітивів) розглянуто у п. 3.2. Як і у випадку з D3.js, фінальний етап побудови графіка розсіювання на WebGL полягає у візуалізації примітивів - координатних точок, попередньо підготовлених у процесі нормалізації. Побудова виконується у два етапи: обробка даних та їх рендеринг засобами WebGL.

На першому етапі відбувається обчислення координат кожної точки вхідного масиву data. Зчитуються нормалізовані значення *x* та *y* діапазонах від 0 до 1 та перетворюються на піксельні координати з урахуванням внутрішніх відступів (MARGIN). Далі координати трансформуються у нормалізований простір пристрою (NDC), необхідний для рендерингу у WebGL. Результати додаються до масиву preparedVertices, у якому зберігаються пари значень для кожної точки.

```
function prepareVertices() {
  preparedVertices = [];

  for (let i = 0; i < data.length; i++) {
    const normX = data[i].x;
    const normY = data[i].y;

    const px = MARGIN.left + normX * (CHART_WIDTH - MARGIN.left -
MARGIN.right);
    const py = MARGIN.top + (1 - normY) * (CHART_HEIGHT - MARGIN.top
- MARGIN.bottom);

    const ndcX = (px / CHART_WIDTH) * 2 - 1;
    const ndcY = (1 - py / CHART_HEIGHT) * 2 - 1;

    preparedVertices.push(ndcX, ndcY);
  }
}
```

Візуалізація здійснюється у функції renderPoints(). Встановлюється область виводу (viewport) відповідно до розмірів canvas, після чого буфер очищується та заповнюється фоновим кольором.

```
gl.viewport(0, 0, CHART_WIDTH, CHART_HEIGHT);
gl.clearColor(1, 1, 1, 1);
gl.clear(gl.COLOR_BUFFER_BIT);
```

Створюється новий буфер, у який передаються підготовлені координати точок. Буфер пов’язується з простором ARRAY_BUFFER, а дані передаються у вигляді типізованого масиву Float32Array.

```
const vertexBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
```

```
    gl.bufferData(gl.ARRAY_BUFFER,   new   Float32Array(preparedVertices),
gl.STATIC_DRAW);
```

Оголошуються джерела коду вершинного та фрагментного шейдерів. Вершинний шейдер приймає координати (a_position), задає розмір точки та визначає її положення у просторі. Фрагментний шейдер задає єдиний колір усіх точок.

```
const vsSource = `
  attribute vec2 a_position;
  void main() {
    gl_PointSize = ${POINT_SIZE}.0;
    gl_Position = vec4(a_position, 0, 1);
  }
`;

const fsSource = `
  void main() {
    gl_FragColor = vec4(0.1, 0.4, 0.8, 1);
  }
`;
```

Шейдери компілюються, вихідні рядки коду перетворюються на об’єкти, які розуміє графічний процесор.

```
const vs = gl.createShader(gl.VERTEX_SHADER);
gl.shaderSource(vs, vsSource);
gl.compileShader(vs);

const fs = gl.createShader(gl.FRAGMENT_SHADER);
gl.shaderSource(fs, fsSource);
gl.compileShader(fs);
```

Обидва шейдери об’єднуються в єдину програму рендерингу, яка лінкується та активується для подальшого використання.

```
const program = gl.createProgram();
gl.attachShader(program, vs);
gl.attachShader(program, fs);
gl.linkProgram(program);
gl.useProgram(program);
```

Налаштовується передача координат у шейдер. Вказується атрибут a_position, який зчитує по дві FLOAT-координати на кожну точку з буфера.

```
gl.drawArrays(gl.POINTS, 0, preparedVertices.length / 2);
```

Останній виклик ініціює безпосередній рендеринг точок. Функція gl.drawArrays виводить усі точки, збережені у буфері, у вигляді примітивів типу POINTS.

```
gl.drawArrays(gl.POINTS, 0, preparedVertices.length / 2);
```

Отже, обидва графічні модулі реалізовано з дотриманням максимально можливої функціональної симетрії: використовуються однакові сценарії навантаження, структура даних, логіка запуску та організація інтерфейсу. Такий підхід дозволяє усунути сторонні фактори, що можуть вплинути на результати, і забезпечити об’єктивне порівняння продуктивності D3.js та WebGL.

4.2 Умови експериментального середовища

Експерименти з вимірювання продуктивності проводились на ноутбуці з наступними характеристиками: операційна система - Ubuntu 18.04.6 LTS (64-bit), процесор – Intel® Core™ i7-10750H CPU @ 2.60GHz (12 потоків), обсяг оперативної пам’яті – 31,2 ГБ, графічний адаптер – NVIDIA GeForce GTX 1650 Ti (PCIe/SSE2), графічне середовище - GNOME 3.28.2.

Вимірювання продуктивності здійснювались за допомогою Chrome DevTools - вбудованого інструменту для моніторингу та аналізу роботи веб-сторінок у браузері Google Chrome. У процесі аналізу використовувалися вкладки Performance та Rendering, що надають можливість відстежувати час побудови графічних елементів, частоту кадрів та частку навантаження на GPU.

Тестування виконувалось у контрольованих умовах на стабільній версії браузера Chrome 127.0.6533.119. Перед запуском кожного сценарію всі фонові вкладки було закрито, розширення вимкнено, вікно браузера відкрите у

звичайному режимі (не інкогніто). Для кожного окремого варіанту навантаження тест повторювався три рази із фіксацією середнього значення ключових метрик.

Для моделювання навантаження у межах кожного сценарію використовувались програмно згенеровані дані. Генерація здійснювалась за допомогою функції generateNormalizedData(count), яка створює масив об’єктів фіксованої довжини. Кожен об’єкт містить два числові поля: x та y, значення яких задаються випадковим чином у межах інтервалу 0;1.

```
export function generateNormalizedData(count) {
  return Array.from({ length: count }, () => ({
    x: Math.random(),
    y: Math.random()
  }));
}
```

У результаті було сформовано набори даних такого типу:

```
[
  {
    "x": 0.810170914434581,
    "y": 0.7208754577109755
  },
  {
    "x": 0.3163136425823916,
    "y": 0.8153370883287558
  },
  {
    "x": 0.9389197156366065,
    "y": 0.8286681122979417
  },
  ...
]
```

Структура даних є універсальною для обох реалізованих графічних модулів та дозволяє забезпечити повну симетрію умов експерименту. Використання нормалізованих значень забезпечує простоту трансформації координат як для SVG-графіки (D3.js), так і для графічного простору WebGL.

4.3 Тестування метрик продуктивності

З метою формалізації критерію “Особливості апаратного рендерингу” та порівняння частки навантаження на GPU було проведено серію замірів за допомогою інструменту Performance у Chrome DevTools (див. рис. 4.4).

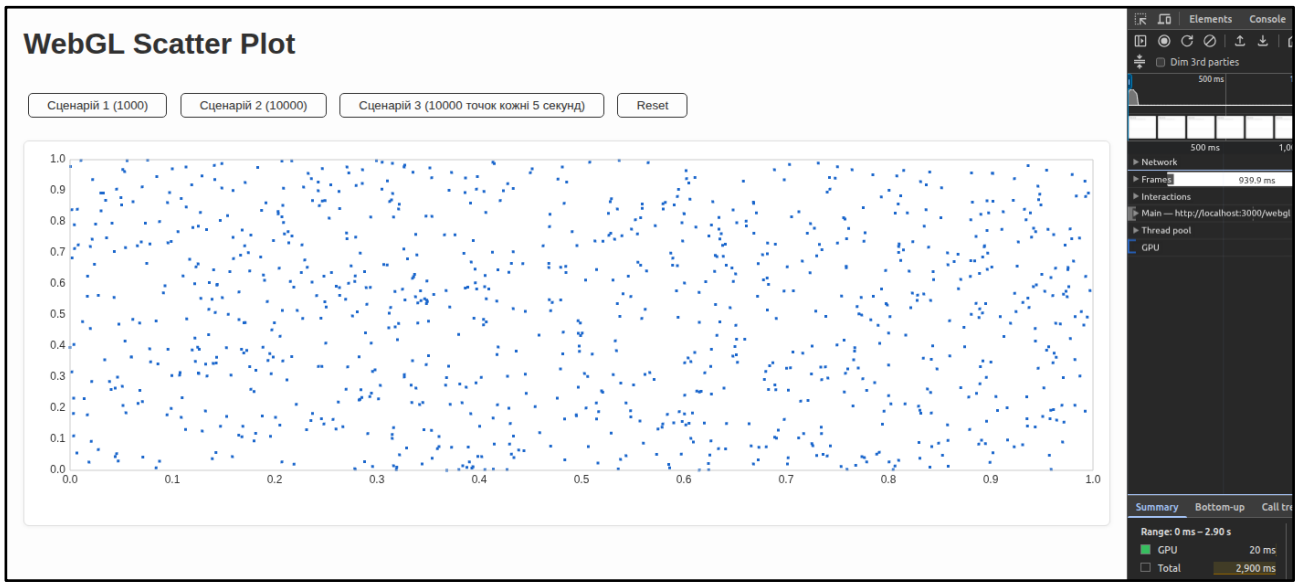

Рисунок 4.4 – Приклад запису метрики навантаження GPU для першого сценарію графічного модулю на WebGL (рисунок виконаний самостійно)

Під час кожного запуску фіксувалися значення GPU time (кількість мілісекунд, протягом яких графічний процесор був активний) та Total time (загальний час виконання профілю). На основі цих даних було розраховано частку GPU-навантаження як відношення GPU time до загального часу (формула 4.1):

$$GPU_{share} = \frac{GPU_{time}}{Total_{time}} \tag{4.1}$$

При тестуванні графічного модуля на D3.js у всіх вимірюваннях GPU time дорівнював нулю, що підтверджує припущення про те, що бібліотека не використовує графічний процесор при побудові SVG-графіки.

При тестуванні графічного модуля на WebGL GPU time варіювався від 19 до 92 мс, а загальний час виконання – від 860 до 12193 мс. На основі цих даних було

обчислено частку GPU-навантаження, яка коливається у межах від 0.70% до 3.46%, із середнім значенням 1.61%.

Аналіз показників свідчить (див. табл. 4.1), що WebGL дійсно активує апаратне прискорення, однак навіть у цьому випадку частка GPU-навантаження залишається відносно невисокою, що свідчить про ефективність обробки та потенціал до масштабування візуалізацій без критичного навантаження на систему.

Таблиця 4.1 – Метрики частки навантаження на GPU (таблиця виконана самостійно)

	Мін. D3.js	Середнє D3.js	Макс. D3.js	Мін. WebGL	Середнє WebGL	Макс. WebGL
GPU time (мс)	0	0	0	19.0	-	92.0
Total time (мс)	-	-	-	860.0	-	12193.0
GPU share (%)	0	0	0	0.7	1.61	3.46

З метою формалізації критерію “Продуктивність” було виміряно час рендерингу (Render Time), що характеризує тривалість процесу побудови графіку після ініціалізації сценарію. Вимірювання здійснювалось у вкладці Performance інструменту Chrome DevTools, де можна зафіксувати початок та кінець візуалізації.

Для проведення вимірювання (див. рис. 4.5) необхідно натиснути кнопку запису (Record), після чого запустити сценарій побудови графіку.

По завершенні запису отримуємо деталізовані метрики: Painting – час, витрачений на візуалізацію пікселів; Scripting – час виконання JavaScript-коду; Rendering – час компоновки стилів, макету та підготовки до рендерингу; Other - інші витрати, не віднесені до основних категорій.

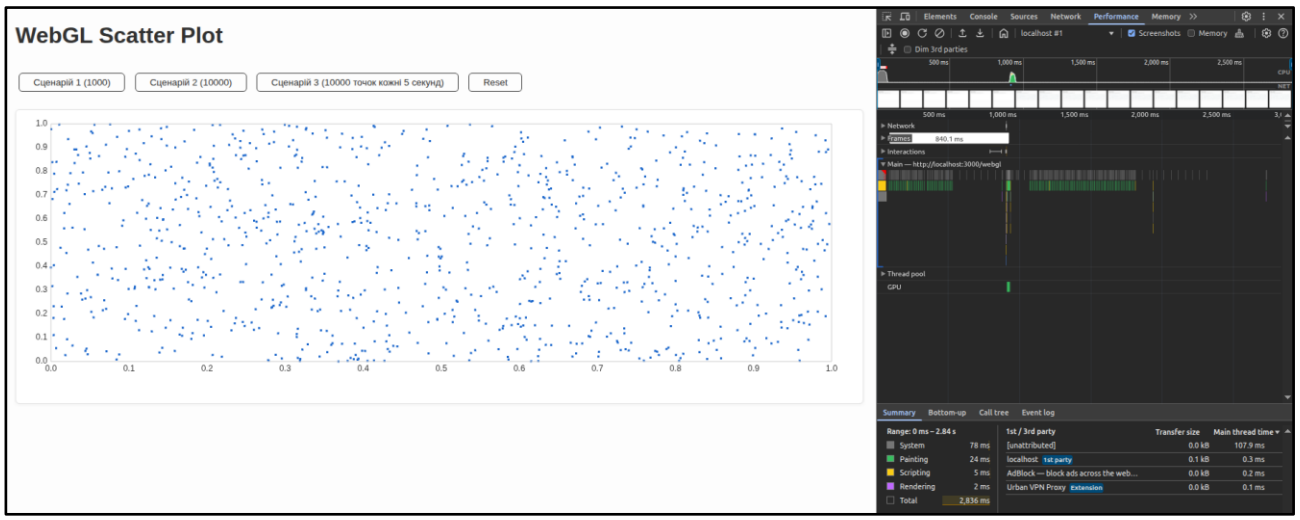

Рисунок 4.5 – Приклад запису метрики Render Time для першого сценарію графічного модулю на WebGL (рисунок виконаний самостійно)

Варто зазначити, що загальний час не є еквівалентом реального часу рендерингу. Це сумарна тривалість всіх подій, які були зафіксовані під час запису, включно з паузами, неактивними періодами та фоновими процесами. Тобто дана метрика охоплює повний інтервал профілювання, а не лише час активної побудови графіки. Саме тому для аналізу продуктивності доцільно орієнтуватись на окремі метрики - Painting, Scripting, Rendering – а не на загальну тривалість запису.

Друга метрика даного критерію – частота кадрів в секунду (FPS), є показником плавності візуалізації та загальної продуктивності графічного модуля під час рендерингу. Вимірювання здійснювалось за допомогою вкладки Rendering у Chrome DevTools, яка містить вбудований індикатор FPS, що відображає кількість кадрів які браузер здатен відтворити за секунду в реальному часі.

Для активації FPS-індикатора необхідно увімкнути опцію Frame Rendering Stats у вкладці Rendering Chrome DevTools (див. рис. 4.6), після чого запустити виконання обраного сценарію побудови графіку. Протягом усього процесу у верхній частині вікна браузера відображається графік коливань FPS, а також їх середнього значення. На відміну від часу рендерингу, частота кадрів є динамічною характеристикою, яка може змінюватися залежно від навантаження на систему, обсягу даних та конкретного етапу візуалізації.

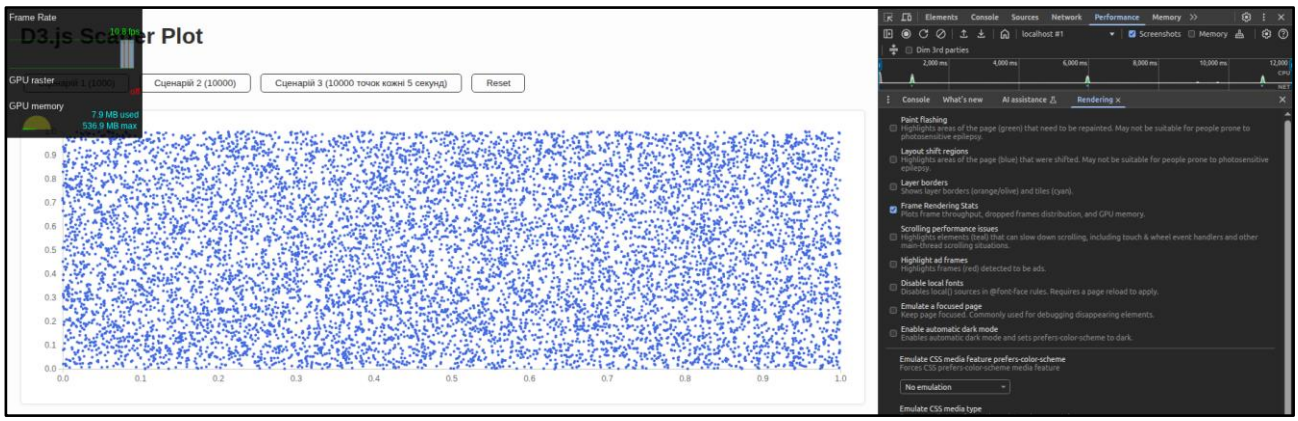

Рисунок 4.6 – Запис метрики FPS для другого сценарію графічного модулю на D3.js (рисунок виконаний самостійно)

Слід зауважити, що окреме зафіксоване значення FPS (наприклад, 10.8 кадрів/сек як на прикладі) є лише миттєвим знімком і не дає уявлення про загальну продуктивність - воно може відповідати проміжному стану, такому як скролінг, очікування чи завершення рендерингу. Для достовірної оцінки навантаження доцільно орієнтуватися на аналіз повної хронології подій у вкладці Performance.

Результати (вид. табл. 4.2) демонструють чітку відмінність у продуктивності між D3.js та WebGL. У першому сценарії (статична діаграма з 1000 точок) обидві технології показали подібні значення: час побудови (RT) для D3.js становив 165 мс, для WebGL - 149 мс; при цьому FPS залишався на високому рівні - понад 119 кадрів за секунду для обох варіантів. Це свідчить про те, що при невеликому обсязі даних обидві технології забезпечують достатню швидкодію.

Таблиця 4.2 – Метрики продуктивності (таблиця виконана самостійно)

Технологія	**RT**	**FPS**
Сценарій 1 (статична діаграма, 1000 точок)		
D3.js	165	120.2
WebGL	149	119

Кінець таблиці 4.2

Технологія	**RT**	**FPS**
Сценарій 2 (статична діаграма, 10 000 точок)		
D3.js	271	115.4
WebGL	111	120.2
Сценарій 3 (динамічна діаграма, + 10 000 точок кожні 5 секунд)		
D3.js	919	80.1
WebGL	179	119.3

У другому сценарії, де кількість точок зросла до 10 000, WebGL зберіг стабільну продуктивність із RT на рівні 111 мс і FPS 120.2, тоді як у D3.js час побудови зріс до 271 мс, а FPS дещо знизився до 115.4. Це вказує на кращу масштабованість WebGL при обробці великих обсягів статичних даних.

Найбільш виразна різниця простежується в третьому сценарії з динамічним зростанням кількості точок (по 10 000 кожні 5 секунд). Для D3.js час відгуку досяг 919 мс, а FPS впав до 80.1, що свідчить про істотне навантаження на систему та погіршення взаємодії з користувачем. У той же час WebGL зберіг стабільні показники: RT – 179 мс, FPS – 119.3, що підтверджує ефективність апаратного рендерингу навіть за умов постійного оновлення даних.

4.4 Тестування якісних метрик

З метою формалізації критерію “Підтримка спільноти” було зібрано актуальні статистичні дані з відкритих репозиторіїв технологій D3.js [46] та WebGL [47], розміщених на платформі GitHub. Такий підхід було обрано з огляду на переваги використання GitHub як джерела даних з відкритою статистикою. Це найбільша платформа відкритого коду з публічними, автоматизованими метриками, що забезпечує об’єктивність аналізу. Більшість сучасних JavaScript-бібліотек розміщено саме на GitHub, тому активність у репозиторіях достовірно

відображає інтерес спільноти. Метрики на кшталт кількості “зірочок”, “форків” і підписників дозволяють побудувати формалізовану шкалу для порівняння технологій. На відміну від статичних публікацій, ці дані постійно оновлюються та є масштабованими, що дає змогу створювати узагальнені рейтинги або великі вибірки для подальшого аналізу.

Результати (див. табл. 4.3) показують, що показники D3.js перевищують відповідні показники WebGL приблизно у 30 разів за кількістю форків, у 34 рази за кількістю підписників та у 40 разів за кількістю зірок, що свідчить про значно вищу популярність і поширеність бібліотеки серед розробників.

Таблиця 4.3 – Метрики GitHub-репозиторіїв (таблиця виконана самостійно)

Технологія	Fork	Watch	Star
D3.js	3609	22900	109000
WebGL	117	670	2700

З метою формалізації критерію “Розширюваність” було проведено порівняльний аналіз підходів до реалізації користувацьких інтерфейсів із використанням бібліотеки D3.js (конструкторський) та низькорівневого API WebGL.

У сучасному програмному забезпеченні розрізняють кілька архітектурних підходів до побудови візуалізацій: конфігураційний, конструкторський та низькорівневий. Конфігураційний підхід передбачає використання готових компонентів із фіксованими параметрами, що практично унеможливлює розширення функціоналу. Таким интерфейсам відповідає значення розширюваності 0.

D3.js реалізує конструкторський підхід: розробник самостійно формує необхідну структуру з базових візуальних елементів, що забезпечує гнучкість і дозволяє адаптувати графік під конкретні потреби. Водночас така побудова передбачає використання наявних механізмів бібліотеки. Таким чином, незважаючи на більшу свободу порівняно з конфігураційними рішеннями, D3.js

не дає повного низькорівневого контролю. З огляду на це, його розширюваність формально оцінено (див. табл. 4.4) як обмежену – з мінімальним ненульовим значенням 0.1 у порівняльній шкалі. WebGL, натомість, як низькорівневе API, забезпечує повний доступ до механізмів рендерингу та дозволяє реалізовувати довільні графічні рішення, що відповідає максимальному рівню розширюваності з оцінкою 1.

Таблиця 4.4 – Метрики критерію Розширюваність (таблиця виконана самостійно)

	D3.js	WebGL
Розширюваність	0.1	1

З метою формалізації критерію “Обмеження” було використано дані з ресурсу Can I Use [48], який агрегує й регулярно оновлює інформацію про підтримку окремих веб-функцій у сучасних браузерах. Завдяки використанню численних незалежних джерел і оцінці на рівні конкретних можливостей, ресурс дозволяє здійснювати точний і достовірний аналіз сумісності технологій.

Оскільки D3.js не є самостійною веб-технологією, оцінка її підтримки в браузерах повинна базуватися на аналізі підтримки SVG як її фундаментальної технології. SVG, за винятком застарілих версій Internet Explorer (6-8) та Android Browser (версій 2.1-2.3), підтримується всіма (див. рис. 4.7) сучасними браузерами (96.99%). Враховуючи, що Internet Explorer є фактично виведеним з експлуатації (більше не підтримується офіційно з 15 червня 2022 року [49]), а версії 2.1-2.3 Android Browse були актуальні у 2009-2010 роках [48], можна вважати, що підтримка SVG у сучасних браузерах є повною (100%).

Згідно з даними ресурсу Can I Use (див. рис. 4.8), підтримка WebGL у сучасних браузерах становить 96.89%.

Однак під час аналізу виявлено додаткові ризики, зокрема апаратну залежність: WebGL потребує GPU-прискорення, що створює залежність від

обладнання та драйверів. У деяких випадках WebGL може бути вимкнено через несумісність, відсутність доступу до GPU або вручну користувачем.

Рисунок 4.7 – Підтримка SVG у браузерах [48]

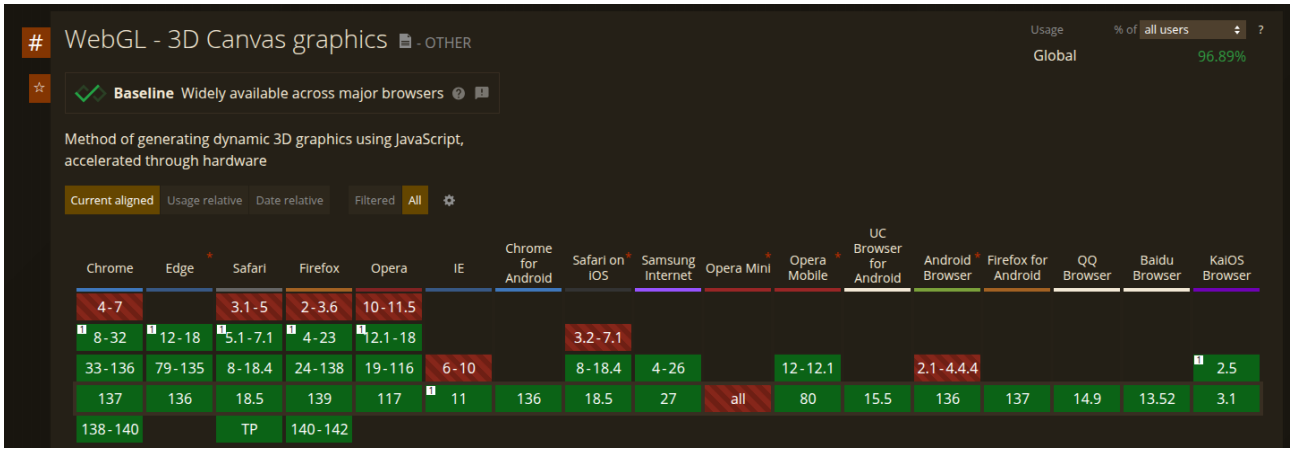

Рисунок 4.8 – Підтримка WebGL у браузерах [48]

Оцінити точну частку таких випадків на глобальному рівні технічно неможливо, оскільки подібна інформація не відображається у відкритій статистиці. У цьому дослідженні було свідомо закладено приблизний ризик на рівні 5%.

Другий ризик, пов’язаний із використанням WebGL, помітний у середовищах з підвищеними вимогами до інформаційної безпеки, де цю технологію можуть блокувати через потенційні вразливості драйверів, ризик зловживань графічними ресурсами та можливість ідентифікації пристроїв через browser fingerprinting [39]. Згідно з даними Hostmaster.ua [50], станом на червень

2025 року в доменній зоні .gov.ua зареєстровано 5215 доменів із загальних 455670, що становить приблизно 1.14%.

Окрім урядових структур, обмеження на використання WebGL можуть також діяти у фінансових установах, транспортних компаніях та інших критичних галузях із жорсткими вимогами до захисту інформації. З урахуванням частки доменів .gov.ua та ймовірної наявності подібних обмежень у суміжних сферах, частку середовищ із підвищеними вимогами до інформаційної безпеки у загальному веб-просторі можна орієнтовно оцінити на рівні 2%.

Другою метрикою критерію “Обмеження” є доступність, що оцінює здатність технології забезпечити коректне відображення й взаємодію з вмістом для людей з вадами зору та різними когнітивними потребами.

SVG є частиною DOM і підтримує атрибути доступності (aria, title тощо), завдяки чому візуалізації на його основі можуть бути повністю сумісними з екранними читачами. Натомість WebGL працює через <canvas>, який не взаємодіє з DOM у контексті доступності. На сьогодні відсутні стандартні рішення для забезпечення повноцінної доступності WebGL-графіки.

Згідно з отриманими даними (див. табл. 4.5), D3.js забезпечує повну підтримку в браузерах та є повністю доступним, тоді як WebGL, хоча й підтримується більшістю браузерів, не відповідає вимогам доступності.

Таблиця 4.5 – Метрики Підтримка в браузерах та Доступність (таблиця виконана самостійно)

Технологія	Підтримка в браузерах	Доступність
D3.js	100%	100%
WebGL	89.89%	0%

5 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ

5.1 Структуризація проблеми

Завдання даного дослідження класифікується як вибір, оскільки основною його метою є визначення оптимальної технології для візуалізації великих обсягів даних у веб-застосунках із різними сценаріями використання. Такий підхід передбачає порівняння альтернатив за низкою релевантних критеріїв з урахуванням потенційних вимог до продуктивності, масштабованості тощо. Ієрархічна структура процесу прийняття рішення (див. рис. 5.1) охоплює три рівня: загальну ціль (завдання), критерії оцінювання та відібрані для аналізу альтернативи – технології WebGL і D3.js.

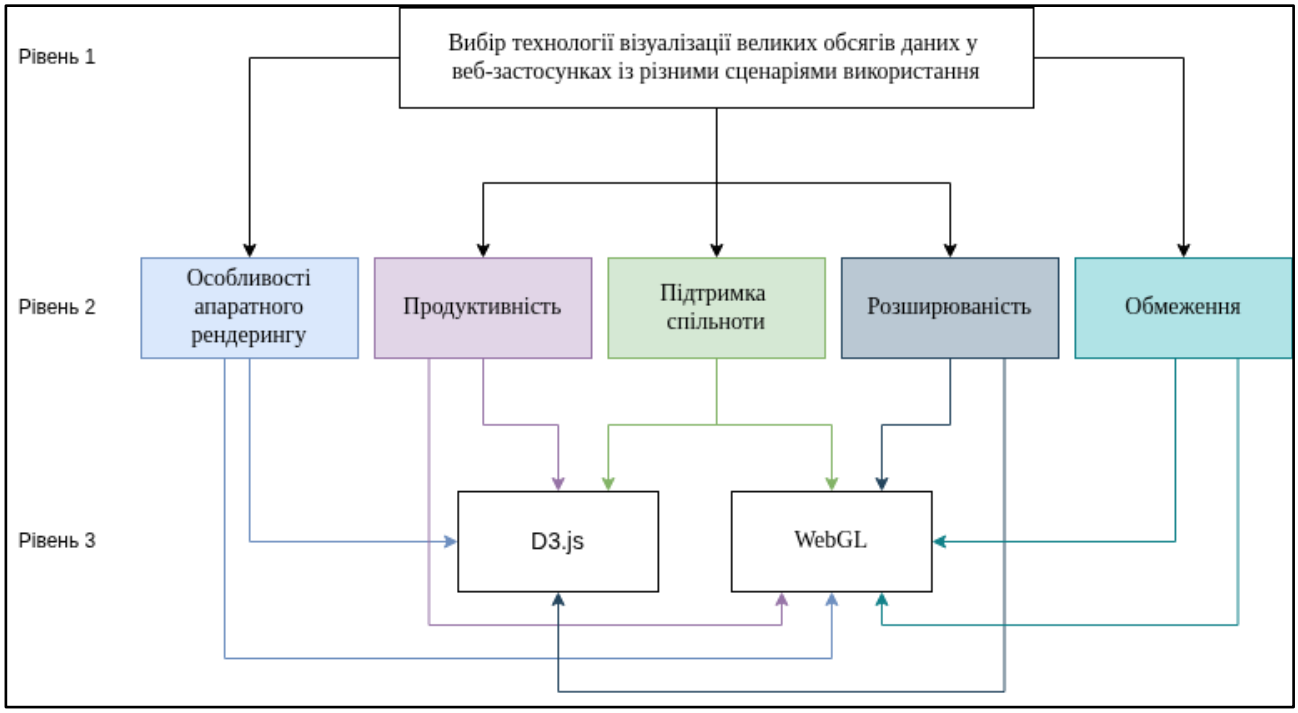

Рисунок 5.1 – Ієрархічна структура (рисунок виконаний самостійно)

На початковому етапі дослідження для побудови матриці попарних порівнянь було застосовано припущення про рівнозначність всіх критеріїв. Такий підхід зумовлений відсутністю конкретного сценарного контексту, в межах якого могли б бути визначені пріоритети окремих параметрів. Без чітко окреслених умов використання (наприклад, обмежень продуктивності, цільової платформи тощо), жоден з критеріїв не може бути об’єктивно виділений як важливіший або

другорядний. Матриця (див. табл. 5.1) містить порівняння за всіма критеріями: особливості апаратного рендерингу (кр. №1), продуктивність (кр. №2), підтримка спільноти (кр. №3), розширюваність (кр. №4), обмеження (кр. №5).

Таблиця 5.1 – Матриця парних порівнянь (таблиця виконана самостійно)

	Кр. №1	Кр. №2	Кр. №3	Кр. №4	Кр. №5
Кр. №1	1	1	1	1	1
Кр. №2	1	1	1	1	1
Кр. №3	1	1	1	1	1
Кр. №4	1	1	1	1	1
Кр. №5	1	1	1	1	1

5.2 Опис критеріїв

Розглянемо перший критерій “Особливості апаратного рендерингу”. У процесі візуалізації браузер використовує як центральний процесор (CPU), так і графічний процесор (GPU), але їхні функції відрізняються. CPU відповідає за основні обчислення, обробку структури DOM і рендеринг векторної графіки. Натомість GPU прискорює окремі завдання, такі як обробка трансформацій (transform), прозорості (opacity) та фільтрів (filter).

У випадку SVG-графіки, яка рендериться через DOM, основне навантаження лягає саме на CPU. Він виконує всі геометричні обчислення, опрацьовує векторні шляхи та відображає елементи. GPU може бути задіяний лише на фінальному етапі – під час виводу зображення, якщо браузер використовує апаратне прискорення стилів або візуальних ефектів.

Згідно з результатами тестування (див. табл. 4.1), WebGL демонструє активне залучення графічного процесора, тоді як при використанні SVG (D3.js) GPU взагалі не задіюється.

Оцінка за критерієм “Особливості апаратного рендерингу” є важливою, оскільки ефективний розподіл навантаження між CPU і GPU визначає

стабільність та передбачуваність роботи візуалізації за різних умов. Розуміння того, яка частина обчислювальної архітектури бере на себе основне навантаження, дозволяє врахувати технічні характеристики пристрою та забезпечити оптимальну продуктивність за мінімального використання ресурсів.

Критерій “Продуктивність” оцінює здатність технології забезпечувати швидкий рендеринг, стабільну частоту кадрів і оцінити затримки при відображенні графічного контенту. Це особливо важливо для інтерактивних візуалізацій, де користувач очікує миттєвої реакції системи на свої дії. Згідно з результатами тестування (див. табл. 4.2), WebGL демонструє стабільно високу продуктивність у всіх сценаріях. Такий результат пояснюється використанням апаратного прискорення через GPU, що дає можливість ефективно обробляти великі обсяги даних, складні геометричні структури та анімовані сцени з мінімальним навантаженням на центральний процесор (CPU).

На відміну від рендерингу через DOM, де кожен графічний елемент створюється і обробляється як окремий об’єкт на рівні браузера, WebGL працює безпосередньо з буферами пам’яті, що дозволяє уникати надлишкових обчислень і оптимізує роботу з великими масивами. Завдяки цьому WebGL є особливо ефективним у ситуаціях, коли візуалізація виконується в реальному часі, включає динамічні оновлення або потребує плавної анімації великої кількості елементів.

Оцінка технології за цим критерієм має прикладне значення, оскільки низька продуктивність напряму впливає на зручність використання, обмежує складність візуалізації та може призводити до втрати точності сприйняття даних. Затримки рендерингу або “підвисання” анімацій негативно позначаються на якості взаємодії користувача з візуалізацією і сприйняттю її змісту.

Розглянемо третій критерій - “Підтримка спільноти”. Під спільнотою розуміється група людей, які активно створюють, використовують, підтримують і розвивають певну технологію. До неї входять розробники, контриб'ютори, користувачі, а також ті, хто створює навчальні матеріали, розробляє плагіни чи надає технічну підтримку. Спільнота функціонує як екосистема, що сприяє обміну знаннями, вирішенню проблем і постійному вдосконаленню інструменту.

Згідно з результатами аналізу репозиторіїв на GitHub (див. табл. 4.3), технологія D3.js має значно вищу підтримку спільноти порівняно з WebGL: кількість зірок перевищує 100 тисяч, а кількість підписників і форків також у десятки разів більша, що свідчить про активну розробку, широку базу користувачів і високий рівень довіри до інструменту. WebGL, хоча і є потужною технологією, має значно менше охоплення, що може ускладнити пошук рішень та прикладів для нових користувачів.

Оцінка технології за критерієм “Підтримка спільноти” є важливою, оскільки активна спільнота забезпечує підтримку, полегшує обмін досвідом і стимулює розвиток інструменту. Широка мережа розробників, користувачів і контрибуторів спрощує вирішення проблем, пришвидшує пошук рішень і сприяє безперервному вдосконаленню технології.

Розглянемо наступний критерій - “Розширюваність”. Оскільки вимоги до проєкту з часом можуть змінюватись (зростання обсягів даних, ускладнення функціоналу або додавання інтерактивних компонентів) здатність технології до розширення дозволяє адаптуватися до змін без значних витрат часу й ресурсів, забезпечити довговічність рішення та мінімізувати ризики технічних обмежень.

D3.js використовує конструкторський підхід, за якого інтерфейс створюється з окремих елементів, а не задається параметризованим об’єктом. Такий підхід надає досить високий рівень гнучкості, оскільки розробник має повний контроль над структурою та логікою візуалізації. WebGL, як низькорівневе API, фактично не має обмежень у розширенні – єдиним чинником є рівень знань і творче бачення розробника.

Згідно з результатами (див. табл. 4.4), обидві технології мають високий потенціал для розширення, але підходи суттєво відрізняються. D3.js забезпечує певну гнучкість через конструкторський підхід, що зручно для швидкої адаптації та налаштувань. WebGL, натомість, відкриває майже необмежені можливості завдяки низькорівневому API, однак вимагає глибших технічних знань від розробника.

Оцінка технології за критерієм “Розширюваність” є значущою, адже вона визначає, наскільки легко рішення може адаптуватися до змін у вимогах проєкту. Обмежена розширюваність може спричинити значні витрати на зміну підходу або навіть повний перехід на інший інструмент. Гнучка технологія, навпаки, дозволяє масштабуватися, додавати інтерактивність і працювати з більшими обсягами даних без необхідності перебудови всієї архітектури.

Розглянемо останній критерій - “Обмеження”. Він об'єднує такі показники, як підтримка в браузерах і доступность, які визначають наскільки широке коло користувачів може коректно взаємодіяти з інтерфейсом. Висока підтримка в браузерах гарантує стабільну роботу застосунку на більшості пристроїв, тоді як доступність безпосередньо впливає на інклюзивність - можливість використання ресурсу людьми з вадами зору та різними когнітивними потребами. Дані показники дозволяють оцінити технологію не лише з технічної, а й із соціально-етичної точки зору, що особливо важливо в контексті державних, освітніх чи публічних проектів.

Згідно з результатами (див. табл. 4.5), D3.js повністю сумісна з усіма сучасними браузерами та має вбудовану підтримку доступності, що робить її надійним вибором для широкої аудиторії. WebGL показує високу, але не абсолютну підтримку в браузерах і не забезпечує доступності для користувачів з особливими потребами.

Оцінка технологій за критерієм “Обмеження” є надзвичайно важливою, оскільки технічні обмеження визначають, наскільки стабільним і доступним буде рішення в реальних умовах. Використання інструментів, які не враховують сумісність із браузерами або потреби різних груп користувачів, може призвести до непередбачуваних труднощів і потреби у додаткових ресурсах для усунення недоліків.

5.3 Агрегація результатів

Оцінювання технологій за різними критеріями потребує порівняння показників, що мають різні одиниці виміру та діапазони значень. Наприклад,

навантаження на GPU подається у вигляді відносної частки (від 0 до 1), тоді як популярність у спільноті вимірюється в тисячах зірок, форків і підписок на GitHub. Розрахунки виконувалися з використанням мін-макс нормалізації показників описаній у п. 2. У разі потреби результати коригувалися до їхнього середнього арифметичного значення. Нормалізовані результати (див. табл. 5.2) є основою для наступного етапу аналізу – аналізу чутливості.

Таблиця 5.2 – Агрегація результатів (таблиця виконана самостійно)

Критерій	D3.js	WebGL
Особливості апаратного рендерингу	0.1	1
Продуктивність	0.63	0.97
Підтримка спільноти	1	0.1
Розширюваність	0.1	1
Обмеження	1	0.1

5.4 Аналіз чутливості

На першому етапі аналізу чутливості було застосовано лінійну адитивну згортку з використанням вагових коефіцієнтів за умови, що всі критерії мають однакову важливість (див. табл. 5.1). У цьому випадку кожному критерію було присвоєно вагу 0.2 (формула 5.1).

$$TS = H \times Wh + P \times Wp + C \times Wc + E \times We + L \times Wl \quad (5.1)$$

WebGL отримала вищу загальну оцінку (див. табл. 5.3) завдяки високій продуктивності та розширюваності. Натомість D3.js має нижчий результат, але вирізняється сильною підтримкою спільноти та високою доступністю.

Таблиця 5.3 – Результати згортки із рівними вагами (таблиця виконана самостійно)

Технологія	TS
D3.js	0.566
WebGL	0.634

Для проведення другого етапу аналізу чутливості було розглянуто два сценарії використання, для яких було створено матриці попарних порівнянь. Матриці попарних порівнянь для кожного сценарію були побудовані на основі змістовного аналізу важливості критеріїв у відповідному контексті з урахуванням їх відносної пріоритетності, визначеної через експертне оцінювання.

Елементи матриці попарних порівнянь a_{ij} визначалися як відношення важливості критерію i до критерію j у межах обраного сценарію, тобто (формула 5.2):

$$a_{ij} = \frac{w_i}{w_j} \tag{5.2}$$

де: w_i, та w_j – умовні оцінки відносної важливості критеріїв, отримані на основі змістовного аналізу.

Перший сценарій: розробка динамічного, високопродуктивного графіку для закритої аналітичної платформи, що передбачає візуалізацію великих масивів даних (100 000 точок). Час на опанування технології та створення MVP обмежено шістьма місяцями (див. табл. 5.4).

Таблиця 5.4 – Матриця парних порівнянь за першим сценарієм (таблиця виконана самостійно)

	Кр. №1	Кр. №2	Кр. №3	Кр. №4	Кр. №5
Кр. №1	1	0.802	2.008	1.335	3.985

Кінець таблиці 5.4

	Кр. №1	Кр. №2	Кр. №3	Кр. №4	Кр. №5
Кр. №2	1.247	1	2.504	1.665	4.97
Кр. №3	0.498	0.399	1	0.665	1.985
Кр. №4	0.749	0.601	1.504	1	2.985
Кр. №5	0.251	0.201	0.504	0.335	1

Другий сценарій: створення десяти статичних графіків (кругова діаграма, гістограма, діаграма розсіювання) з мінімальними інтерактивними елементами для відкритої освітньої платформи (див. табл. 5.5).

Таблиця 5.5 – Матриця парних порівнянь за другим сценарієм (таблиця виконана самостійно)

	Кр. №1	Кр. №2	Кр. №3	Кр. №4	Кр. №5
Кр. №1	1	0.251	0.335	0.504	0.201
Кр. №2	3.985	1	1.335	2.008	0.802
Кр. №3	2.985	0.749	1	1.504	0.601
Кр. №4	1.985	0.498	0.665	1	0.399
Кр. №5	4.97	1.247	1.665	2.504	1

Для кожного сценарію на основі матриці попарних порівнянь із застосуванням пропорційного методу було визначено вагові коефіцієнти критеріїв (див. табл. 5.6).

Таблиця 5.6 – Вагові коефіцієнти для сценаріїв використання (таблиця виконана самостійно)

Критерій	1 сценарій	2 сценарій
Особливості апаратного рендерингу	0.267	0.067

Кінець таблиці 5.6

Критерій	1 сценарій	2 сценарій
Продуктивність	0.333	0.267
Підтримка спільноти	0.133	0.200
Розширюваність	0.200	0.133
Обмеження	0.067	0.333

Вагові коефіцієнти надали можливість повторно застосувати метод адитивної згортки для кожного зі сценаріїв, що, у свою чергу, забезпечило формалізовану основу для порівняння технологій WebGL та D3.js з урахуванням контекстно залежних пріоритетів.

Аналіз отриманих результатів показує (див. табл. 5.7), що вибір технології для візуалізації у веб-застосунках має ґрунтуватися на конкретних вимогах проєкту. При рівномірному розподілі вагових коефіцієнтів WebGL дещо перевершила D3.js. Проте аналіз чутливості виявив, що кінцеві оцінки значною мірою залежать від пріоритетів проєкту.

Таблиця 5.7 – Результати лінійної адитивної агрегації з урахуванням визначених вагових коефіцієнтів для обох сценаріїв використання (таблиця виконана самостійно)

Технологія	1 сценарій	2 сценарій
D3.js	0.353	0.733
WebGL	0.813	0.433

У першому сценарії, де основними критеріями були продуктивність і ефективність рендерингу, WebGL отримала значно вищий підсумковий бал. Це пояснюється тим, що ця технологія оптимізована для роботи з великими обсягами даних і повністю використовує потужність графічного процесора. Хоча D3.js має зручний інтерфейс та потужні можливості для роботи з SVG, вона не може

конкурувати з WebGL за швидкістю візуалізації. Тому у випадках, коли проєкт потребує інтенсивної обробки даних і високої продуктивності, WebGL є кращим вибором.

Другий сценарій дав протилежний результат: коли пріоритетними стали доступність, підтримка спільноти та легкість інтеграції, перевага була за D3.js. Це зумовлено її великою популярністю серед розробників, наявністю великої кількості готових прикладів і гнучкістю в адаптації до різних завдань без значних вимог до графічного рендерингу.

На основі отриманих даних були сформульовані наступні рекомендації:

– використання WebGL доцільне у проектах, які потребують обробки великих обсягів даних у реальному часі, складних анімованих графічних елементів або високої частоти оновлення сцени. Технологія забезпечує ефективне використання ресурсів графічного процесора, що особливо актуально для інтерактивних дашбордів, систем моніторингу, візуалізацій часових рядів та симуляційних моделей;
– D3.js є кращим вибором для задач, де пріоритетом є гнучкість, доступність і швидкість впровадження. Завдяки широкій спільноті, розгалуженій документації та великій кількості прикладів, ця бібліотека ідеально підходить для створення адаптивних, кастомізованих візуалізацій з невисокими вимогами до продуктивності - наприклад, у звітності, аналітичних панелях або освітніх проектах;
– у гібридних сценаріях, де частина візуалізацій потребує високої продуктивності, а інша - простоти налаштування, доцільним є поєднання WebGL та D3.js. Наприклад, D3 може використовуватись для побудови вісей, контролів або текстових елементів, а WebGL - для основного графу або інтенсивної графічної сцени;
– під час планування архітектури проекту варто враховувати не лише технічні переваги технологій, але й рівень підготовки команди, наявні ресурси та довгострокову підтримку. D3.js має нижчий поріг входження, що дозволяє залучати ширше коло розробників, тоді як WebGL потребує

глибшого розуміння роботи з графікою, що зумовлює необхідність планування додаткового часу на навчання команди та освоєння інструментів;

– окрему увагу слід приділити пристроям користувачів: якщо цільова аудиторія використовує застарілі браузери або слабкі пристрої, можливості GPU можуть бути обмеженими - у таких випадках застосування WebGL слід додатково перевіряти на сумісність та стабільність.

ВИСНОВКИ

У межах кваліфікаційної роботи проведено дослідження, спрямоване на визначення оптимальної технології для візуалізації великих обсягів даних у веб-застосунках із різними сценаріями використання. Об’єктом дослідження є технології візуалізації великих обсягів даних у веб-середовищі, предметом - критерії та методи оцінки ефективності WebGL і D3.js у відповідному контексті.

Відповідно до поставленої мети виконано аналіз предметної області, охарактеризовано функціональні можливості обох технологій, виявлено ключові відмінності у підходах до рендерингу, інтеграції та продуктивності.

На основі методології MCDA побудовано узгоджену ієрархічну модель для формалізованого порівняння технологій. Здійснено всі етапи багатокритеріального аналізу: структуровано проблему, описано та оцінено критерії, виконано агрегування результатів та проведено аналіз чутливості. Вагові коефіцієнти розраховано за допомогою матриць попарних порівнянь та пропорційного методу. Емпіричні дані зібрано шляхом інструментального вимірювання продуктивності реалізованих рендеринг-модулів та фіксації релевантних метрик. Агрегацію виконано за допомогою лінійної адитивної згортки після нормалізації критеріїв за min–max.

Емпірична частина дослідження включала розробку двох рендеринг-модулів [51] з використанням WebGL та D3.js. Проведено інструментальне вимірювання таких показників продуктивності, як частота кадрів, час рендерингу, частка навантаження на GPU. На основі зібраних метрик сформовано таблиці для подальшого оцінювання ефективності.

Результати дослідження поглиблюють підходи до застосування багатокритеріального аналізу в задачах вибору технологій візуалізації у веб-застосунках. Запропонована модель дозволяє адаптувати критерії під конкретні потреби проекту.

За результатами аналізу сформульовано практичні рекомендації. D3.js доцільно використовувати у випадках, де важлива гнучкість, швидке розгортання

та доступність; WebGL – для задач із високими вимогами до продуктивності, великої кількості графічних об’єктів та складної анімації. У гібридних сценаріях рекомендовано комбіноване використання обох технологій.

Результати роботи апробовано на двох наукових конференціях (PIC S&T 2024 [52], eStream 2025 [53]), за матеріалами доповідей прийнято до публікації дві наукові статті. Отримані результати можуть бути використані в прикладних проектах як основа для вибору технології візуалізації у веб-застосунках.

Очікувана ефективність полягає в оптимізації вибору інструментів розробки, що дозволить скоротити витрати на підтримку та розгортання, підвищити адаптивність, стабільність і масштабованість інтерфейсів візуалізації у веб-застосунках.

У подальших дослідженнях доцільно зосередитися на зниженні порогу входження для використання WebGL у прикладних задачах. Перспективним напрямом є створення допоміжних бібліотек, що поєднують переваги низькорівневого рендерингу із зручністю декларативного підходу. Також доцільно дослідити питання інтеграції WebGL з високорівневими фреймворками в умовах обмежених ресурсів користувача.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Bostock M. Keynote – CSVConf 2017 / youtube. URL: https://www.youtube.com/watch?v=aT4JvF7sglg&ab_channel=csvconf (дата звернення: 01.05.2025).

2. Rusakova N., Ponikarovska N. and Shirokopetleva M., Research on the Efficiency of Applying WebGL and D3.js in the Creation of Graphical Elements in Web Applications // 2025 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream), Vilnius, Lithuania, 2025, pp. 1-5, doi: 10.1109/eStream66938.2025.11016891.

3. Comparative analysis of WebGL and D3.js. N.Rusakova, N.Ponikarovska, M.Shirokopetleva, The International Conference “Problems of Infocommunications. Science and Technology” (PIC S&T′2024), Kharkiv, Ukraine, 2025, in press.

4. Сотська Г., Шмельова Т. Словник мистецьких термінів. – Херсон: Видавництво “Стар”, 2016. – 52 с.

5. Великий тлумачний словник сучасної української мови (з доп. і допов.) / уклад. і гол. ред. В. Т. Бусел. – К.; Ірпінь: ВТФ “Перун”, 2005. – 1728 с.

6. Internet Archive. WayBack Machine / офіційний веб-сайт. URL: https://web.archive.org/ (дата звернення: 01.05.2025).

7. Gillies J., Cailliau R. How the Web was Born: The Story of the World Wide Web. – Oxford: Oxford University Press, 2000. – 340 p.

8. Chart.js / документація. URL: https://www.chartjs.org/ (дата звернення: 01.05.2025)

9. C3.js / документація. URL: https://c3js.org/ (дата звернення: 01.05.2025).

10. Billboard.js / документація. URL: https://naver.github.io/billboard.js/ (дата звернення: 01.05.2025).

11. Dygraphs / документація. URL: https://dygraphs.com/ (дата звернення: 01.05.2025).

12. Apache ECharts / документація. URL: https://echarts.apache.org/en/index.html (дата звернення: 01.05.2025).

13. Recharts / документація. URL: https://recharts.org/en-US (дата звернення: 01.05.2025).

14. Victory / документація. URL: https://nearform.com/open-source/victory/ (дата звернення: 01.05.2025).

15. vue-chartjs / документація. URL: https://vue-chartjs.org/ (дата звернення: 01.05.2025).

16. vue-echarts / документація. URL: https://vue-echarts.dev/ (дата звернення: 01.05.2025).

17. visx / документація. URL: https://airbnb.io/visx/ (дата звернення: 01.05.2025).

18. RawGraphs / документація. URL: https://www.rawgraphs.io/ (дата звернення: 01.05.2025).

19. D3.js / документація. URL: https://d3js.org/ (дата звернення: 01.05.2025).

20. Three.js / документація. URL: https://threejs.org/ (дата звернення: 01.05.2025).

21. regl / документація. URL: https://regl-project.github.io/ (дата звернення: 01.05.2025).

22. deck.gl / документація. URL: https://deck.gl/ (дата звернення: 01.05.2025).

23. plotly.js / документація. URL: https://plotly.com/javascript/ (дата звернення: 01.05.2025).

24. Ishizaka A., Nemery P. Multi-Criteria Decision Analysis: Methods and Software. – New York: Wiley, 2013. – 320 p.

25. How the Virus Got Out / The New York Times. URL: https://www.nytimes.com/interactive/2020/03/22/world/coronavirus-spread.html (дата звернення: 01.05.2025).

26. NSA Files: Decoded / The Guardian. URL: https://www.theguardian.com/world/interactive/2013/nov/01/snowden-nsa-files-surveillance-revelations-decoded (дата звернення: 01.05.2025).

27. Observable / інтерактивна платформа. URL: https://observablehq.com/ (дата звернення: 01.05.2025).

28. Gapminder / аналітична платформа. URL: https://www.gapminder.org/ (дата звернення: 01.05.2025).

29. Інтерактивна платформа Cesium / візуалізація 3D-даних. URL: https://cesium.com/ (дата звернення: 01.05.2025).

30. WebGL-лабораторія від NASA / інтерактивна візуалізація. URL: https://science.nasa.gov/eyes/ (дата звернення: 01.05.2025).

31. Rininsland A., Teller S. D3.js 4.x Data Visualization: Learn to visualize your data with JavaScript. – 3rd ed. – Birmingham: Packt, 2017. – 208 p.

32. Meeks E. D3.js in Action: Data Visualization with JavaScript. – 2nd ed. – Shelter Island: Manning Publications, 2018. – 366 p.

33. Iglesias M. Pro D3.js: Use D3.js to Create Maintainable, Modular, and Testable Charts. – New York: Apress, 2019. – 244 p.

34. Zhu N. Data Visualization with D3 4.x Cookbook. – 2nd ed. – Birmingham: Packt Publishing, 2017. – 380 p.

35. Murray S. Interactive Data Visualization for the Web. – 2nd ed. – Sebastopol: O'Reilly Media, 2017. – 270 p.

36. Matsuda K., Lea R. WebGL Programming Guide: Interactive 3D Graphics Programming with WebGL. – Boston: Addison-Wesley Professional, 2013. – 552 p.

37. Parisi T. WebGL: Up and Running. – 1st ed. – Sebastopol: O'Reilly Media, 2012. – 227 p.

38. WebGL Fundamentals / навчальний сайт. URL: https://webglfundamentals.org/ (дата звернення: 01.12.2024).

39. Belkin A., Gelernter N., Cidon I. The Risks of WebGL: Analysis, Evaluation and Detection / conference paper : веб-сайт. URL: https://dl.acm.org/doi/10.1007/978-3-030-29962-0_26 (дата звернення: 01.12.2024).

40. Бондаренко М.Ф., Хахаиов В.И., Лещинская И.А., Русакова Н.Е. Инфраструктура анализа логических ассоциативных отношений // Радиоэлектроника и информатика. – Харьков: ХНУРЭ, 2010. – №1.

41. Guzhov, V., Smelyakov, K. Free DBMSs Performance for an Inventory Management System based on Spring Boot // 2024 IEEE Open Conference of

Electrical, Electronic and Information Sciences (eStream). 2024. P. 1 - 5.

42. Vysotska V., Smelyakov K., Sharonova N., Filipov O., Kotelnykov R. Fast Color Images Clustering for Real-Time Computer Vision and AI System // CEUR Workshop Proceedings. – 2024. – P. 161–177.

43. Grynenko A., Turuta O., Kasheparov R., Kalynychenko O., Turuta O. Estimation and Visualization of Webcam Eye Tracking for Text Reading // CEUR Workshop Proceedings. – 2024. – P. 102–110.

44. Gruzdo I., Kyrychenko I., Tereshchenko G., Shanidze O. Analysis of Models Usability Methods Used on Design Stage to Increase Site Optimization // CEUR Workshop Proceedings. – 2023. – P. 387–409.

45. Vysotska V., Shubin I., Mezentsev M., Kobernyk K., Chetverikov G. Ukrainian Big Data: The Problem of Databases Localization // COLINS-2024: 8th International Conference on Computational Linguistics and Intelligent Systems. – 2024. – P. 122–133.

46. D3 / GitHub. URL: https://github.com/d3/d3 (дата звернення: 01.04.2025).

47. WebGL / GitHub. URL: https://github.com/KhronosGroup/WebGL (дата звернення: 01.04.2025).

48. Can I use / довідник підтримки веб-технологій. URL: https://caniuse.com/ (дата звернення: 01.04.2025).

49. Internet Explorer 11 has retired and is officially out of support - what you need to know / Windows Experience Blog. URL: https://blogs.windows.com/windowsexperience/2022/06/15/internet-explorer-11-has-retired-and-is-officially-out-of-support-what-you-need-to-know/ (дата звернення: 01.04.2025).

50. Статистика українського сегменту домену .UA / Hostmaster. URL: https://www.hostmaster.ua/UAstat/ (дата звернення: 01.05.2025).

51. GitHub-репозиторій з матеріалами до кваліфікаційної роботи URL:https://github.com/nponikarovska/2025_M_PI_IPZzdm_23_1_Ponikarovska_N_A (дата звернення: 02.06.2025).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ЗА НАУКОВИМИ НАПРЯМАМИ КЕРІВНИКА ТА НАУКОВЦІВ КАФЕДРИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

2. Rusakova N., Ponikarovska N. and Shirokopetleva M., Research on the Efficiency of Applying WebGL and D3.js in the Creation of Graphical Elements in Web Applications // 2025 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream), Vilnius, Lithuania, 2025, pp. 1-5, doi: 10.1109/eStream66938.2025.11016891.

3. Comparative analysis of WebGL and D3.js. N.Rusakova, N.Ponikarovska, M.Shirokopetleva, The International Conference “Problems of Infocommunications. Science and Technology” (PIC S&T′2024), Kharkiv, Ukraine, 2025, in press.

40. Бондаренко М.Ф., Хахаиов В.И., Лещинская И.А., Русакова Н.Е. Инфраструктура анализа логических ассоциативных отношений // Радиоэлектроника и информатика. – Харьков: ХНУРЭ, 2010. – №1.

41. Guzhov, V., Smelyakov, K. Free DBMSs Performance for an Inventory Management System based on Spring Boot // 2024 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream). 2024. P. 1 - 5.

42. Vysotska V., Smelyakov K., Sharonova N., Filipov O., Kotelnykov R. Fast Color Images Clustering for Real-Time Computer Vision and AI System // CEUR Workshop Proceedings. – 2024. – P. 161–177.

43. Grynenko A., Turuta O., Kasheparov R., Kalynychenko O., Turuta O. Estimation and Visualization of Webcam Eye Tracking for Text Reading // CEUR Workshop Proceedings. – 2024. – P. 102–110.

44. Gruzdo I., Kyrychenko I., Tereshchenko G., Shanidze O. Analysis of Models Usability Methods Used on Design Stage to Increase Site Optimization // CEUR Workshop Proceedings. – 2023. – P. 387–409.

45. Vysotska V., Shubin I., Mezentsev M., Kobernyk K., Chetverikov G. Ukrainian Big Data: The Problem of Databases Localization // COLINS-2024: 8th International Conference on Computational Linguistics and Intelligent Systems. – 2024. – P. 122–133.